

Piece-wise Linear Electrical Circuit Simulation



... circuit simulation
at the system level

User Manual

Version 3.3

How to Contact Plexim:

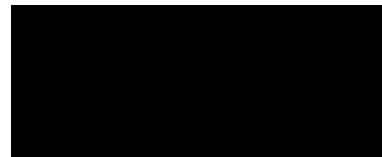
- | | |
|--|-------|
|  +41 44 445 24 10 | Phone |
|  +41 44 445 24 11 | Fax |
|  Plexim GmbH
Technoparkstrasse 1
8005 Zurich
Switzerland | Mail |
|  info@plexim.com | Email |
|  http://www.plexim.com | Web |

PLECS User Manual

© 2002–2012 by Plexim GmbH

The software PLECS described in this manual is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from Plexim GmbH.

PLECS is a registered trademark of Plexim GmbH. MATLAB, Simulink and Real-Time Workshop are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.



Contents

Contents	iii
Before You Begin	1
Installing the PLECS Blockset	1
Automatic Installation	1
Manual Installation on Microsoft Windows	1
Manual Installation on Mac OS X / Linux	2
Configuring PLECS	3
Installing Different Versions of the PLECS Blockset in Parallel	3
Uninstalling the PLECS Blockset	3
Installing PLECS Standalone	5
Installation on Microsoft Windows	5
Installation on Mac OS X	5
Installation on Linux	5
What's New in Version 3.3	6
Licensing	7
Network Licensing	7

1 Getting Started	9
Getting Started with the PLECS Blockset	9
A Simple Passive Network	9
Buck Converter	15
Getting Started with PLECS Standalone	18
A Simple Passive Network	18
Buck Converter	22
2 How PLECS Works	25
Modeling Dynamic Systems	25
System Equations	26
Block Diagrams	26
Physical Models	27
Simulating Dynamic Systems	27
Model Initialization	28
Model Execution	30
Sampled Data Systems	32
Sample Times	32
Multirate Systems	33
Troubleshooting	34
3 Using PLECS	35
Configuring PLECS	35
General	35
Libraries	36
Thermal	36
Scope Colors	37
Creating a New Circuit with the PLECS Blockset	37
Customizing the Circuit Block	37
Using the Library Browser	39
Components	41

Specifying Component Parameters	41
Displaying Parameters in the Schematic	42
Changing Component Names	42
Changing the Orientation of Components	42
Getting Component Help	42
Libraries	43
Creating a New Library in PLECS Blockset	43
Creating a New Library in PLECS Standalone	43
Creating a Library Reference	43
Updating a Library Reference	44
Breaking a Library Reference	44
Connections	45
Wires	45
Signals	45
Creating Branches	45
Annotations	45
Subsystems	46
Creating a Subsystem by Adding the Subsystem Block	46
Creating a Subsystem by Grouping Existing Blocks	46
Arranging Subsystem Terminals	47
Resizing a Subsystem Block	47
Placing the Subsystem Label	48
Masking Subsystems	49
Mask Icon	49
Mask Parameters	51
Mask Probe Signals	53
Mask Documentation	54
Unprotecting Masked Subsystems	55
Circuit Browser	56
Showing Masked Subsystems	56
PLECS Probe	57

Copying a Probe	58
Controlling Access to Circuits and Subcircuits	59
Encrypting Circuits and Subcircuits	59
Exporting Circuits for the PLECS Viewer	60
Exporting Schematics	61
Using the PLECS Scope	62
Getting Started	62
Zoom Operations	63
Changing Curve Properties	64
Spreading Signals	64
Cursors	65
Fourier Analysis	66
Saving a View	66
Adding Traces	67
Saving and Loading Trace Data	67
Scope Parameters	67
Printing and Exporting	67
Using the Fourier Analysis	68
Calculation Parameters	68
Display Parameters	69
Zoom, Export and Print	70
Calculation of the Fourier coefficients	70
Using the XY Plot	71
Time Range Window	71
Zoom, Save View, Export and Print	72
Simulation Parameters	73
PLECS Blockset Parameters	73
PLECS Standalone Parameters	75

4 Thermal Modeling	79
Heat Sink Concept	79
Implementation	80
Thermal Loss Dissipation	80
Semiconductor Losses	80
Ohmic Losses	84
Heat Sinks and Subsystem	84
Thermal Description Parameter	85
Assigning Thermal Data Sheets	85
Using Reference Variables	86
Thermal Library	88
Library Structure	88
Global and Local Data Sheets	88
Creating New Data Sheets	89
Browsing the Thermal Library	89
Thermal Editor	90
Editing Switching Losses	91
Editing Conduction Losses	91
Editing the Thermal Equivalent Circuit	91
Semiconductor Loss Specification	93
Single Semiconductor Switch Losses	93
Diode Losses	93
Losses of Semiconductor Switch with Diode	94
5 Magnetic Modeling	97
Equivalent circuits for magnetic components	97
Coupled inductors	98
Reluctance-resistance analogy	98
Permeance-capacitance analogy	100
Magnetic Circuit Domain in PLECS	101
Modeling Non-Linear Magnetic Material	102
Saturation Curves for Soft-Magnetic Material	103

6 Analysis Tools	105
Steady-State Analysis	105
Algorithm	105
Fast Jacobian Calculation for Thermal States	106
Limitations	107
Reference	107
AC Analysis	107
Impulse Response Analysis	108
Algorithm	108
Compensation for Discrete Pulse	108
Reference	109
Usage in PLECS Standalone	110
Steady-State Analysis	110
AC Sweep	111
Impulse Response Analysis	113
Extraction of State-Space Matrices	113
Application Example	114
Usage in the PLECS Blockset	116
Steady-State Analysis	116
AC Sweep / Loop Gain Analysis	118
Impulse Response Analysis	121
Extraction of State-Space Matrices	123
Application Example	124

7 C-Scripts	133
How C-Scripts Work	133
C-Script Functions	134
Modeling Discontinuities	136
Sample Time	138
User Parameters	140
Runtime Checks	141
C-Script Examples	141
A Simple Function – Times Two	141
Discrete States – Sampled Delay	142
Continuous States – Integrator	142
Event Handling – Wrapping Integrator	143
Piecewise Smooth Functions – Saturation	144
Multiple Sample Times – Turn-on Delay	146
C-Script Macros	148
8 Simulation Scripts	151
Command Line Interface in PLECS Blockset	151
Simulation Scripts in PLECS Standalone	155
Overview of PLECS Scripting Extensions	156
Example Script	158
XML-RPC Interface in PLECS Standalone	159
Establishing an XML-RPC Connection to PLECS	159
Overview of XML-RPC Commands	160
Example Script	162
Scripted Simulation and Analysis Options	162

9 Code Generation with Real-Time Workshop	167
Code Generation Targets	167
Rapid Simulation Target	168
Deploying Rapid Simulation Executables	169
Tunable Circuit Parameters in Rapid Simulations	170
Real-Time Target	170
Unsupported Components	171
Maximum Number of Switches	171
Limiting the Code Size	171
Natural Commutation	172
Code Generation Options	172
10 Components by Category	175
System	175
Control	176
Sources	176
Math	176
Continuous	177
Delays	177
Discontinuous	177
Discrete	178
Filters	178
Functions & Tables	178
Logical	179
Modulators	179
Transformations	180
Small Signal Analysis	180
Electrical	181
Sources	181
Meters	181
Passive Components	181

Power Semiconductors	182
Switches	183
Transformers	183
Machines	184
Converters	184
Electronics	185
Thermal	185
Magnetic	186
Additional Simulink Blocks	186
11 Component Reference	189
1D Look-Up Table	190
2D Look-Up Table	191
2-Pulse Generator	192
3D Look-Up Table	193
3-Phase Overmodulation	194
6-Pulse Generator	195
Abs	196
Ambient Temperature	197
Air Gap	198
Ammeter	199
Blanking Time	200
Blanking Time (3-Level)	201
Breaker	202
Brushless DC Machine	203
Brushless DC Machine (Simplified)	207
C-Script	211
Capacitor	213
Clock	215
Combinatorial Logic	216
Comparator	217

Configurable Subsystem	218
Constant	219
Constant Heat Flow	220
Constant Temperature	221
Controlled Heat Flow	222
Controlled Temperature	223
Current Source (Controlled)	224
Current Source AC	225
Current Source DC	226
D Flip-flop	227
DC Machine	228
Dead Zone	230
Delay	231
Diode	232
Diode with Reverse Recovery	234
Diode Rectifier (3ph)	237
Discrete Fourier Transform	238
Discrete Mean Value	239
Discrete RMS Value	240
Discrete Total Harmonic Distortion	241
Discrete Transfer Function	242
DLL	244
Double Switch	248
Edge Detection	249
Electrical Ground	250
Electrical Label	251
Electrical Port	252
Flux Rate Meter	253
Fourier Series	254
Function	255
Gain	256

GTO	257
GTO (Reverse Conducting)	259
Heat Flow Meter	261
Heat Sink	262
Hit Crossing	263
Hysteretic Core	264
Ideal 3-Level Converter (3ph)	266
Ideal Converter (3ph)	267
Ideal Transformer	268
IGBT	269
IGBT 3-Level Converter (3ph)	271
IGBT Converter (3ph)	273
IGBT with Diode	275
IGBT with Limited di/dt	277
IGCT (Reverse Blocking)	281
IGCT (Reverse Conducting)	283
Induction Machine	285
Induction Machine (Open Stator Windings)	290
Induction Machine (Squirrel-Cage)	293
Induction Machine with Saturation	296
Inductor	302
Integrator	304
JK Flip-flop	306
Leakage Flux Path	308
Linear Core	309
Linear Transformer (2 Windings)	310
Linear Transformer (3 Windings)	312
Logical Operator	314
Magnetic Permeance	315
Magnetic Port	316
Magnetic Resistance	317

Math Function	318
Memory	319
Meter (3-Phase)	320
Minimum / Maximum	321
MMF Meter	322
MMF Source (Constant)	323
MMF Source (Controlled)	324
Monoflop	325
MOSFET	326
MOSFET Converter (3ph)	328
MOSFET with Diode	329
MOSFET with Limited di/dt	331
Moving Average	333
Mutual Inductor	334
Mutual Inductance (2 Windings)	336
Mutual Inductance (3 Windings)	338
Op-Amp	340
Op-Amp with Limited Output	341
Peak Current Controller	342
Periodic Average	343
Periodic Impulse Average	344
Permanent Magnet Synchronous Machine	345
Pi-Section Line	349
Piece-wise Linear Resistor	351
Polar to Rectangular	353
Product	354
Pulse Delay	355
Pulse Generator	356
Quantizer	357
Ramp	358
Rate Limiter	359

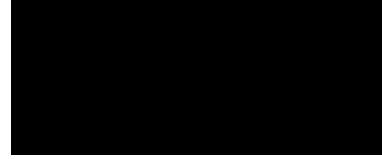
Rectangular to Polar	360
Relational Operator	361
Relay	362
Resistor	363
Rounding	364
Saturable Capacitor	365
Saturable Core	367
Saturable Inductor	370
Saturable Transformers	372
Saturation	374
Sawtooth PWM	375
Sawtooth PWM (3-Level)	377
Scope	379
Set/Reset Switch	380
Signal Demultiplexer	381
Signal From	382
Signal Goto	383
Signal Import	384
Signal Multiplexer	386
Signal Outport	387
Signal Selector	388
Signal Switch	389
Signum	390
Sine Wave	391
Small Signal Gain	392
Small Signal Perturbation	393
Small Signal Response	394
Space Vector Modulator	395
SR Flip-flop	399
State Space	400
Step	401

Contents

Subsystem	402
Sum	403
Switch	404
Switched Reluctance Machine	405
Symmetrical PWM	409
Symmetrical PWM (3-Level)	411
Synchronous Machine (Round Rotor)	413
Synchronous Machine (Salient Pole)	418
Thermal Capacitor	424
Thermal Chain	425
Thermal Ground	426
Thermal Port	427
Thermal Resistor	428
Thermometer	429
Thyristor	430
Thyristor Rectifier/Inverter	432
Thyristor with Reverse Recovery	433
To File	435
Transfer Function	436
Transformation 3ph->RRF	438
Transformation 3ph->SRF	439
Transformation RRF->3ph	440
Transformation RRF->SRF	441
Transformation SRF->3ph	442
Transformation SRF->RRF	443
Transformers (3ph, 2 Windings)	444
Transformers (3ph, 3 Windings)	447
Transport Delay	450
TRIAC	451
Triangular Wave Generator	453
Trigonometric Function	454

Triple Switch	455
Turn-on Delay	456
Variable Capacitor	457
Variable Inductor	460
Variable Magnetic Permeance	464
Variable Resistor with Constant Capacitor	466
Variable Resistor with Constant Inductor	467
Variable Resistor with Variable Capacitor	468
Variable Resistor with Variable Inductor	470
Voltage Source (Controlled)	472
Voltage Source AC	473
Voltage Source AC (3-Phase)	474
Voltage Source DC	475
Voltmeter	476
Winding	477
Wire Multiplexer	478
Wire Selector	479
XY Plot	480
Zener Diode	481
Zero Order Hold	482
12 Additional Simulink Blocks	483
AC Sweep	484
Discrete Analysis	487
Impulse Response Analysis	488
Loop Gain Analysis	490
Modulators	491
Steady-State Analysis	492
Timer	494
Transformations	495

Contents



Before You Begin

Installing the PLECS Blockset

Installing the PLECS Blockset on your system is easy. You do not need to have system administrator permissions. Since the PLECS Blockset requires MATLAB and Simulink make sure these programs are installed on your system.

Automatic Installation

The “Blockset” directory of the PLECS CD-ROM contains an M-file `installplecs.m`. When you execute this file from MATLAB it will automatically detect your platform, optionally search online for newer PLECS versions, install PLECS to a folder of your choice and set up your MATLAB path. You can use the installer both for a fresh installation and to update an existing installation.

Manual Installation on Microsoft Windows

- 1** Run the installer executable by double-clicking it. PLECS can be installed for the current user or all users of a machine. To install PLECS for all users the installer must be executed with administrator privileges.
- 2** If you have purchased a license for the full version you will have received a license file `license.dat`. Copy this file to your harddisk. During the installation the installer allows you to browse to the license file. As an alternative you can choose to install the license for the PLECS Viewer.

The file can also be copied into the installation directory after the installation has completed. PLECS searches the license file in the same directory where `plecs.m` is located.

- 3** The PLECS installer will add PLECS to the MATLAB search path. If an error is encountered (e.g. because of insufficient access privileges) the changes to the MATLAB path can also be done manually. In this case add the installation directory and the subdirectory `demos` to your search path using the Path Browser in MATLAB. The Path Browser is found under the menu item “File → Set Path → Add Folder”.
- 4** If you previously had installed an older version of PLECS execute

```
plecsclear  
rehash toolboxcache
```

in the MATLAB command line.

You can always re-run the installation assistant to change the license file or the MATLAB path. To do this, start `plecs.exe` in the subdirectory `bin\win32`.

Manual Installation on Mac OS X / Linux

- 1** Untar with

```
tar -xzf filename.tar.gz
```

in a directory of your choice. This will create a new sub-directory named `plecs` containing the required files.

- 2** If you have purchased a license for the full version you will have received a license file `license.dat`. Copy this file into the just created directory named `plecs`.

If you would like to install the PLECS Viewer, copy the file `viewerlicense.dat` from `plecs/private` into the parent directory `plecs` and rename it to `license.dat`.

- 3** In MATLAB, add the new directory `plecs` and the subdirectory `demos` to your search path. Use the Path Browser under the menu item “File → Set Path → Add Folder”. Alternatively, edit directly the file `pathdef.m` in the directory `matlabroot/toolbox/local/`. If you do not have file system permission to modify the file `pathdef.m` add the commands

```
addpath('plecs_directory');
addpath('plecs_directory/demos');
```

to the file `~/matlab/startup.m`. (In case the file does not exist create an empty file `startup.m` in the subdirectory `matlab` of your home directory.)

- 4** If you previously had installed an older version of PLECS execute

```
plecsclear
rehash toolboxcache
```

in the MATLAB command line.

Configuring PLECS

For information about setting global configuration options for PLECS see “Configuring PLECS” (on page 35).

Installing Different Versions of the PLECS Blockset in Parallel

If you want to keep different versions of PLECS installed in parallel on one computer, you must ensure that only one version is on your MATLAB path at any time during a MATLAB session. Otherwise, loss of data may occur. Before changing the MATLAB path, be sure to clear the currently loaded PLECS module by entering `plecsclear` at the MATLAB command prompt. As an additional precaution you should restart MATLAB after the change.

Uninstalling the PLECS Blockset

Uninstalling the PLECS Blockset is as easy as installing it.

- 1** Locate the directory where PLECS is installed by entering

```
which plecs
```

in the MATLAB command line.

- 2** Remove the PLECS directory and its subdirectory `demos` from the search path. Depending on how the directories were added to the path during installation, this is done using the Path Browser or by editing the file

`pathdef.m` in the directory `matlabroot/toolbox/local/` or the file `~/matlab/startup.m`.

- 3** Quit MATLAB.
- 4** On Windows, deinstall PLECS Blockset by choosing the appropriate entry in the Windows control panel. On Mac OS X and Linux just delete the PLECS directory.

Installing PLECS Standalone

Installing PLECS on your system is easy. You do not need to have system administrator permissions.

Installation on Microsoft Windows

- 1** Run the installer executable by double-clicking it. PLECS can be installed for the current user or all users of a machine. To install PLECS for all users the installer must be executed with administrator privileges.
- 2** If you have purchased a license for the full version you will have received a license file `license.dat`. During the installation the installer asks you to copy the file into the installation directory. The file can also be copied into the installation directory after the installation has completed. PLECS searches the license file in the same directory where `plecs.exe` is located.

Installation on Mac OS X

- 1** Open the downloaded disk image by double-clicking it.
- 2** Copy PLECS to the Application folder.
- 3** If you have purchased a license for the full version you will have received a license file `license.dat`. Copy this file to your harddisk. When PLECS is started for the first time it will offer to install the license file.

Installation on Linux

- 1** Untar with

```
tar -xzf filename.tar.gz
```

in a directory of your choice. This will create a new sub-directory named `plecs` containing the required files.

- 2** If you have purchased a license for the full version you will have received a license file `license.dat`. Copy this file into the `plecs` directory.

What's New in Version 3.3

The following list describes new features and enhancements added in PLECS 3.3:

- The Magnetic library allows the user to include magnetic designs for inductors and transformers in the simulation model. See “Magnetic Modeling” (on page 97).
- Color and other curve properties of the PLECS Scope can now be customized. See section “Scope Colors” (on page 37) and “Changing Curve Properties” (on page 64) for details.
- Trace data can now be saved and loaded into a PLECS Scope. See “Saving and Loading Trace Data” (on page 67) for details.
- Multiple overlaying signals can automatically be separated in the PLECS Scope. See “Spreading Signals” (on page 64) for details.
- The state space matrices can now be accessed also in PLECS Standalone. See section “Extraction of State-Space Matrices” (on page 113) for details.
- New components were added to the PLECS library: the Moving Average (on page 333), the Periodic Average (on page 343) and the Periodic Impulse Average (on page 344).
- The behavior of a PLECS Probe during copy operations has changed. See “Copying a Probe” (on page 58).
- PLECS Standalone for Mac OS X is now a 64-bit application. Existing shared libraries for use with the DLL block must be recompiled for 64-bit support.

Licensing

When you install the full version of PLECS you must have a valid license file `license.dat`. This file will be sent to you by email when you purchase a license for PLECS. Copy the file `license.dat` into the directory where you have installed PLECS.

If the license file is not present or contains invalid data you will still be able to open or save Simulink models containing PLECS circuits. However, you cannot modify a circuit or run a simulation.

Note The PLECS Blockset for Simulink scans the license file only once when the module is loaded by MATLAB. Therefore, if you reinstall the license file you need to clear the PLECS module before the changes can become effective. You can do this by entering `plecsclear` at the MATLAB command prompt.

Network Licensing

If you purchase one or more concurrent licenses for PLECS, the license server program FLEXlm is employed to control access to PLECS. FLEXlm is a product of Macrovision Corporation. The license sent to you must be installed on the license server. This file contains information that identifies the computer running the license manager and specifies the number of concurrent licenses you have purchased.

On the client computer(s), you need to use a text editor to create the license file `license.dat` in the PLECS directory with the following content:

```
SERVER hostname ANY  
USE_SERVER
```

where *hostname* is the name of the computer running the license manager.

PLECS tries to obtain a license from the server the first time you load a model or library containing a PLECS circuit. If the license is not granted – either because the server is down or unreachable or because the licensed number of concurrent users is already reached – PLECS will fall back to an *unlicensed mode*. In this mode you cannot modify a circuit or run a simulation; saving a model is still possible. In order to retry to obtain a license you first

Before You Begin

need to close all models (including the PLECS library). Once granted, a license is returned to the server when you close the last model containing a PLECS circuit.

If the connection to the license server is lost *after* you have obtained a license, PLECS will *temporarily* switch to the unlicensed mode. Upon successful reconnection to the server, PLECS will switch back to normal operation.

Getting Started

Let us have a quick tour and see how PLECS is used. Our aim is to show the essential elements of PLECS in real applications without regarding all the details, rules, and exceptions. At this stage, we are not trying to be complete. We want to get you as soon as possible to the point where you can set up useful applications. Many of the details are not necessary at the beginning and can be studied later.

The following section addresses users of the PLECS Blockset for Simulink. If you are using the stand-alone version of PLECS please continue with section “Getting Started with PLECS Standalone” (on page 18).

Getting Started with the PLECS Blockset

To access PLECS you simply need to enter `plecslib` in the MATLAB command line. This will bring up a Simulink model that contains a generic PLECS block named “Circuit” and various component libraries. In the libraries you find electrical components, from which you can create your circuits. Alternatively, you may access the PLECS toolbox by opening it in the Simulink library browser.

A Simple Passive Network

The only way to become familiar with a new program is by using it. For this reason we are presenting here two example circuits that you can reconstruct on your computer. The examples are based on each other, since the features of PLECS will be explained step by step.

The first electrical system we are going to model is a simple RLC network as shown in Fig. 1.1. A capacitor is charged by a DC voltage source via an RL-branch and its voltage is monitored with a voltmeter.

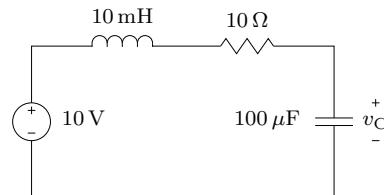


Figure 1.1: Simple RLC network

In order to enter the circuit in PLECS we have to open a new Simulink model. Into the model window we copy the block “Circuit” from the PLECS library by dragging it with the mouse. Our Simulink model should now look like Fig. 1.2.

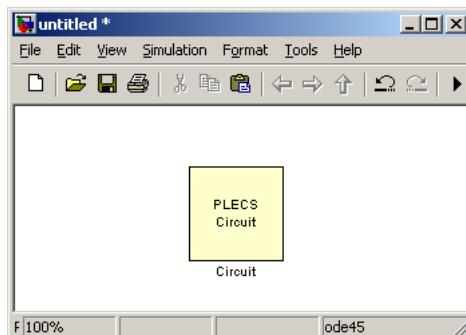


Figure 1.2: Simulink model

Components

A double-click on the PLECS block will open an empty schematic window with a menu bar quite similar to the one of a Simulink window. The components required for our circuit must be copied into this window from the components libraries. Like in Simulink, this is done by dragging them with the mouse. If you want to copy components already placed in the window hold down the **Ctrl** control key or use the right mouse button. The components that you need for the RLC network can be found in the library “Electrical” in the sub-libraries “Sources”, “Meters” and “Passive Components”.

After you have copied all components the schematic window should look like Fig. 1.3. If not, move the components with the left mouse button. To rotate selected components press **Ctrl-R**, to flip them horizontally press **Ctrl-F**. All these functions can also be accessed via the menu bar.

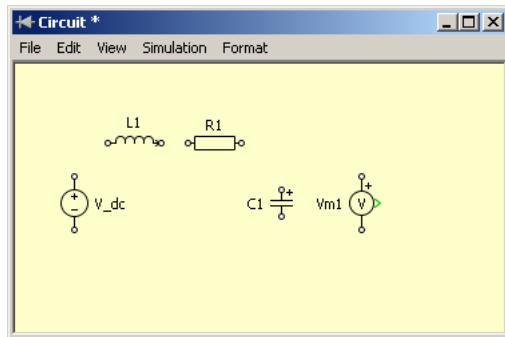


Figure 1.3: PLECS schematic

Note You cannot place Simulink objects in a PLECS schematic and vice versa since both programs do not share the same Graphical User Interface.

Connections

The unconnected electrical terminals of a component are marked with little hollow circles. If we bring the mouse pointer close to such a terminal the pointer shape changes from an arrow to a cross. We now can drag a connection to another component by holding the left mouse button down. When we approach another terminal or an existing connection the pointer shape changes into a double cross. As soon as we release the mouse button an electrical connection will be created.

For drawing a branch connection place the mouse pointer on an existing connection where you want the branch to start. With the right mouse button or with the left mouse button while holding down the **Ctrl** key you can create a connection from there to the desired destination.

Component Properties

Each component is identified by a unique name, which is chosen automatically. You may change it as you wish by double-clicking on it in the schematic. The name is intended only for documentation purposes and does not affect the simulation. Of greater importance are the parameters that determine, for example, the inductance of an inductor, the capacity of an capacitor, or the voltage of a DC voltage source. A double-click on the component icon opens a dialog box in which you can set these parameters. Fig. 1.4 shows the dialog box for an inductor.

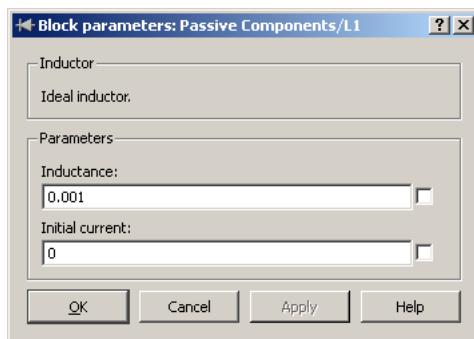


Figure 1.4: Inductor dialog box

If you want selected parameters to be displayed in the schematic, you must check the check box on the right side of the edit field. For reasons of clarity we prefer to display only the most important parameters of a component.

Units

Like Simulink PLECS does not know anything about units. It is your responsibility that variables are scaled correctly. For power electronics we recommend the use of SI quantities. However, if you want to employ PLECS for the simulation of power systems it may be more appropriate to work with “per unit” quantities.

For every component enter the values according to the schematic in Fig. 1.1. In the dialog boxes of the inductor and the capacitor you can additionally set the initial current resp. the initial voltage. Please leave both values at zero.

Signals

Up to now our electrical circuit lacks a connection with the Simulink environment. You will notice this from the fact that the PLECS block in Simulink does not have inputs or outputs. In order to add inputs and outputs we must copy the respective port blocks from the library “System” into the schematic. In our case we want to access in Simulink the voltage measured by the voltmeter. Therefore, we need the “Signal Outport” block that exports a signal into the parent system.

Signals in PLECS correspond to the connections between Simulink blocks. They provide unidirectional information interchange between components and with Simulink.

Connect the output of the voltmeter with the input of the port block. In Simulink, connect a Scope to the output of the PLECS block and start the simulation. In order to see something of the more interesting part of the simulation you probably need to set the stop time to 0.1. By this time you should have something like Fig. 1.5 and Fig. 1.6 on your screen.

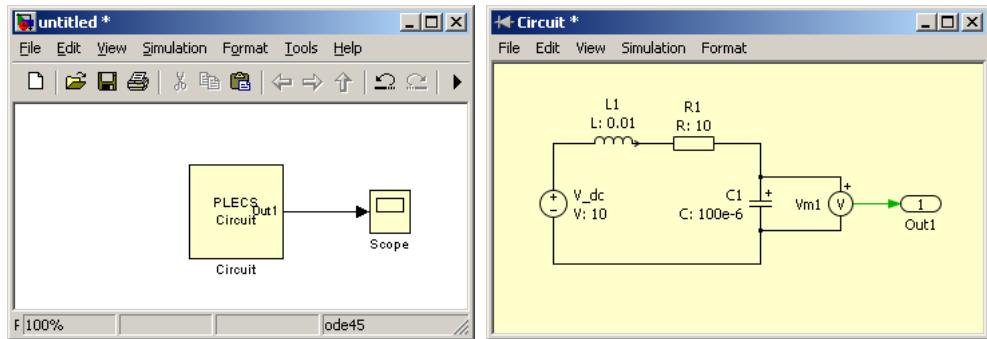


Figure 1.5: Complete model

Adding More Measurements

If you want to measure other quantities in the circuit, simply add the required voltmeters and ammeters. The measured signals can be exported to Simulink with additional port blocks. Alternatively you can bundle the measured signals into a vector by using the multiplexer for signals “Signal Multiplexer” from the library “System”.

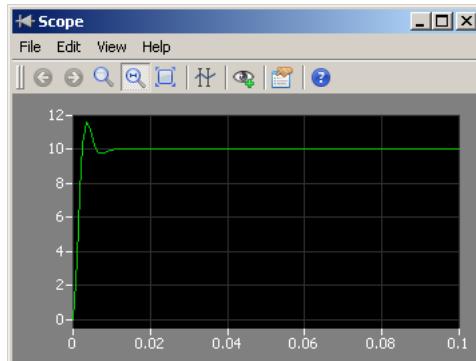


Figure 1.6: Simulation result

You can also add scopes in the PLECS schematic directly. The "Scope" block can be found in the library "System".

Importing Signals

You have already learned how to export signals from the electrical circuit to Simulink via the output block. In the same manner you can also import signals from Simulink into your circuit, usually to control sources.

Let us see how the capacitor in our example charges and discharges if we apply a pulsed voltage. In the schematic we replace the DC voltage source by a controlled one. Copy the input block "Signal Import" into the schematic and connect it to the voltage source. The PLECS block in Simulink now also has an input terminal. Any Simulink signal that you connect to this terminal will be translated into a voltage in the electrical circuit. In Fig. 1.7 we used a pulse generator with a period of 0.04 sec and an amplitude of 10.

The signal generated by the pulse generator is discrete, i.e. its value changes abruptly. Normally, the PLECS Scope would determine the signal type automatically and display vertical slopes. In this case, however, the discrete signal coming from the pulse generator is multiplexed with a continuous signal before reaching the Scope. In order to avoid trapezoidal curves, the signal type must be set manually to "discrete" in the Data window of the Scope (see Fig. 1.8).

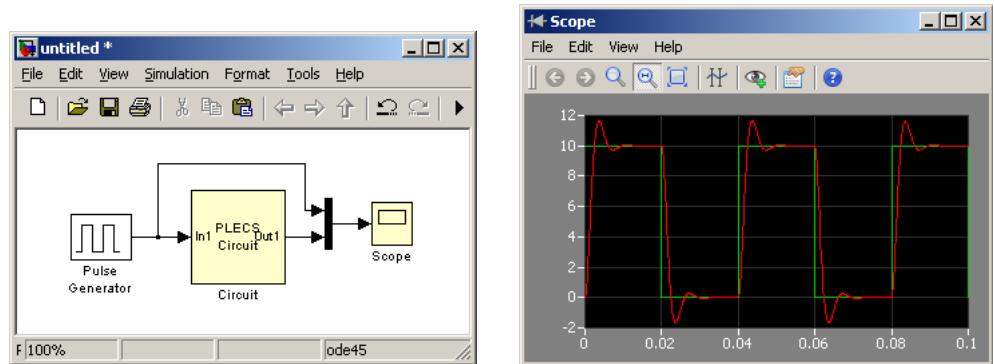


Figure 1.7: RLC network with a pulsed voltage source

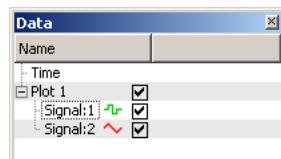


Figure 1.8: Data window of the PLECS Scope

Buck Converter

In the next example we will introduce the concept of ideal switches, which distinguishes PLECS from other simulation programs. It will be shown how switches are controlled, i.e. either by voltages and currents in the system or by external signals.

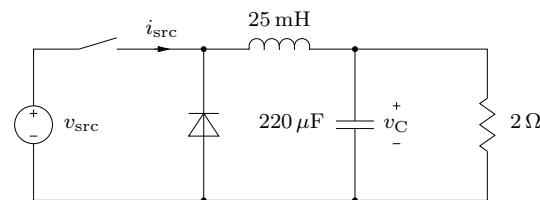


Figure 1.9: Schematic of buck converter

Switches

In the buck converter outlined in Fig. 1.9 we will model the transistor as an entirely controllable switch and bear in mind that it may conduct current only in one direction. We also need a free-wheeling diode. The diode is a switch that closes as the voltage across it becomes positive, and opens as the current through it becomes negative.

The diode can be found in the library “Electrical / Power Semiconductors” and the switch in the library “Electrical / Switches”. All components in these libraries are based on ideal switches that have zero on-resistance and infinite off-resistance. They open and close instantaneously. In some components like the diode you may add a forward voltage or a non-zero on-resistance. If you are unsure about these values leave them at zero.

In order to control the switch in our buck converter we import another signal from Simulink and connect it to the switch. The switch will close upon a non-zero signal and open when the signal goes back to zero.

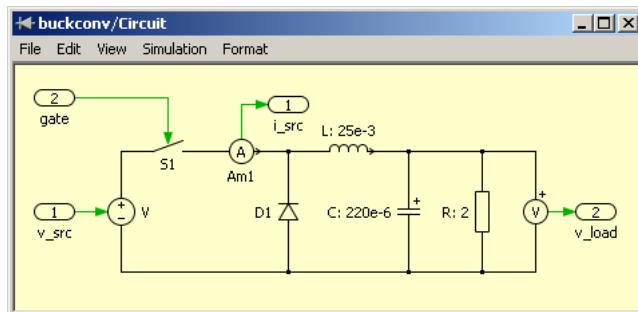


Figure 1.10: Electrical part of buck converter

By now you should be able to model the electrical part of the buck converter as shown in Fig. 1.10. For the buck converter we will implement a hysteresis type control that keeps the capacitor voltage roughly in a ± 0.2 V band around 6 V. To make things a bit more interesting we apply a step change from 12 V down to 8 V to the input voltage during the simulation.

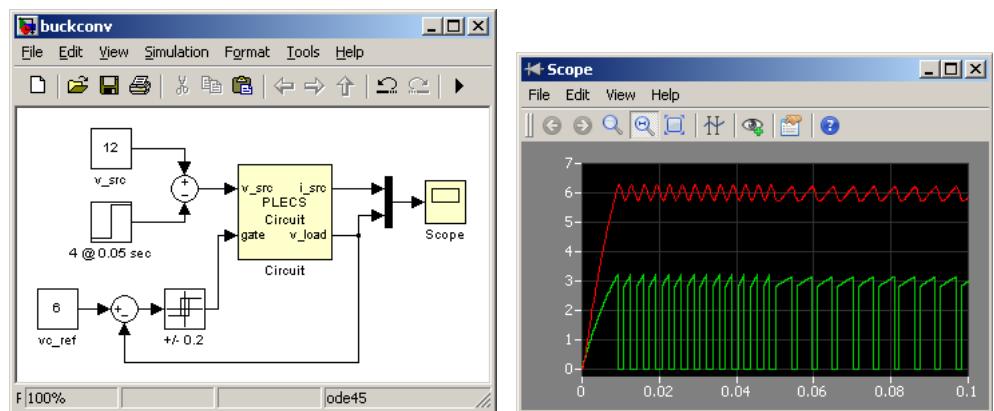


Figure 1.11: Simulation of buck converter with hysteresis control

Getting Started with PLECS Standalone

The only way to become familiar with a new program is by using it. For this reason we are presenting here two example circuits that you can reconstruct on your computer. The examples are based on each other, since the features of PLECS will be explained step by step.

After starting PLECS the PLECS Library browser is displayed. In the libraries you find various components from which you can create your circuits. You can browse through the available libraries and see which components are available.

A Simple Passive Network

The first electrical system we are going to model is a simple RLC network as shown in Fig. 1.12. A capacitor is charged by a DC voltage source via an RL-branch and its voltage is monitored with a voltmeter.

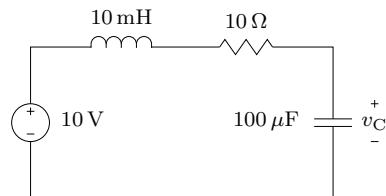


Figure 1.12: Simple RLC network

In order to enter the circuit in PLECS we have to open a new PLECS model. This is done by selecting “New Model” from the “File” Menu in the Library Browser.

Components

The components required for our circuit must be copied into this window from the Library Browser. This is done by dragging them with the mouse. If you want to copy components already placed in the window hold down the **Ctrl** control key or use the right mouse button.

The electrical components that you need for the RLC network can be found in in the library “Electrical” in the sub-libraries “Sources”, “Meters” and “Passive Components”. The scope is located in the library “System”. Instead of

browsing for the components you can also search for them by entering the first letters of the component you need in the search bar. For example, typing sc shows you the scope, res all available resistors etc.

After you have copied all components the schematic window should look like Fig. 1.13. If not, move the components with the left mouse button. To rotate selected components press **Ctrl-R**, to flip them horizontally press **Ctrl-F**. All these functions can also be accessed via the menu bar.

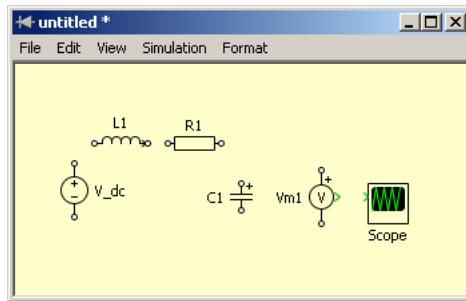


Figure 1.13: PLECS schematic

Connections

The unconnected electrical terminals of a component are marked with little hollow circles. If we bring the mouse pointer close to such a terminal the pointer shape changes from an arrow to a cross. We now can drag a connection to another component by holding the left mouse button down. When we approach another terminal or an existing connection the pointer shape changes into a double cross. As soon as we release the mouse button an electrical connection will be created.

For drawing a branch connection place the mouse pointer on an existing connection where you want the branch to start. With the right mouse button or with the left mouse button while holding down the **Ctrl** key you can create a connection from there to the desired destination.

Component Properties

Each component is identified by a unique name, which is chosen automatically. You may change it as you wish by double-clicking on it in the schematic. The name is intended only for documentation purposes and does not affect the

simulation. Of greater importance are the parameters that determine, for example, the inductance of an inductor, the capacity of an capacitor, or the voltage of a DC voltage source. A double-click on the component icon opens a dialog box in which you can set these parameters. Fig. 1.14 shows the dialog box for an inductor.

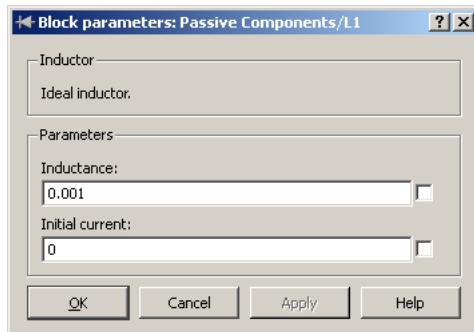


Figure 1.14: Inductor dialog box

If you want selected parameters to be displayed in the schematic, you must check the check box on the right side of the edit field. For reasons of clarity we prefer to display only the most important parameters of a component.

Units

PLECS does not know anything about units. It is your responsibility that variables are scaled correctly. For power electronics we recommend the use of SI quantities. However, if you want to employ PLECS for the simulation of power systems it may be more appropriate to work with “per unit” quantities.

For every component enter the values according to the schematic in Fig. 1.12. In the dialog boxes of the inductor and the capacitor you can additionally set the initial current resp. the initial voltage. Please leave both values at zero.

Signals

In addition to the electrical connections (wires) that are used to connect electrical components PLECS also makes use of unidirectional signals. The signals are painted in green and have an arrowhead to indicate their direction. In the RLC example a signal connects the output terminal of the voltmeter to the input terminal of the scope.

PLECS uses signals to carry non-electrical information like measurement values or triggering pulses for switches. Signals can be used in calculations and displayed in a scope. Electrical connections cannot be fed into a scope directly, you always have to use a volt- or ammeter to convert the electrical quantities into a signal first.

By this time your model should look similar to Fig. 1.15. To start the simulation, press **Ctrl-T** or select “Start” from the “Simulation” menu. In order to see something of the more interesting part of the simulation you need to set the stop time to 0.1. To do this, open the Simulation Parameters dialog by clicking the corresponding menu entry in the “Simulation” menu or press **Ctrl-E**.

You should now get the simulation results shown in below.

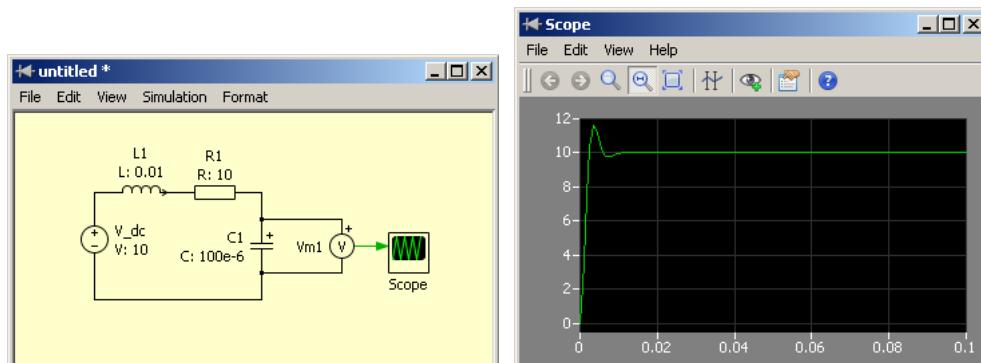


Figure 1.15: Complete model and simulation result

Adding Control Blocks

To enhance our model we would like to add some dynamic behavior into our static electrical model. Let us see how the capacitor in our example charges and discharges if we apply a pulsed voltage. In the schematic we replace the DC voltage source by a controlled one. The input of the voltage source can be any signal generated from one of the control blocks in PLECS. In Fig. 1.16 we used a pulse generator with a period of 0.04 sec and an amplitude of 10 to control the voltage source.

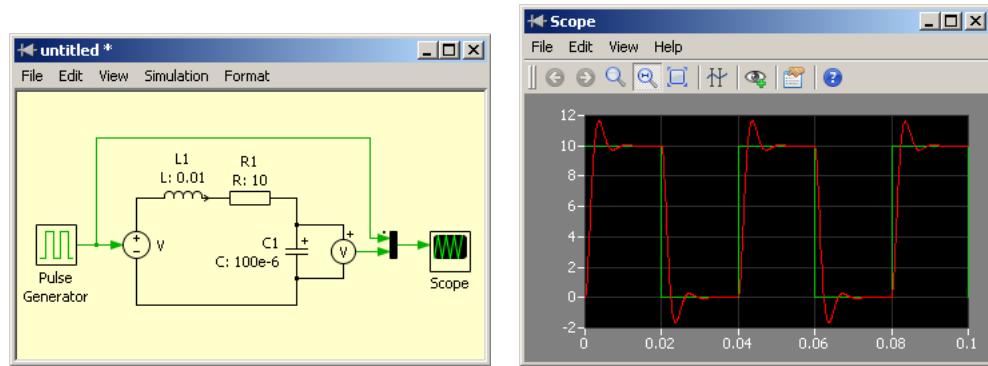


Figure 1.16: RLC network with a pulsed voltage source

Buck Converter

In the next example we will introduce the concept of ideal switches, which distinguishes PLECS from other simulation programs. It will be shown how switches are controlled, i.e. either by voltages and currents in the system or by external signals.

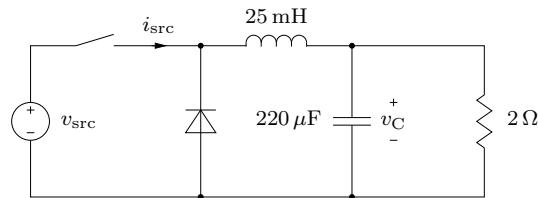


Figure 1.17: Schematic of buck converter

Switches

In the buck converter outlined in Fig. 1.17 we will model the transistor as an entirely controllable switch and bear in mind that it may conduct current only in one direction. We also need a free-wheeling diode. The diode is a switch that closes as the voltage across it becomes positive, and opens as the current through it becomes negative.

The diode can be found in the library “Electrical / Power Semiconductors” and the switch in the library “Electrical / Switches”. All components in these li-

ibraries are based on ideal switches that have zero on-resistance and infinite off-resistance. They open and close instantaneously. In some components like the diode you may add a forward voltage or a non-zero on-resistance. If you are unsure about these values leave them at zero.

The switch is controlled by an external signal. It will close upon a non-zero input and open when the signal goes back to zero.

We start with the electrical part of the buck converter first. By now you should be able to model it as shown in Fig. 1.18.

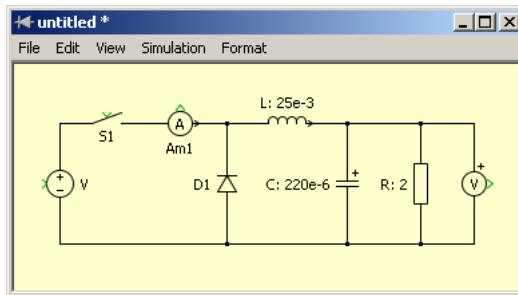


Figure 1.18: Electrical part of buck converter

Subsystems

We'd also like to separate the electrical part from the control part. This has no effect on the simulation result but makes the whole system more structured. Once you have completed the circuit from Fig. 1.18, select all components (either by clicking on an empty space in the upper left corner of the schematic and dragging a frame to the lower right corner, or by pressing **Ctrl-A**). Now create a new subsystem by selecting “Create Subsystem” from the “Edit” menu or by pressing **Ctrl-G**. The electrical components are now in a new subsystem “Sub”. You can rename it to something more meaningful, e.g. “Circuit” and change the icon size by dragging one of the selected corners. You can also move the name label to another position by clicking and dragging it to the borders or the corners of the icon. Now your system should look similar to Fig. 1.19.

To connect the subsystem to the outer schematic we need to place ports into it. Drag two Signal Imports and two Signal Exports into the subsystem schematic and connect them to the voltage source, the switch, the volt- and

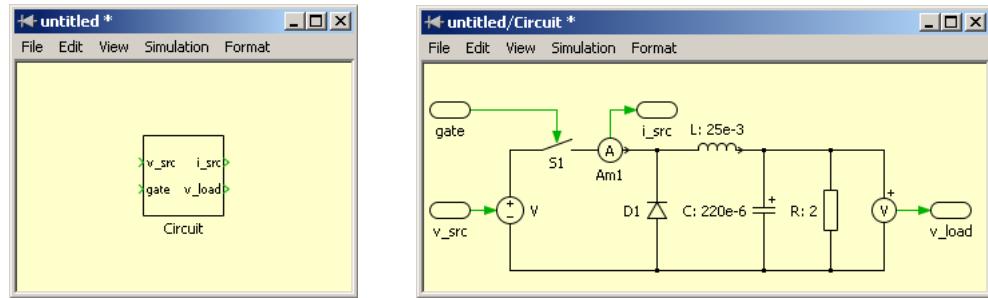


Figure 1.19: Electrical Subsystem

the ammeter respectively. Note that a new terminal appears in the subsystem icon for each port that you drag into the subsystem schematic.

For the buck converter we will implement a hysteresis type control that keeps the capacitor voltage roughly in a ± 0.2 V band around 6 V. To make things a bit more interesting we apply a step change from 12 V down to 8 V to the input voltage during the simulation.

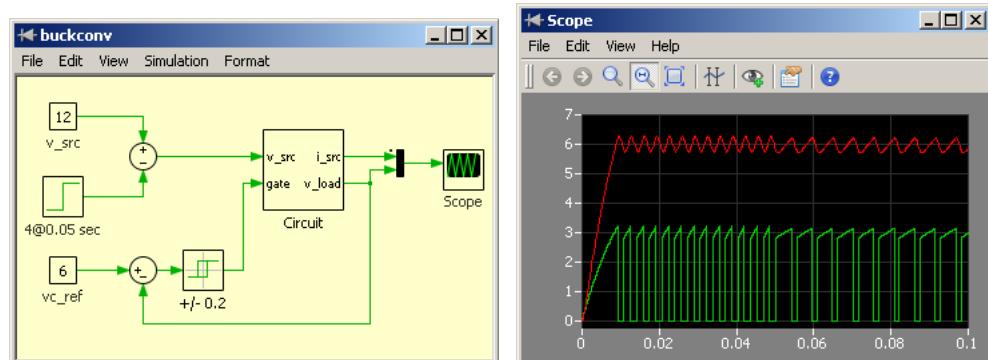


Figure 1.20: Simulation of buck converter with hysteresis control

Demo Models

Now that you've built your first own models in PLECS it may be worthwhile to take a look at the demo models that come with PLECS. Open the demo model browser by selecting "Demo Models" from the "View" Menu.

2

How PLECS Works

PLECS is a software package for modeling and simulating dynamic systems. As with any other software package, in order to make the best use of it you should have a basic understanding of its working principles. Before delving into the question how PLECS works, however, it is worthwhile to distinguish between the terms *modeling* and *simulation*.

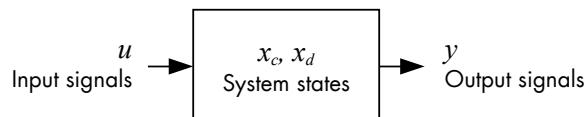
The term *modeling* refers to the process of extracting knowledge from the system to be simulated and representing this knowledge in some formal way. The second part – i.e. the representation of knowledge – can be more or less straightforward depending on the formalism used. PLECS offers three different formalisms – equations (implemented as C-code), block diagrams and physical models – that can be used in the same modeling environment. They are described in the following section.

The term *simulation* refers to the process of performing experiments on a model in order to predict how the real system would behave under the same conditions. More specifically, in the context of PLECS, it refers to the computation of the trajectories of the model's states and outputs over time by means of an *ordinary differential equation* (ODE) solver. This is described in the second section.

Modeling Dynamic Systems

A system can be thought of as a black box as depicted below. The system does not exchange energy with its environment but only information: It accepts input signals u , and its reactions can be observed by the output signals y .

A system can have internal state variables that store information about the system's past and influence its current behavior. Such state variables can be continuous, i.e. they are governed by differential equations, or discrete, i.e.



they change only at certain instants. An example of a continuous state variable is the flux or current of an inductor; an example of a discrete state variable is the state of a flip flop.

System Equations

One way to describe a system is by mathematical equations. Typical system equations are listed below:

- An output function describes the system's outputs in terms of the current time, the system's inputs and its internal states.
- If the system has discrete states, an update function determines if and how they change at a given time for the current inputs and
- If the system has continuous states, a derivative function describes their derivatives with respect to time. internal states.

Symbolically, these functions can be expressed as follows:

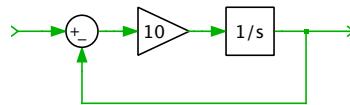
$$\begin{aligned} y &= f_{\text{output}}(t, u, x_c, x_d) \\ x_d^{\text{next}} &= f_{\text{update}}(t, u, x_c, x_d) \\ \dot{x}_c &= f_{\text{derivative}}(t, u, x_c, x_d) \end{aligned}$$

Such a description is most convenient for implementation in a procedural programming language like C.

Block Diagrams

A more graphic modeling method that is commonly used in control engineering is a block diagram such as the one below which shows a low pass filter.

Each of the three blocks is again a dynamic system in itself, that can be described with its own set of system equations. The blocks are interconnected



with directed lines to form a larger system. The direction of the connections determines the order in which the equations of the individual blocks must be evaluated.

Physical Models

Block diagrams are very convenient to model control structures where it is clear what the input and output of a block should be. This distinction is less clear or impossible for physical systems.

For instance, an electrical resistor relates the quantities voltage and current according to Ohm's law. But does it conduct a current because a voltage is applied to it, or does it produce a voltage because a current is flowing through it? Whether the first or the second formulation is more appropriate depends on the context, e.g. whether the resistor is connected in series with an inductor or in parallel with a capacitor. This means that it is not possible to create a single block that represents an electrical resistor.

Therefore, block diagrams with their directed connections are usually not very useful for modeling physical systems. Physical systems are more conveniently modeled using schematics in which the connections between individual components do not imply a computational order.

PLECS currently supports physical models in the electrical domain and the thermal domain (in the form of lumped parameter models).

Simulating Dynamic Systems

A simulation is performed in two phases – initialization and execution – that are described in this section.

Model Initialization

Physical Model Equations

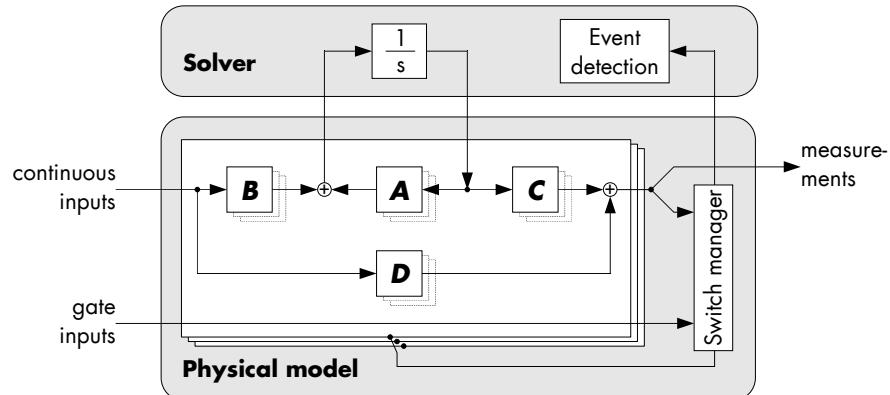
PLECS first sets up the system equations for the physical model according to e.g. Kirchhoff's current and voltage laws. If the physical model contains only ideal linear and/or switching elements it can be described by a set of piecewise linear state-space equations:

$$\dot{x} = A_\sigma x + B_\sigma u$$

$$y = C_\sigma x + D_\sigma u$$

The subscript σ is due to the fact that each state-change of a switching element leads to a new set of state-space matrices.

The complete physical model is thus represented by a single, atomic subsystem. The following figure shows the interaction between the physical subsystem, the surrounding block diagram and the ODE solver.



The physical subsystem accepts external input signals for controllable sources and for switching elements and it provides an output signal containing the values of physical measurements. During the simulation, the derivatives of the physical state variables are calculated and handed over to the solver which in turn calculates the momentary values of these state variables.

The Switch Manager monitors the gate signals and the internal measurements and decides whether a switching action is necessary. If so, it initiates the calculation of a new set of equations. The Switch Manager also provides auxiliary signals – so-called zero-crossing signals – to the solver for proper location of the exact instants when a switching should occur.

State-Space Discretization When used with a fixed-step solver, PLECS transforms the physical model into a discrete state-space model. The continuous state-space equations are discretized using the bilinear transformation (also known as Tustin's method). The integration of the state variables is thus replaced with a simple update rule:

$$\begin{aligned}\mathbf{x}_n &= \mathbf{A}_d \mathbf{x}_{n-1} + \mathbf{B}_d (\mathbf{u}_n + \mathbf{u}_{n-1}) \\ \mathbf{A}_d &= \left(1 - \frac{\Delta t}{2} \mathbf{A}\right)^{-1} \cdot \left(1 + \frac{\Delta t}{2} \mathbf{A}\right) \\ \mathbf{B}_d &= \left(1 - \frac{\Delta t}{2} \mathbf{A}\right)^{-1} \cdot \frac{\Delta t}{2} \mathbf{B}\end{aligned}$$

where Δt is the discretization time step.

With naturally commutated switching devices such as diodes and thyristors, the natural switching instants will generally not coincide with a time step of the discretized circuit model. The Switch Manager detects such non-sampled events and uses an interpolation scheme to ensure that the state variables are always consistent with the switch positions.

Block Sorting

After the setup of the physical model, PLECS determines the execution order of the block diagram. As noted above, the physical model is treated as a single atomic subsystem of the block diagram. The execution order is governed by the following *computational causality*:

If the output function of a block depends on the current value of one or more input signals, the output functions of the blocks that provide these input signals must be evaluated first.

Direct feedthrough The property of an input port whether or not its current signal values are required to compute the output function is called *direct feedthrough*. For example, the output function of a linear gain is

$$y = k \cdot u$$

and so the input signal of the gain has direct feedthrough. In contrast, the output function of an integrator is

$$y = x_c$$

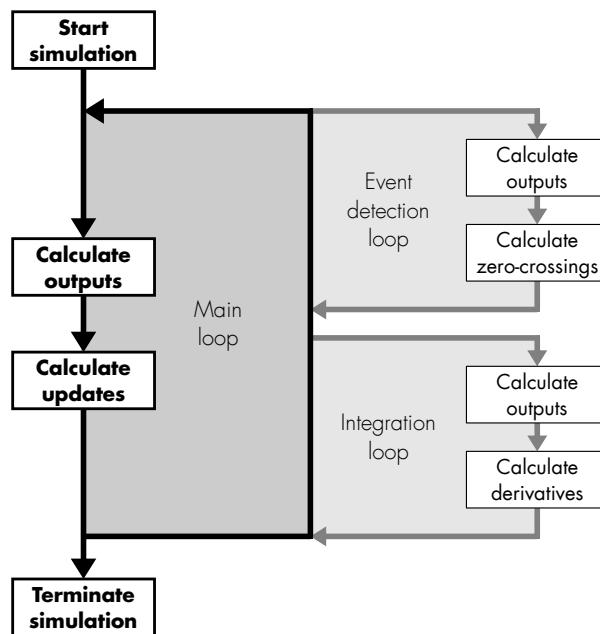
i.e. the integrator just outputs its current state regardless of the current input. The integrator input therefore does *not* have direct feedthrough.

Algebraic loops An *algebraic loop* is a group of one or more blocks that are connected in a circular manner, so that the output of one block is connected to a direct feedthrough input of the next one.

For such a group it is impossible to find a sequence in which to compute their output functions because each computation involves an unknown variable (the output of the previous block). Instead, the output functions of these blocks must be *solved simultaneously*. PLECS currently cannot simulate block diagrams that contain algebraic loops.

Model Execution

The figure below illustrates the workflow of the actual simulation.



Main Loop

The main simulation loop – also called a *major time step* – consists of two actions:

- 1 The output functions of all blocks are evaluated in the execution order that was determined during block sorting. If a model contains scopes, they will be updated at this point.
- 2 The update functions of blocks with discrete state variables are executed to compute the discrete state values for the next simulation step.

Depending on the model and the solver settings, the solver may enter one or both of the following minor loops.

Integration Loop

If a model has continuous state variables, it is the task of the solver to numerically integrate the time derivatives of the state variables (provided by the model) in order to calculate the momentary values of the states variables.

Depending on the solver algorithm, an integration step is performed in multiple stages – also called *minor time steps* – in order to increase the accuracy of the numerical integration. In each stage the solver calculates the derivatives at a different intermediate time. Since the derivative function of a block can depend on the block's inputs – i.e. on other blocks' outputs – the solver must first execute all output functions for that particular time.

Having completed an integration step for the current step size, the solver checks whether the local integration error remains within the specified tolerance. If not, the current integration step is discarded and a new integration is initiated with a reduced step size.

Event Detection Loop

If a model contains discontinuities, i.e. instants at which the model behavior changes abruptly, it may register auxiliary event functions to aid the solver in locating these instants. Event functions are block functions and are specified implicitly as *zero-crossing functions* depending on the current time and the block's inputs and internal states.

For instance, if a physical model contains a diode, it will register two event functions, $f_{\text{turn on}} = v_D$ and $f_{\text{turn off}} = i_D$, depending on the diode voltage and current, so that the solver can locate the exact instants at which the diode should turn on and off.

If one or more event functions change sign during the current simulation step, the solver performs a bisection search to locate the time of the first zero-crossing. This search involves the evaluation of the event functions at different intermediate times. Since the event function of a block – like the derivative function – can depend on the block's inputs, the solver must first execute all output functions for a particular time. Also these intermediate time steps are called *minor time steps*.

Having located the first event, this will reduce the current step size so that the next major time step is taken *just after* the event.

Sampled Data Systems

PLECS allows you to model sampled data systems, i.e. discrete systems that change only at distinct times. You can model systems that are sampled periodically or at variable intervals, systems that contain blocks with different sample rates, and systems that mix continuous and discrete blocks.

Sample Times

Sample times are assigned on a per-block basis, and some blocks may have more than one sample time. PLECS distinguishes between the following sample time types:

Continuous A *continuous sample time* is used for blocks that must be updated in every major and minor time step. This includes all blocks that have continuous state variables, such as the Integrator or Transfer Function.

Semi-Continuous A *semi-continuous sample time* is used for blocks that must be updated in every major time step but whose output does not change during minor time steps. This applies for instance to the Memory block, which always outputs the input value of the previous major time step.

Discrete-Periodic A *periodic sample time* is used for blocks that are updated during major time steps at regular intervals.

Discrete-Variable A *variable sample time* is used for blocks that must be updated during major time steps at variable intervals which are specified by the blocks themselves.

For most block types the sample time is automatically assigned. Discrete blocks and the C-Script block (see page 211) have a parameter **Sample Time** allowing you to specify the sample time explicitly. A sample time is specified

as a two-element vector consisting of the sample period and an offset time. The offset time can be omitted if it is zero.

The following table lists the different sample time types and their corresponding parameter values.

Sample Time Parameter Values

Type	Value
Continuous	[0, 0] 0
Semi-Continuous	[0, -1]
Discrete-Periodic	[T_p , T_o] T_p : Sample period, $T_p > 0$ T_o T_o : Sample offset, $0 \leq T_o < T_p$
Discrete-Variable	[-2, 0] -2

Multirate Systems

Systems that contain blocks with multiple different discrete-periodic sample times are called multirate systems. For such systems, PLECS calculates a base sample time as the greatest common divisor of the periods and offsets of the individual sample times. The individual periods and offsets are then expressed as integer multiples of the base sample time.

This is necessary in order to avoid synchronization problems between blocks with different sample times that would occur when the sample hits are calculated using floating-point arithmetic. For instance, in double precision floating-point arithmetic $3*1e-4$ is not equal to $3e-4$ (even though the difference is only about $5.4 * 10^{-20}$).

In order to find the greatest common divisor, PLECS may slightly adjust individual sample periods or offsets within a relative tolerance of approximately $\pm 10^{-8}$. PLECS does not allow the base sample time to become smaller than 10^{-6} times the largest sample period in order to avoid overflows in the integer arithmetic.

Troubleshooting

If PLECS fails to find an appropriate base sample time it will show a corresponding error message. There are three possibilities to resolve the problem:

Adjusting the sample times Adjust the sample times of the individual blocks in the system so that PLECS can find a base sample time within the above constraints. Whenever possible, specify sample times as rational numbers instead of decimal fractions. For instance, for a block that is sampled with a frequency of 30 kHz enter $1/30e3$ instead of $3.3333e-5$.

Allow multiple base sample times You can allow PLECS to use different base sample rates for different groups of block sample times. To do so, uncheck the option **Use single base sample rate** in the simulation parameters dialog. Only block sample times within the same group are then guaranteed to be synchronized with each other.

Disable sample time synchronization You can disable the sample time synchronization altogether by unchecking the option **Synchronize fixed-step sample times** in the simulation parameters dialog. This is generally not recommended.

The last two options are only available when using a continuous state-space model with a variable-step solver.

Using PLECS

The user interface of PLECS very closely resembles that of Simulink. Circuits are built using the same simple click and drag procedures that you use to build a model. This chapter explains those aspects of PLECS that either are unique to PLECS or work differently from Simulink.

Configuring PLECS

The PLECS configuration parameters can be modified per user in the PLECS Preferences dialog. Choose the menu entry **Preferences...** from the **File** menu (**PLECS** menu on OS X) to open it.

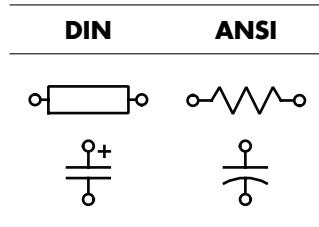
General

The language used by PLECS can be specified in the **Language** field. PLECS uses the language settings of your computer as default setting. Available languages are *English* and *Japanese*. To activate the new language settings PLECS must be restarted.

The setting **Symbol format** controls whether resistors and capacitors are drawn in DIN or ANSI style. The table below shows the different component representation for both settings.

When the **Grid** setting is set to on a grid is displayed in the background of schematic windows for easier placement of components and their connections.

The maximum amount of memory that is used by PLECS during the simulation can be controlled with the setting **Cache size limit**. Once PLECS reaches the memory limit it will discard earlier computation results which may have to be recalculated later during the simulation. On the other hand the value should not be higher than about one third of the physical memory of



the computer where PLECS is running, otherwise the simulation performance may be degraded due to swapping.

In PLECS Standalone the **XML-RPC interface** can be enabled or disabled for external scripting. When enabled, PLECS listens on the specified TCP port for incoming XML-RPC connections. See chapter “XML-RPC Interface in PLECS Standalone” (on page 159) for details on using the XML-RPC interface.

When opening a model, PLECS can reopen all scope windows that were open when the model was saved. The option **Scope windows** enables or disables this behavior.

Libraries

To add custom libraries to the library browser add these libraries in the **User libraries** settings. All custom libraries must be located on the library search path, which is defined differently depending on the PLECS edition:

- For PLECS Standalone the library search path can be changed in the **Search path** settings on the same preferences page.
- For PLECS Blockset the custom libraries must be located on the Matlab search path. The Matlab search path can be set from the Matlab file menu. The **Search path** settings are not available in the PLECS Blockset preferences.

Thermal

The setting **Thermal description search path** contains the root directories of the thermal library. See section “Thermal Library” (on page 88) for more details.

Scope Colors

The **Scope background** setting determines whether the PLECS scopes are drawn with a black or white background.

The **Scope palette** setting determines the appearance of the curves inside the PLECS scopes. To create a new custom palette, select any existing palette and click on **Duplicate**. To remove a palette, click on **Remove**. Note that the default palette is read-only and cannot be removed.

The **Signals** group box lists the base properties used for the curves in a scope plot. You can specify color, line style and line width individually for each curve. If a plot contains more curves than the number of entries in this list, PLECS will restart at the beginning. The default palette specifies six solid, one pixel wide line styles.

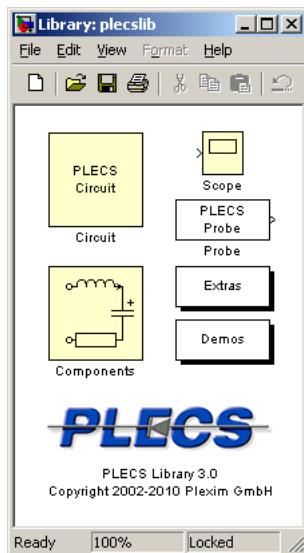
The **Distinguish traces by** setting specifies how different traces for a specific signal are distinguished from each other (see “Adding Traces” on page 67). In the default palette, traces are distinguished by brightness, i.e. by using different shades of the base color. In custom palettes, you can alternatively distinguish traces by varying the color, line style or width. The selected property will then not be available in the signal list. Again, if a plot has more traces than the number of entries in this list, PLECS will restart at the beginning.

Creating a New Circuit with the PLECS Blockset

Open the PLECS library by typing `plecslib` at the MATLAB command prompt. On Windows you can also use the Simulink library browser and click on the entry **PLECS**. Copy the Circuit block from the PLECS library into your Simulink model, then double-click the block to open the schematic editor.

Customizing the Circuit Block

You can customize the mask of the Circuit block to a certain extent, e.g. in order to change the block icon or to define mask parameters. For information on Simulink block masks please refer to the Simulink documentation.



Note You may not change the mask type or remove the callback from the initialization commands. Doing so will break the interface and may lead to loss of data.

If you define mask parameters for the Circuit block, PLECS evaluates component parameters in the mask workspace rather than the MATLAB base workspace. The mask workspace contains both the mask parameters and any additional variables defined by the mask initialization commands. For details on parameter evaluation see “Specifying Component Parameters” (on page 41).

By default, a double-click on the Circuit block opens the schematic editor. This can be changed by editing the **OpenFcn** parameter of the block. To change the behavior so that a double-click opens both the schematic editor and the mask dialog,

- 1 Select the block, then choose **Block Properties** from the **Edit** menu or from the block’s context menu.
- 2 On the **Callbacks** pane of the block properties dialog, select **OpenFcn** from the function list and change the content of the callback function to

```
plecs('sl',202); open_system(gcb,'mask');
```

Alternatively, you can change the behavior so that a double-click opens only the mask dialog. Then, add a checkbox to the dialog that will open the schematic editor when you click on it:

- 1** Select the block, then choose **Block Properties** from the **Edit** menu or from the block's context menu.
- 2** On the **Callbacks** pane of the block properties dialog select **OpenFcn** from the function list and clear the content of the callback function.
- 3** Select the block, then choose **Edit Mask** from the **Edit** menu or from the block's context menu.
- 4** On the **Parameters** pane of the mask editor add a checkbox parameter with the prompt **Open schematic** and the variable name **openschematic**. As a dialog callback for the new parameter enter

```
if (strcmp(get_param(gcb,'openschematic'),'on'))
    set_param(gcb,'openschematic','off');
    plecs('sl',202);
end
```

Using the Library Browser

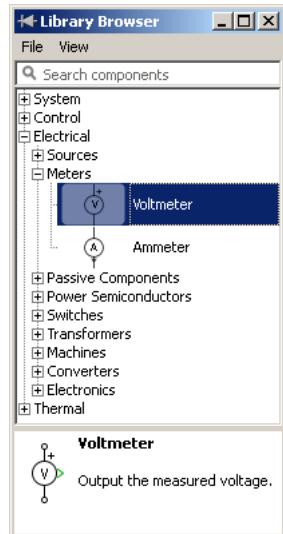
In PLECS Blockset the library browser is opened by a double-click on the Components block in the PLECS library. In PLECS Standalone it is opened automatically when the program is started. It can always be re-opened by selecting **Library Browser** in the **Window** menu.

You can navigate through the component library by clicking on the tree entries. Alternatively, you can search for a specific component by typing part of its name into the search bar.

Drag the components you need from the library browser into the schematic editor.

Note In PLECS Blockset you cannot place Simulink blocks in a PLECS schematic or PLECS components in a Simulink model since both programs do not share the same Graphical User Interface.

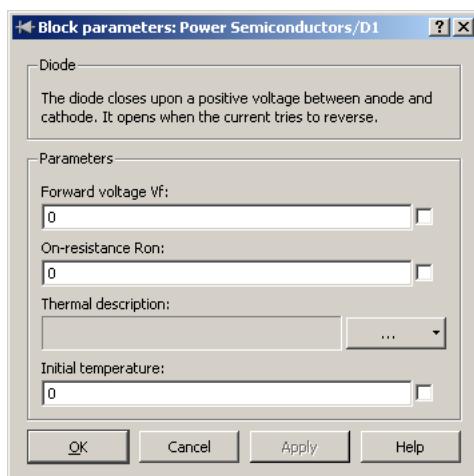
3 Using PLECS



Components

Specifying Component Parameters

Every component has a dialog box to view and modify the component parameters. The dialog box appears when you double-click on the component icon.



Most component parameters accept MATLAB expressions as values, provided that they evaluate to an acceptable result. Parameter expressions are evaluated when you start a simulation or update the Simulink model. In case an error occurs during evaluation of the parameters, an error dialog appears and the corresponding component is highlighted.

An exception to this behavior are parameters that affect the appearance of the component such as the parameter **Number of windings** of the Mutual Inductor (see page 334) or the parameter **Width** of the Wire Multiplexer (see page 478). Such parameters must be literal values and are evaluated immediately.

Using Workspace Variables in Parameter Expressions

Parameter expressions that are not evaluated immediately can include MATLAB variables. Expressions are evaluated as a whole in one workspace. By default, the evaluation workspace is the MATLAB base workspace. However, you can define local mask workspaces for subsystems that will then be used

for the parameter evaluation in the underlying schematics. For information on subsystem mask workspaces see “Mask Parameters” (on page 51).

You can also mask the Circuit block as a whole. This is necessary e.g. if you want parameter expressions to be evaluated in the Simulink model workspace, or when you use the `sim` command from within a MATLAB function and want to access the function workspace. For more information see “Customizing the Circuit Block” (on page 37).

Displaying Parameters in the Schematic

You can cause PLECS to display any component parameter beneath the block icon in the schematic. You specify the parameters to be displayed using the check boxes next to the edit fields in the dialog box. Parameter values can be edited in the schematic directly by double-clicking them.

Changing Component Names

The component name is also entered in the dialog box. All component names in the same schematic must be unique and must contain at least one non-space character. Trailing spaces are removed from the names.

Changing the Orientation of Components

You can change the orientation of a component by choosing one of these commands from the **Format** menu:

- The **Rotate** command rotates a component clockwise 90 degrees (**Ctrl-R**).
- The **Flip left/right** command flips a component horizontally (**Ctrl-F**).
- The **Flip up/down** command flips a component vertically (**Ctrl-I**).

Note Unlike in Simulink, flipping a component is not equivalent to rotating it 180 degrees.

Getting Component Help

Use the **Help** button in the dialog box to get online help about the component.

Libraries

Libraries enable you to ensure that the custom components or masked subsystems used in your circuit are always up-to-date. Or, the other way round, if you are developing your own custom components you can use a library to ensure that changes you make to your component models are automatically propagated to a user's circuit upon loading.

Creating a New Library in PLECS Blockset

To create a new component library, open the PLECS Extras library and copy the PLECS Library block into a Simulink model or library. The Simulink model must be named (i.e. saved) before you can copy components from the component library.

To add the new library to the library browser it has to be added to the list of user libraries in the PLECS Preferences (see chapter "Configuring PLECS" (on page 35) for details).

Creating a New Library in PLECS Standalone

Any model file in PLECS Standalone can be used as a library file. Additionally it is also possible to use PLECS Blockset libraries in PLECS Standalone. To make model file available as a library the file has to be added to the library list in the PLECS preferences (see chapter "Configuring PLECS" (on page 35) for details).

To create a new library file, create a new model file, copy the desired components into it and save it in a directory on the library path. The library path is also set in the PLECS preferences.

Creating a Library Reference

When you copy a library component – either into a circuit schematic or into another or even the same component library – PLECS automatically creates a reference component rather than a full copy. You can modify the parameters of the reference component but you cannot mask it or, if it is already masked, edit the mask. You can recognize a library reference by the string "(link)" displayed next to the mask type in the dialog box or by the string "Link" displayed in the title bar of the underlying schematic windows.

The reference component links to the library component by its full path, i.e. the Simulink path of the PLECS Library block and the path of the component within the component library as they are in effect at the time the copy is made. If PLECS is unable to resolve a library reference it highlights the reference component and issues an error message.

You can fix an unresolved library reference in two ways

- Delete the reference component and make a new copy of the library component.
- In the PLECS Blockset, add the directory that contains the required Simulink model to the MATLAB path and reload the circuit.

Updating a Library Reference

Library references are only resolved upon loading of a circuit. If you make changes to a library component you will need to close and reload all circuits that reference this component in order to propagate the changes.

Breaking a Library Reference

You can break the link between a library reference and the library component. The reference then becomes a simple copy of the library component; changes to the library component no longer affect the copy.

In order to break the link between a reference and its library component, select the reference component, then choose **Break library link** from the **Edit** menu or from the component's context menu.

Connections

Connections define the relationship and interaction between components. PLECS knows different connection types that are explained in this section.

Wires

Wires are ideal electrical connections between two points. They are drawn in black color. A wire can connect one electrical port with another. Several electrical ports can be connected using wire branches.

All points connected by a wire or wire branches have the same electrical potential. The schematic editor does not allow to create wire loops, i.e. connect two points that already have the same potential.

Signals

Signals represent a directed flow of values from the output of one component to the input of one or several other components. Values can be either scalars or vectors. The width of a signal is determined when the simulation is started.

Creating Branches

For drawing a branch connection place the pointer on an existing connection or node where you want the branch to start. With the right mouse button or with the left mouse button while holding down the **Ctrl** key you can create a connection from there to the desired destination.

Annotations

You can annotate circuits with text labels. Create an annotation by double-clicking in an unoccupied area of your PLECS circuit and start typing. You can move an annotation by selecting and dragging it with the mouse. Choose **Text alignment** from the **Format** menu to change the text alignment of the annotation.

Subsystems

Subsystems allow you to simplify a schematic by establishing a hierarchy, where a Subsystem block is on one layer and the elements that make up the subsystem are on another. Subsystems also enable you to create your own reusable components. For more information see “Masking Subsystems” (on page 49).

You can create a subsystem in two ways:

- Add a Subsystem block to your schematic, then open that block and add the blocks it contains to the subsystem.
- Select a number of blocks, then group those blocks into a subsystem.

Creating a Subsystem by Adding the Subsystem Block

To create a new subsystem, first add a Subsystem block to the schematic, then add the elements that make up the subsystem:

- 1 Copy the Subsystem block from the System library into your schematic.
- 2 Double-click on the Subsystem block in order to open it.
- 3 In the empty Subsystem window, build the subsystem. Use the different terminal blocks (e.g. Inport, Outport and the Electrical Port) to configure the interface of the subsystem.

Creating a Subsystem by Grouping Existing Blocks

If a schematic already contains the blocks you want to convert to a subsystem, you can create the subsystem by grouping those blocks:

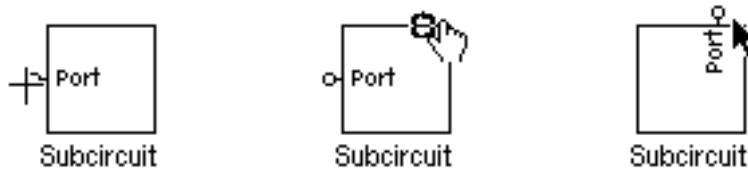
- 1 Select the blocks and connections that you want to include in the subsystem within a bounding box.
- 2 Choose **Create subsystem** from the **Edit** menu. PLECS replaces the selected blocks with a Subsystem block.

Arranging Subsystem Terminals

When you add a port to a subsystem schematic, a corresponding terminal appears at a free slot on the border of the Subsystem block. If necessary, the Subsystem block is resized automatically in order to accommodate the new terminal.

You can move a terminal to another free slot on the border by dragging it with the middle mouse button. While you hold down the mouse button, a circle shows the free slot nearest to the mouse pointer. As an alternative you can press the left mouse button while holding down the **Shift** key. When you release the mouse button, the terminal is moved.

The figures below show a Subsystem block before, during and after moving a terminal.

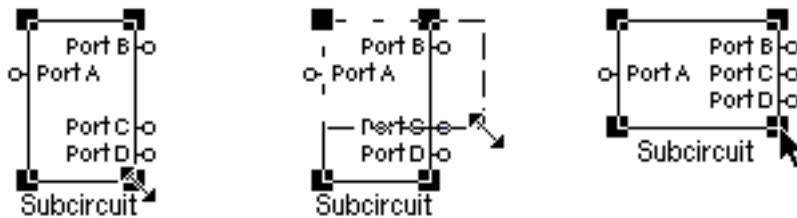


Notice how the shape of the cursor changes to crosshairs as you move it into the capture radius of the terminal. When you press and hold down the center mouse button, the cursor shape changes to a pointing hand.

Resizing a Subsystem Block

To change the size of a Subsystem block, select it, then drag one of its selection handles. While you hold down the mouse button, a dashed rectangle shows the new size. When you release the mouse button, the block is resized. The minimum size of a Subsystem block is limited by the number of terminals on each side.

The figures below show a Subsystem block before, during and after resizing.



Notice how the terminals on the right edge of the Subsystem block are shifted after you release the mouse button in order to fit into the new frame. The block height cannot be reduced further because the terminals cannot be shifted any closer.

Placing the Subsystem Label

The label of a Subsystem block can be placed at any of the following nine positions: at the middle of the four edges, at the four corners, or in the center of the block. To change the placement of the label, drag it to a new location. While you hold down the mouse button, a dashed rectangle shows the new position. When you release the mouse button, the label is moved.

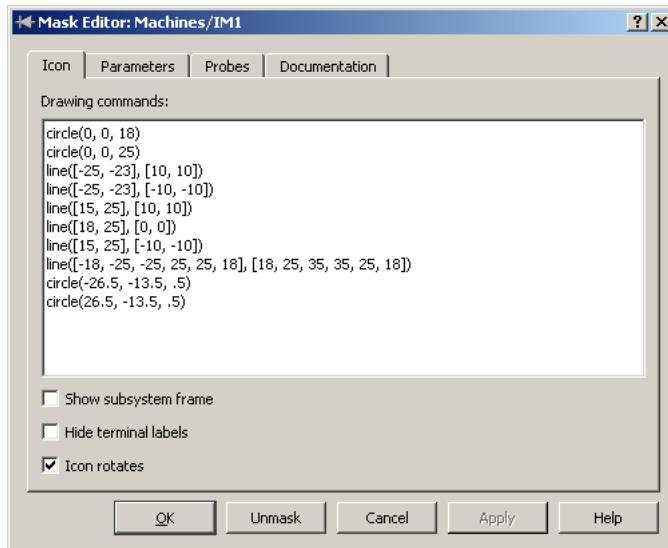
Masking Subsystems

Masking a subsystem allows you to create a custom user interface for a Subsystem block that hides the underlying schematic, making it appear as an atomic component with its own icon and dialog box. Many of the components in the PLECS component library are in fact masked subsystems.

To mask a subsystem, select the Subsystem block, then choose **Mask subsystem** from the **Edit** menu or from the block's context menu. The mask editor appears. The mask editor consists of four tabbed panes that are described in detail below.

Mask Icon

The **Icon** pane enables you to create icons that show descriptive text or labels, graphics and images.



Mask Icon Drawing Commands

The available drawing commands are described below. If you enter more than one command, the graphic objects are drawn in the order in which the com-

mands appear. In case an error occurs during evaluation of the commands PLECS displays three question marks (?) in the mask icon.

Note Unlike with Simulink masks, the PLECS drawing commands do not have access to variables defined in the mask or base workspace.

Text

`text('text')` displays a text in the center of the icon.

`text(x, y, 'text' [, fontsize])` places the text at the coordinates *x* and *y*. The optional argument *fontsize* allows you to specify the font size.

The displayed text does not rotate or flip together with the icon. It is always displayed from left to right and it is centered both horizontally and vertically at its position.

Line

`line(xvec, yvec)` plots the vector *yvec* against the vector *xvec*. Both vectors must have the same length. The vectors may contain NaN and inf values. When NaNs or infs are encountered, the line is interrupted and continued at the next point that is not NaN or inf.

Patch

`patch(xvec, yvec)` draws a solid polygon whose vertices are specified by the vectors *xvec* and *yvec*. Both vectors must have the same length.

Circle

`circle(x, y, r)` draws a circle at the coordinates *x* and *y* with the radius *r*.

Image

`image(xvec, yvec, imread('filename') [, 'on'])` reads an image from the file *filename* and displays it on the mask icon. The parameter *filename* must either be an absolute filename (e.g. C:\images\myimage.png) or a relative filename that is appended to the model's directory (e.g. images\myimage.png). The two-element vectors *xvec* and *yvec* specify the minimum and maximum coordinates of the image's extent.

Use the optional flag 'on' to indicate that the image data should rotate or flip together with the mask icon. By default, this is set to 'off', and the image data remains stationary.

Color

`color(r, g, b)` changes the current drawing color. The new color is given by *r*, *g* and *b* which specify the red, green and blue components. Each value is given as an integer in the range from 0 to 255.

Examples:

`color(0, 0, 0)` changes the color to black.
`color(255, 0, 0)` changes the color to red.
`color(255, 255, 255)` changes the color to white.

Mask Icon Coordinates

All coordinates used by the mask drawing commands are expressed in pixels. The origin of the coordinate system is always the center of the block icon; it is moved when the block is resized.

Use the icon frame and/or the terminal locations as reference points in order to position graphic elements. Both the frame and the terminals snap to a grid of 10 by 10 pixels.

Mask Icon Properties

Show subsystem frame

The subsystem frame is the rectangle that encloses the block. It is drawn if this property is set, otherwise it is hidden.

Hide terminal labels

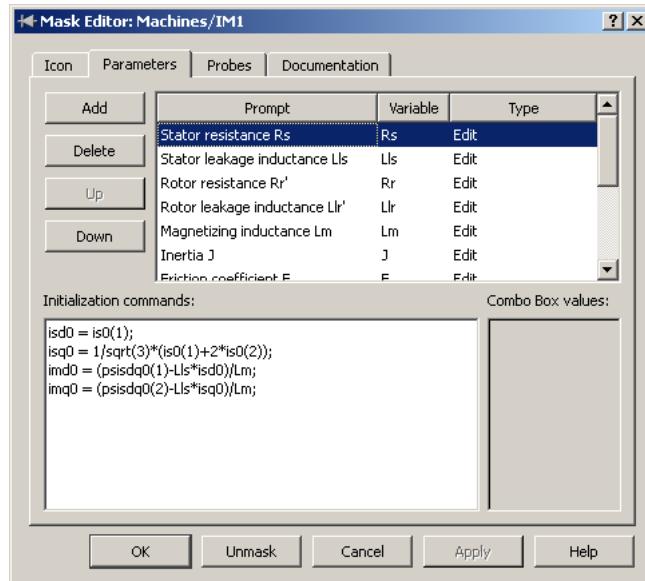
This property controls whether the terminal labels underneath the icon are shown or hidden. A terminal label is only shown if this property is unset and the name of the corresponding port block is visible.

Icon rotates

If drawing commands are given this property determines whether the drawn icon rotates if the component is rotated. The drawn icon remains stationary if this property is unchecked.

Mask Parameters

The **Parameters** pane enables you to define the parameters that will appear in the dialog box of the masked subsystem.



Prompts and Associated Variables

Mask parameters are defined by a *prompt*, a *variable name* and a *type*. The prompt provides information that helps the user identify the purpose of a parameter. The variable name specifies the variable that is to store the parameter value.

Mask parameters appear on the dialog box in the order they appear in the prompt list. Parameters of type **Edit** are shown as a text edit field. Parameters of type **Combo Box** offer a choice of predefined values. The possible values are defined in the **Combo Box values** field with each line representing one value. Parameters of type **Check Box** can be set to **false** or **true**. Parameters of type **Thermal** allow to specify a thermal description. See section “Thermal Description Parameter” (on page 85) for more details.

You can add or remove parameters or change their order by using the four buttons to the left of the prompt list.

Variable Scope

PLECS associates a local variable workspace with each masked subsystem that has one or more mask parameters defined. Components in the underlying

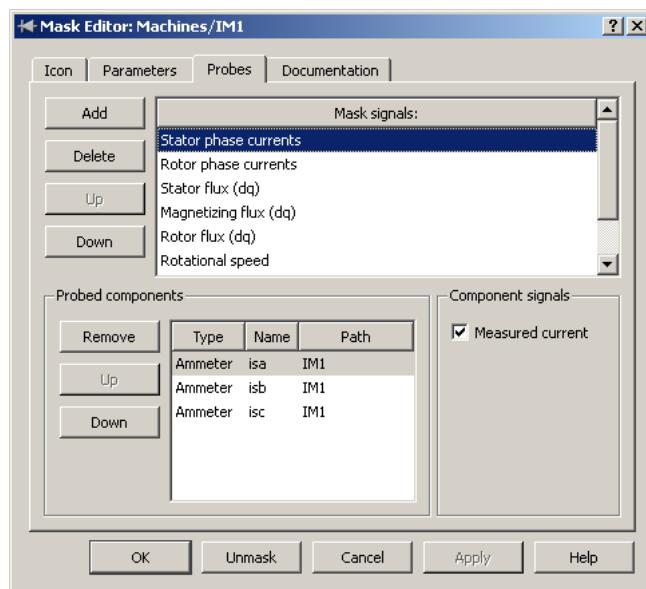
schematics can only access variables that are defined in this mask workspace.

Initialization Commands

The mask initialization commands are evaluated in the mask workspace when a simulation is started. You can enter any valid MATLAB expression, consisting of MATLAB functions, operators, and variables defined in the mask workspace. Variables defined in the base workspace cannot be accessed.

Mask Probe Signals

The **Probes** pane enables you to define the probe signals that the masked subsystem will provide to the PLECS Probe. Mask probe signals appear in the probe editor in the order they appear in the mask signal list. You can add or remove signals or change their order by using the four buttons to the left of the signal list.

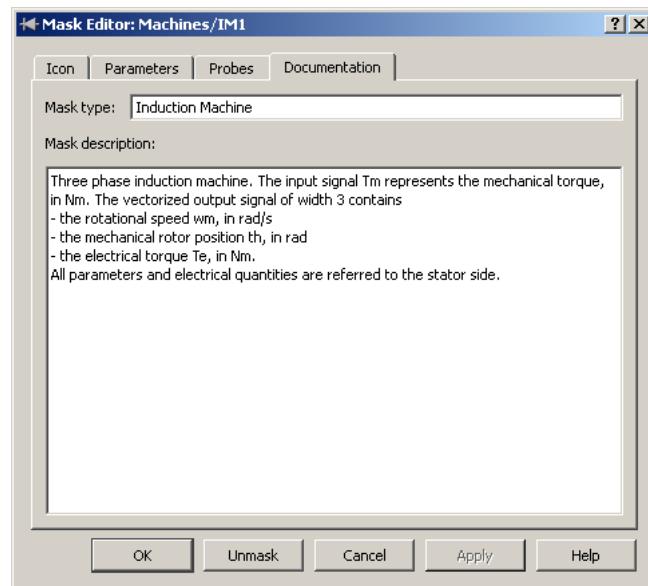


Mask probe signals are defined as vectors of probe signals from components below the subsystem mask. For this reason the controls in the lower half of the dialog are identical to those of the probe editor. In order to define a mask

signal, select the signal in the list and then drag the desired components into the dialog window. The new components are added to the bottom of the list of probed components. Next, select the components one by one and enable the desired component signals in the list on the right side by using the check boxes.

Mask Documentation

The **Documentation** pane enables you to define the descriptive text that is displayed in the dialog box of the masked subsystem.



Mask Type

The mask type is a string used only for purposes of documentation. PLECS displays this string in the dialog box and appends "(mask)" in order to differentiate masked subsystems from built-in components.

Mask Description

The mask description is informative text that is displayed in the dialog box in the frame under the mask type. Long lines of text are automatically wrapped to fit into the dialog box. You can force line breaks by using the **Enter** or **Return** key.

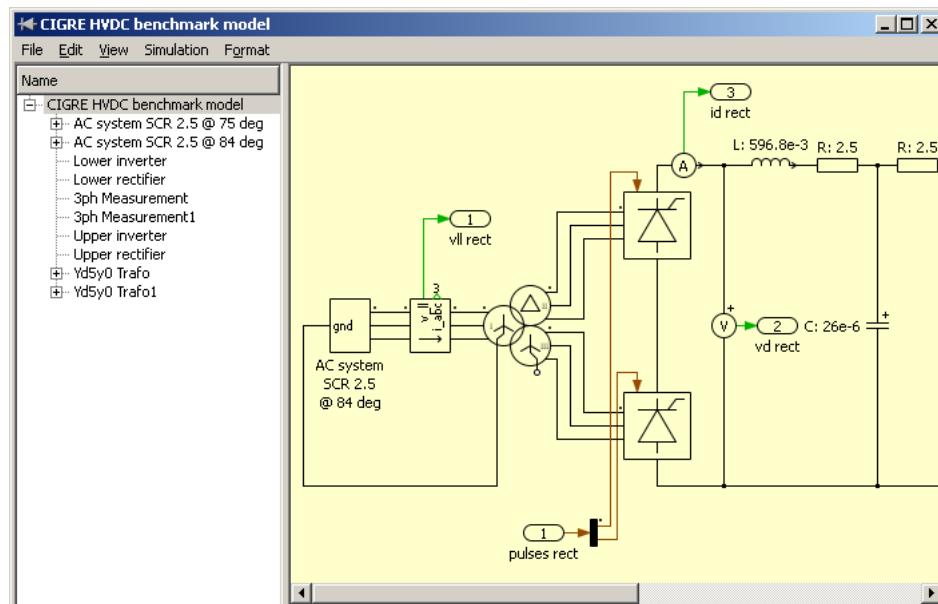
Unprotecting Masked Subsystems

If you define a mask icon for a Subsystem block, PLECS automatically protects the block and the underlying schematic. You can no longer resize the Subsystem block or modify the sub-schematic. The purpose of this protection is to prevent the user from making unintentional changes that might render the icon useless.

If you want to change a masked Subsystem block, you can unprotect it by choosing **Unprotect** from the **Edit** menu or from the block's context menu. You can later protect it again by choosing **Protect** from the same menus.

Circuit Browser

The Circuit Browser enables you to navigate a circuit diagram hierarchically. To display the Circuit Browser, select **Show circuit browser** from the **Circuit browser options** submenu of the **View** menu of the schematic editor.



The editor window splits into two panes. The left pane shows a tree-structured view of the circuit hierarchy. The right pane displays the schematic of the selected (sub-)circuit.

The first entry in the tree view corresponds to the top-level schematic of your circuit. A “+” or “-” sign next to a name indicates that the corresponding schematic contains one or more subcircuits. By double-clicking on the entry you can expand or collapse the list of these subcircuits. To view the schematic of any (sub-)circuit listed in the tree view, select the entry by clicking on it.

Showing Masked Subsystems

By default the Circuit Browser does not list masked subsystems. You can change this behavior by selecting **Show masked subsystems** from the **Circuit browser options** submenu of the **View** menu of the schematic editor.

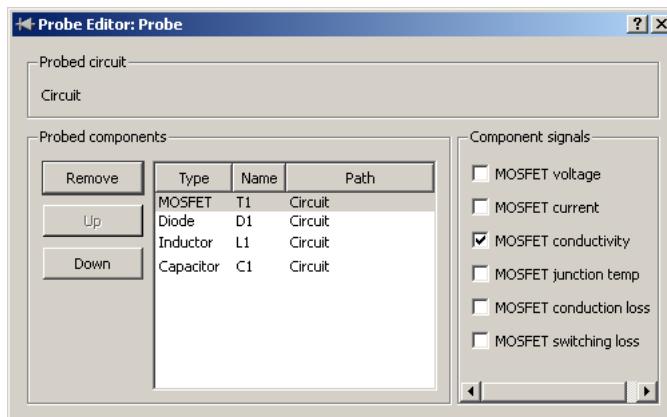
PLECS Probe

PLECS
Probe

The PLECS Probe enables you to monitor various quantities in a circuit. Most intrinsic components provide one or more *probe signals* that describe their current state, input, or output signals. For instance, an inductor provides a probe signal that monitors the inductor current; the probe signals of a diode are the diode voltage, current and conduction state.

The PLECS Probe can either be used in a PLECS schematic or – for the PLECS Blockset – in a Simulink model. To use the PLECS Probe in a schematic use the Probe block from the “System” Library.

In order to use the PLECS Probe in Simulink, drag the Probe block from the PLECS library into the Simulink model that contains the circuit which you want to probe. Double-click the icon to open the probe editor window.



This window contains the following information.

Probed circuit For the Simulink Probe the text box across the top shows the name of the circuit that you are probing and its path, i.e. the Simulink system containing the Circuit block.

Note A Simulink Probe must be in the same Simulink model as the Circuit block whose components you want to monitor. In addition, a Simulink Probe block only accepts components from one single Circuit block at a time.

Probed components The list box on the left side shows the components that you have selected for probing. The components are identified by their type, name and path within the circuit. For adding components to this list, simply select them in the schematic editor and drag them into the probe editor. The new components are appended at the bottom of the list. You can reorder the components by using the **Up**, **Down** and **Remove** buttons.

Component signals The list box on the right side shows the available probe signals for the selected component. Use the check boxes next to the signal names in order to enable or disable individual signals. You can simultaneously edit the signal states of several components provided that the components have the same type. In order to select multiple components, hold the **Shift** or **Ctrl** key while clicking on a list entry.

For PLECS Probes that are used in a PLECS schematic there are two ways to add components to the probe: Either drag them into the **Probed components** area in the probe dialog (see above) or drop them onto the Probe block directly.

The output of the Probe block is a vector signal consisting of all enabled probe signals. If no probe signal is enabled a warning message will be printed to the command window and the block will output a scalar zero.

Copying a Probe

When you copy a PLECS Probe in a PLECS schematic, one of the following three cases can apply:

- 1 If a probed component is copied *simultaneously* with a Probe block referring to it, the copied Probe block will refer to the *copy* of the component.
- 2 Else, if the Probe block is copied within *the same* circuit, the copied Probe block will refer to the *original* component.
- 3 Else (i.e. if the Probe block is copied into *a different* circuit), the probe reference will be removed.

For technical reasons it is not possible to determine whether a PLECS Probe for Simulink is copied simultaneously with a Circuit block. Therefore, PLECS only distinguishes between the following two cases:

- 1 If you copy a Simulink Probe block within *the same* model, the copied Probe block will always refer to the original components.
- 2 If you copy a Probe block into *a different* model, all data is cleared from the copied block.

Controlling Access to Circuits and Subcircuits

PLECS allows you to control user access to individual subcircuits or to complete circuits. In particular, you can prevent a user from viewing or modifying a schematic while still allowing the user to simulate a circuit.

To change the access settings of a circuit, open the permissions dialog box by choosing **Circuit permissions** from the **File** menu. To change the settings of a subcircuit, choose **Subcircuit permissions** from the **Edit** menu or from the block's context menu.

You can grant or deny the following privileges:

- The **View** privilege controls whether a user can view the schematic of a circuit or subcircuit.
- The **Modify** privilege controls whether a user can modify the schematic of a circuit or subcircuit. For a subcircuit it also controls whether the mask definition may be modified.

If you apply access restrictions you will be asked for a password to prevent an unauthorized person from lifting these restrictions. The access settings can only be changed again if the correct password is provided.

Encrypting Circuits and Subcircuits

When PLECS saves a circuit with access restrictions to the Simulink model file, it encrypts the respective sections to protect the circuit description from unauthorized access.

Exporting Circuits for the PLECS Viewer

This section applies only to the PLECS Blockset.

The PLECS Viewer enables you to share your circuit models with users that do not have a license for PLECS. The PLECS Viewer is available for free and allows a user to simulate and optionally view – but not modify – a circuit model, provided that it bears a special signature. In particular, the PLECS Viewer does not permit changing a component parameter, nor is it possible to specify parameters as variables from the MATLAB workspace.

In order to export a circuit for use with the PLECS Viewer, choose **Export for PLECS Viewer** from the **File** menu. If the Simulink model has unsaved changes you will be asked to save them before you can proceed. Afterwards a dialog allows you to specify a filename for the Viewer version of the model. PLECS will then automatically copy the current model to the specified export file, replace component parameters that access the MATLAB workspace with their actual values, break any links to component libraries, and sign it for use with the Viewer. The original model itself remains unchanged.

You can also export a model using the MATLAB command line interface. The command line interface also allows you to protect any PLECS circuit against opening in the PLECS Viewer. See section “Export for PLECS Viewer” (on page 153) for more details.

Note An exported circuit can not be changed by anyone – not even by its creator. It is therefore advisable that you keep the original model for later use and that you choose export filenames that are easily distinguished from the original.

Exporting Schematics

PLECS allows you to export the schematic to a bitmap or PDF file for documentation. The supported image formats are:

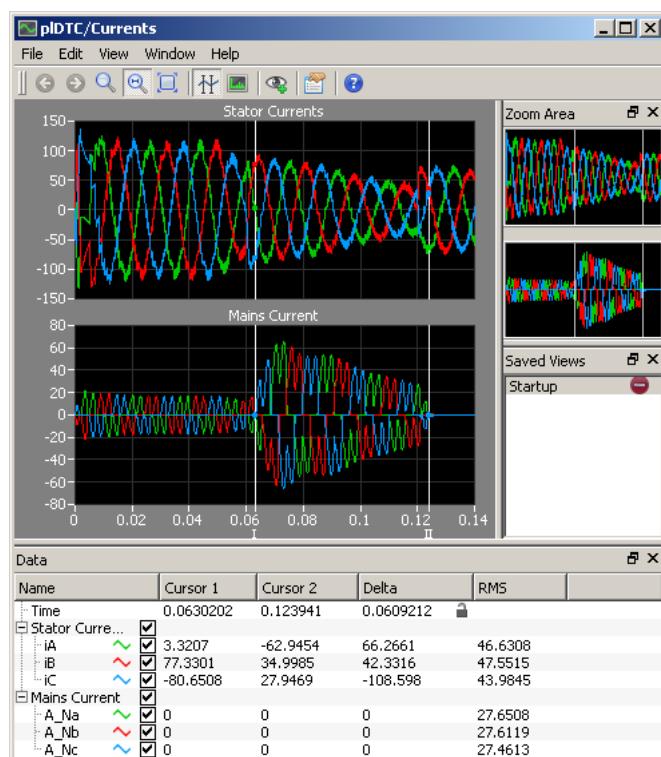
- JPEG (Bitmap)
- TIFF (Bitmap)
- PNG (Bitmap)
- SVG (Scalable Vector Graphics)
- PDF (Portable Document Format)
- PS (PostScript)

To export a schematic choose **Export...** from the **File** menu and select your desired output format. A second dialog lets you specify the export options for the specific format, e.g. the bitmap resolution.

It is also possible to copy schematics to other applications directly via the clipboard. To copy an image of the current schematic to the clipboard choose **Copy as image** from the **Edit** menu, then select **Paste** from the **Edit** menu in your target application.

Using the PLECS Scope

The PLECS scope is used to display simulation results and offers powerful zooming and analysis tools to simplify viewing and processing results. The PLECS scope can be placed on the Simulink worksheet or in the PLECS circuit. The appearance of the PLECS scope is depicted below. The scope contains a plot area and optional Zoom view, Saved view and Data view windows.



Getting Started

To use the scope, drag the scope block from the PLECS library onto your worksheet or schematic diagram. The scope block for Simulink can be found in the top level of the PLECS library. The scope block for a PLECS circuit is located in the PLECS **Sources & Meters** library.

Double clicking on the Scope block opens the Scope window. The main window of the scope can contain multiple plots. Plots can be quickly added or removed by right clicking the plot area and selecting **Insert plot above**, **Insert plot below** or **Remove plot** from the context menu.

The optional Zoom view, Saved view and Data view windows can be opened by right-clicking on the toolbar area. They can also be opened from the **View** menu. These optional windows can be docked and undocked from the main window. To dock them in main window, simply drag them to the desired location inside the main window.

Zoom Operations

Zooming is performed by clicking on the plot area and dragging the mouse until the desired area is selected. Two zoom modes exist: Constrained Zoom and Free Zoom. The zoom mode is selected using the toolbar button. To temporarily switch zoom modes, the **Ctrl** key (**cmd** key with Mac OS) can be pressed.

Constrained Zoom

With Constrained Zoom, zooming is only performed in the x or y direction. The zoom direction is selected by moving the mouse horizontally or vertically.

Free Zoom

With Free Zoom mode activated, the zoom area is defined by dragging the zoom cursor over a certain portion of the plot.

Zoom to Fit

Zoom to fit will fit the entire waveform into the plot window.

Zoom to Specification

A zoom range can also be manually specified. Double-clicking on the x or y axis opens a window in which the x or y range of the zoom area can be entered.

Previous View, Next View

Every time a zoom action is performed, the view is stored in the view history. The previous and next view buttons allow you to navigate backwards and forwards through the view history.

Panning

A zoom area can be panned by dragging the x or y axes of the plot with the hand symbol that appears.

Zoom Area Window

The zoom area window displays the entire waveform and highlights the zoom view that is displayed in the plot window. Constraint Zoom and Free Zoom can also be performed in the zoom area window. The zoom area window is activated by right clicking on the toolbar.

Changing Curve Properties

By default, the curves for the different signals and/or traces in a plot are drawn with a pen that is defined by the palette selected in the PLECS preferences (see “Scope Colors” on page 37).

To change individual curve properties (color, line style and width), right-click on a plot and select **Edit curve properties** from the context menu. This will open a table listing the properties of all visible curves. To change a particular property, double-click on the corresponding table cell.

Locally changed properties are highlighted with a white background and are stored persistently in the model file. In contrast, properties that are defined by the global scope palette have a grey background. To remove all local changes click on **Restore Defaults**.

Spreading Signals

When using a single plot to display multiple signals that assume only a small number of discrete values (such as gate signals), it can be difficult to properly see the value that a particular signal has. You can have the scope automatically separate the signals in a plot by offsetting and scaling them ap-

propriately. All signals are scaled by the same factor and the offsets are distributed evenly in order to maintain the proportions between the signal. Vertical scrolling and zooming is disabled in this mode.

To enable signal spreading, right-click on a plot and select **Spread signals** from the context menu. While spreading is enabled, the y-axis will only display the zero-lines for the individual signals, and zooming in the y-direction is disabled.

Cursors

The cursors are used for measuring waveform values and analyzing the simulation results. Cursors can be positioned by dragging them to a specific time location, or by manually entering a value in the **Time** row in the Data Window.

When the cursors are moved, they will snap to the nearest simulation time step. To place the cursors arbitrarily, hold down the **Shift** key while moving the cursor. The values in the data window will be displayed in *italics* to indicate they are interpolated from the two nearest time steps.

Data Window

When the cursors are activated, the data window appears if it was not already open. By default, the data window displays two columns in which the time and data value of each signal at the position of each cursor are given. The signal names are also displayed and can be modified by double-clicking on the name.

A right-click into the Data Window shows a context menu. Selecting “Copy to Clipboard” copies the current contents of the table to the system clipboard. Afterwards the data can be pasted into other applications, e.g. a spreadsheet tool or word processor.

Signal Type

A small icon that represents the signal type is shown next to the signal name in the data view window. Signals can be of the continuous, discrete or impulse type. The scope automatically determines the signal type from the port settings of the connected signal to ensure the signal is displayed correctly. The signal type can be overridden if necessary by clicking on the signal type icon.

Analyzing Data

Right-clicking on the data view header line in the data view window allows for additional data analysis columns to be displayed. For example, difference, RMS, min, max, and total harmonic distortion (THD) analysis can be performed. The analysis is performed on the data between the two cursors. For meaningful RMS and THD values the cursor range must be equal to the period of the fundamental frequency.

Locking the Cursors

Locking the cursors can be useful for performing measurements over a fixed time period, such as the time period of an ac voltage. When dragging one of the locked cursors, the other cursor will be moved in parallel at a specified time difference. To lock the cursors, the **Delta** column in the Data Window must be made visible by right-clicking on the table header. The desired cursor distance can be entered in the **Time** row of the **Delta** column. The cursors can be unlocked by double-clicking on the lock icon in the **Delta** column.

Fourier Analysis

A Fourier analysis of the data in the current cursor range is accessible from the View menu. The use of the Fourier analysis is detailed in section “Using the Fourier Analysis” (on page 68).

Saving a View

A particular zoom view can be saved by pressing the eye button. The saved views window will appear if it was not already displayed and the new view will be added to the saved views list. To access a particular saved view, click on the view name in the saved views window. Saved views can be renamed by double clicking the name of the view, and reordered by clicking and dragging an entry up and down in the list. A view can be removed with the red delete button.

Adding Traces

After a simulation has been completed the resulting curves can be saved as a trace. Traces allow to compare the results of different simulation runs.

A new trace is added by either pressing the **Hold current trace** button in the toolbar or by pressing the green plus button next to the **Current Trace** entry in the Traces window. To remove a trace press the red minus button next to the trace in the Traces window. Held traces can be reordered by clicking and dragging an entry up and down in the list.

Traces can also be added and removed by simulation scripts. For details, see section “Holding and Clearing Traces in Scopes” (on page 152).

Saving and Loading Trace Data

Existing traces in a scope can be saved by selecting **Save trace data...** from the File menu. The saved traces can be loaded into a scope for later reference. The scope into which the trace data is loaded must have the same number of plots as the scope from which the data was saved. The number of input signals per plot should also match, otherwise the trace data is lost when a new simulation is started.

Scope Parameters

The scope parameters dialog allows for the appearance of the scope to be changed and automatic or custom zoom settings to be applied to the x and y axes. More information can be found in the Scope Parameters Description (see page 379). The plot background color can be changed in the PLECS preferences (see section “Configuring PLECS” (on page 35)).

Printing and Exporting

A plot can be printed or exported from the File menu. When printing, the appearance of the plot and legend can be changed using the Page Setup option. When exporting, the plot style can also be changed and the output size of the image can be customized.

The data table can be exported to e.g. Microsoft Excel using the clipboard. To copy the data to the clipboard open the context menu by right-clicking and choose "Copy to clipboard".

Using the Fourier Analysis

The Fourier Analysis is available from the **View** menu in the PLECS scope window.

The Fourier analysis window shows the magnitude of the Fourier coefficients for the given number of harmonics. The analysis range for the Fourier analysis is determined by the cursors in the scope window. By default it is assumed that the cursor range covers exactly one period of the base frequency, though this can be changed in the Fourier parameters. Note that aliasing effects will be visible if the cursor time range is not an exact integer multiple of the inverse base frequency.

Calculation Parameters

Base Frequency

The analysis range T is always bound to the cursor range in the PLECS scope. In general it consists of n periods of the base frequency, i.e. $T = \frac{n}{f_0}$.

A click on the frequency input field **f:** in the window title bar opens the Base Frequency dialog. Two modes are available to set the base frequency: by freely positioning the cursors in the PLECS scope or by entering the numerical values directly in the Base Frequency dialog.

The first mode is activated by selecting **Calculate from cursor range** in the Base Frequency dialog. In this mode it is assumed that the cursor range covers a single base period. The two cursors can be positioned independently from each other and should be set as exactly as possible to the start and end of a single base period. The corresponding base frequency is displayed in the window toolbar.

If the base frequency is known beforehand it can be entered directly by choosing **Set base frequency**. In this mode the scope cursors are locked to the number of base periods. Moving the cursors still allows you to select the analysis range without changing the base frequency.

Number of Fourier Coefficients

The number of Fourier Coefficients which are calculated can be changed in the input field **N:** in the window title bar.

Display Parameters

Display frequency axis

The frequency axis is either shown underneath each plot or underneath the last plot only.

Frequency axis label

The text is shown below the frequency axis.

Scaling

The Fourier analysis window offers three options to scale the Fourier coefficients: **Absolute** displays the absolute value of each coefficient. **Relative, linear** scales all coefficients such that the coefficient of the base frequency is 1. When set to **Relative, logarithmic (dB)** the coefficients are displayed on a logarithmic scale in Decibels relative to the coefficient of the base frequency.

Table data

The table below the Fourier plots shows the calculated Fourier coefficients. The values can be displayed without phase (**Magnitude only**), with phase values in radians (**Magnitude, phase (rad)**) or with phase values in degree (**Magnitude, phase (degree)**).

The following items can be set for each plot independently:

Title

The name which is displayed above the plot.

Axis label

The axis label is displayed on the left of the y-axis.

Y-limits

The initial lower and upper bound of the y-axis. If set to **auto**, the y-axis is automatically scaled such that all data is visible.

Signal Type

As in the scope window the signal type in the Fourier analysis window can be changed by clicking the small icon next to the signal name in the data view window. Available types are bars, stems and continuous. By default the signals are displayed as bars. Changing the signal type for one signal will affect all signals in the same plot.

Zoom, Export and Print

The Fourier analysis window offers the same zoom, export and print operations as the PLECS scope. See section “Using the PLECS Scope” (on page 62) for details.

Calculation of the Fourier coefficients

The following approximation is made to calculate the Fourier coefficients of a signal with variable sampling intervals ΔT_m :

$$\mathcal{F}(n) = \frac{2}{T} \int_T f(t) e^{-j\omega_0 n t} dt \approx \frac{2}{T} \sum_m \int_{\Delta T_m} f_m(t) e^{-j\omega_0 n t} dt$$

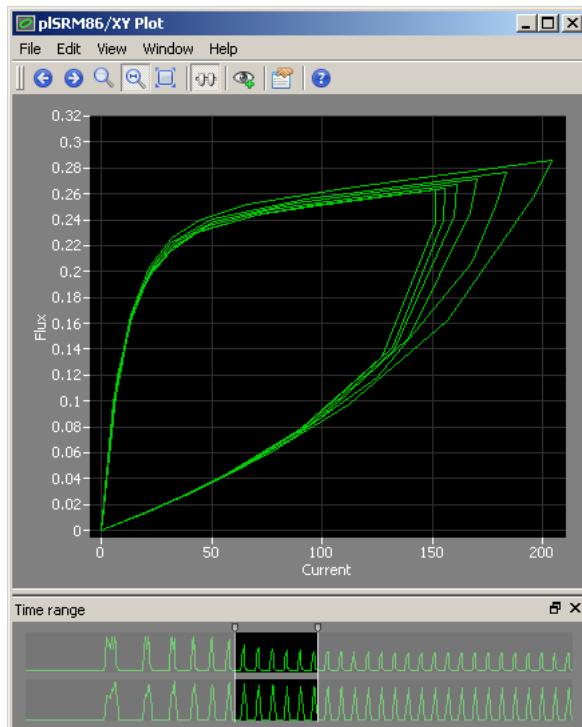
where

$$\begin{aligned} f_m(t) &= a_m t + b_m && \text{for continuous signals} \\ f_m(t) &= b_m && \text{for discrete signals} \end{aligned}$$

A piecewise linear approximation is used for continuous signals. Compared to a fast Fourier transformation (FFT) the above approach also works for signals which are sampled with a variable sample rate. The accuracy of this approximation highly depends on the simulation step size, ΔT_m : A smaller simulation step size yields more accurate results.

Using the XY Plot

The XY plot is used to display the relationship between two signals, x and y. In every simulation step the x and y input signals are taken as coordinates for a new point in the XY plot. Consecutive points are connected by a direct line.



Time Range Window

The time range window allows you to restrict the data that is used for plotting. The window is accessible from the **View** menu.

The time range can be modified by moving its left and right boundary. The inactive time range is grayed out. By clicking into the time range, the active time range can be shifted without changing its length. Any change of the time range is reflected in the XY plot immediately.

If a time range is specified in the XY plot parameters it is used as the default width of the time range in the time range window. A detailed parameter description is available in the XY Plot documentation (see page 480).

Zoom, Save View, Export and Print

The XY plot offers the same zoom, export and print operations as the PLECS scope. See section “Using the PLECS Scope” (on page 62) for details.

Simulation Parameters

PLECS Blockset Parameters

This section describes the simulation parameters available in the PLECS Blockset for Simulink. For the standalone simulation parameters please refer to the next section.

To open the parameter dialog, select **PLECS parameters** from the **Simulation** menu of the schematic editor.

Circuit Model Options

Diode Turn-On Threshold This parameter globally controls the turn-on behavior of line commutated devices such as diodes, thyristors, GTOs and similar semiconductors. A diode starts conducting as soon as the voltage across it becomes larger than the sum of the forward voltage and the threshold voltage. Similar conditions apply to the other line commutated devices. The default value for this parameter is $1e-3$.

For most applications the threshold could also be set to zero. However, in certain cases it is necessary to set this parameter to a small positive value to prevent line commutated devices from bouncing. Bouncing occurs if a switch receives an opening command and a closing command repeatedly in subsequent simulation steps or even within the same simulation step. Such a situation can arise in large, stiff systems that contain many interconnected switches.

Note The Diode Turn-On Threshold is *not* equivalent to the voltage drop across a device when it is conducting. The turn-on threshold only delays the instant when a device turns on. The voltage drop across a device is solely determined by the forward voltage and/or on-resistance specified in the device parameters.

Type This parameter lets you choose between the *continuous* and *discrete* state-space method for setting up the physical model equations. For details please refer to section “Physical Model Equations” (on page 28).

When you choose **Continuous state-space**, PLECS employs the Simulink solver to solve the differential equations and integrate the state variables. The

Switch Manager communicates with the solver in order to ensure that switching occurs at the correct time. This is done with Simulink's zero-crossing detection capability. For this reason the continuous method can only be used with a variable-step solver.

In general, the default solver of Simulink, `ode45`, is recommended. However, your choice of circuit parameters may lead to stiff differential equations, e.g. if you have large resistors connected in series with inductors. In this case you should choose one of Simulink's stiff solvers.

When you choose **Discrete state-space**, PLECS discretizes the linear state-space equations of the physical model as described in section “State-Space Discretization” (on page 29). All other continuous state variables are updated using the Forward Euler method. This method can be used with both *variable-step* and *fixed-step* solvers.

Discrete State-Space Options

Sample time This parameter determines the rate with which Simulink samples the circuit. A setting of `auto` or `-1` means that the sample time is inherited from the Simulink model.

Refine factor This parameter controls the internal step size which PLECS uses to discretize the state-space equations. The discretization time step Δt is thus calculated as the sample time divided by the refine factor. The refine factor must be a positive integer. The default is 1.

Choosing a refine factor larger than 1 allows you to use a sample time that is convenient for your discrete controller while at the same time taking into account the usually faster dynamics of the electrical system.

ZC step size This parameter is used by the Switch Manager when a non-sampled event (usually the zero crossing of a current or voltage) is detected. It controls the relative size of a step taken across the event. The default is `1e-9`.

Tolerances The error tolerances are used to check whether the state variables are consistent after a switching event. The defaults are `1e-3` for the relative tolerance and `1e-6` for the absolute tolerance.

Note The discrete method cannot be used with circuits that contain direct non-linear feedbacks because in conjunction with Tustin's method this would lead to algebraic loops.

This applies for instance to the non-saturable induction machine models. If you must simulate an induction machine with the discrete method, use the Saturable Induction Machine (see page 296) instead. The non-linear feedback paths in this model contain Integrator blocks (see page 304) which prevent the algebraic loops.

PLECS Standalone Parameters

This section describes the simulation parameters available for PLECS Standalone. For the PLECS Blockset simulation parameters please refer to the previous section.

To open the parameter dialog, select **Simulation parameters** from the **Simulation** menu of the schematic editor or press **Ctrl-E**.

Simulation Time

Start Time The start time specifies the initial value of the simulation time variable t at the beginning of a simulation, in seconds. The initial conditions specified in the block parameters must match the specified start time.

Stop Time The simulation ends when the simulation time has advanced to the specified stop time.

Solver

These two parameters let you choose between *variable-step* and *fixed-step* solvers. A fixed-step solver uses the same step size – i.e. the simulation time increment – throughout a simulation. The step size must be chosen by the user so as to achieve a good balance between accuracy and computational effort.

A variable-step solver can adopt the step size during the simulation depending on model dynamics. At times of rapid state changes the step size is reduced

to maintain accuracy; when the model states change only slowly, the step size is increased to save unnecessary computations. The step size can also be adjusted in order to accurately simulate discontinuities. For these reasons, a variable-step solver should generally be preferred.

DOPRI is a variable-step solver using a fifth-order accurate explicit Runge-Kutta formula (the Dormand-Prince pair). This solver is most efficient for *non-stiff systems* and is selected by default. A *stiff system* can be sloppily defined as one having time constants that differ by several orders of magnitudes. Such a system forces a non-stiff solver to choose excessively small time steps. If DOPRI detects stiffness in a system, it will abort the simulation with the recommendation to switch to a stiff solver.

RADAU is a variable-step solver for *stiff systems* using a fifth-order accurate fully-implicit three-stage Runge-Kutta formula (Radau IIa). For non-stiff systems DOPRI is more efficient than RADAU.

The fixed-step solver **Discrete** does not actually solve any differential equations but just advances the simulation time with fixed increments. If this solver is chosen, the linear state-space equations of the physical model are discretized as described in section “State-Space Discretization” (on page 29). All other continuous state variables are updated using the Forward Euler method. Events and discontinuities that occur between simulation steps are accounted for by a linear interpolation method.

Variable-Step Solver Options

Max Step Size The maximum step size specifies the largest time step that the solver can take and should not be chosen unnecessarily small. If you suspect that the solver is missing events, try reducing the maximum step size. However, if you just require more output points for smoother curves, you should increase the *refine factor* (see below).

Initial Step Size This parameter can be used to suggest a step size to be used for the first integration step. The default setting *auto* causes the solver to choose the step size according to the initial state derivatives. You should only change this parameter if you suspect that the solver is missing an event at the beginning of a simulation.

Tolerances The relative and absolute specify the acceptable local integration errors for the individual state variables according to

$$\text{err}_i \leq \text{rtol} \cdot |x_i| + \text{atol}_i$$

If all error estimates are smaller than the limit, the solver will increase the step size for the following step. If any error estimate is larger than the limit, the solver will discard the current step and repeat it with a smaller step size.

The default absolute tolerance setting `auto` causes the solver to update the absolute tolerance for each state variable individually, based on the maximum absolute value encountered so far.

Refine factor The refine factor is an efficient method for generating additional output points in order to achieve smoother results. For each successful integration step, the solver calculates $r - 1$ intermediate steps by interpolating the continuous states based on a higher-order polynomial. This is computationally much cheaper than reducing the maximum step size (see above).

Fixed-Step Solver Options

Fixed step size This parameter specifies the fixed time increments for the solver and also the sample time used for the state-space discretization of the physical model.

Circuit Model Options

The parameters in this group are described in detail in the previous section covering the PLECS Blockset parameters.

System State

This parameter controls how the system state is initialized at the beginning of a simulation. The system state comprises

- the values of all physical storage elements (e.g. inductors, capacitors, thermal capacitances),
- the conduction states of all electrical switching elements (e.g. ideal switches, diodes), and
- the values of all continuous and discrete state variables in the control block diagram (e.g. integrators, transfer functions, delays).

Block parameters When this option is selected, the state variables are initialized with the values specified in the individual block parameters.

Stored system state When this option is selected, the state variables are initialized globally from a previously stored system state; the initial values specified in the individual block parameters are ignored. This option is disabled if no state has been stored.

Store system state... Pressing this button after a transient simulation run or an analysis will store the final system state along with a time stamp and an optional comment. When you save the model, this information will be stored in the model file so that it can be used in future sessions.

Note Adding or removing blocks that have continuous or discrete state variables associated with them will invalidate a stored system state.

Model Initialization Commands

The model initialization commands are executed when a simulation is started in order to populate the base workspace. You can use variables defined in the base workspace when specifying component parameters (see “Specifying Component Parameters” on page 41).

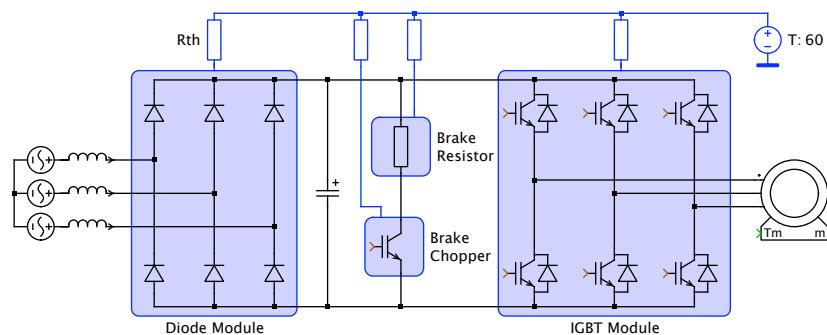
4

Thermal Modeling

Thermal management is an important aspect of power electronic systems and is becoming more critical with increasing demands for compact packaging and higher power density. PLECS enables you to include the thermal design with the electrical design at an early stage in order to provide a cooling solution suitable for each particular application.

Heat Sink Concept

The core component of the thermal library is an idealized heat sink (see page 262) depicted as a semitransparent box in the figure below. A heat sink absorbs the thermal losses dissipated by the components within its boundaries. At the same time, a heat sink defines an isotherm environment and propagates its temperature to the components which it encloses.

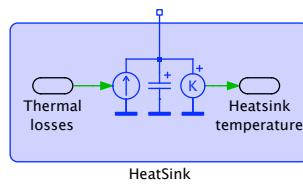


Heat conduction from one heat sink to another or to an ambient temperature is modeled with lumped thermal resistances and capacitances that are con-

nected to the heat sinks. This approach allows you to control the level of detail of the thermal model.

Implementation

Each heat sink has an intrinsic thermal capacitance versus the thermal reference node. All thermal losses absorbed by the heat sink flow into this capacitance and therefore raise the heat sink temperature. Heat exchange with the environment occurs via the external connectors.



You may set the intrinsic capacitance to zero, but then you must connect the heat sink either to an external thermal capacitance or to a fixed temperature, i.e. the Constant Temperature block (see page 221) or the Controlled Temperature block (see page 223).

Thermal Loss Dissipation

There are two classes of intrinsic components that dissipate thermal losses: semiconductor switches and ohmic resistors.

Semiconductor Losses

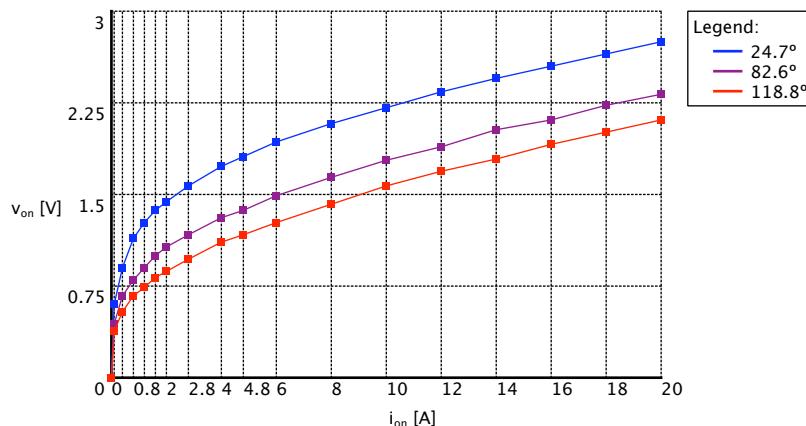
Power semiconductors dissipate losses due to their non-ideal nature. These losses can be classified as conduction losses and switching losses. For completeness the blocking losses due to leakage currents need to be mentioned, but they can usually be neglected.

Semiconductor losses are specified by referencing a thermal data sheet in the component parameter **Thermal description**. See section “Thermal Description Parameter” (on page 85) and “Thermal Library” (on page 88) for more details.

Conduction Losses

The conduction losses can be computed in a straightforward manner as the product of the device current and the device voltage. By default the on-state voltage is calculated from the electrical device parameters as $v = V_f + R_{on} \cdot i$.

However, PLECS also allows you to specify the on-state voltage used for the loss calculation as an arbitrary function of the device current and the device temperature: $v = v_{on}(i, T)$. This function is defined in the **Conduction loss** tab of the thermal description as a 2D look-up table (see “Thermal Editor” (on page 90)).

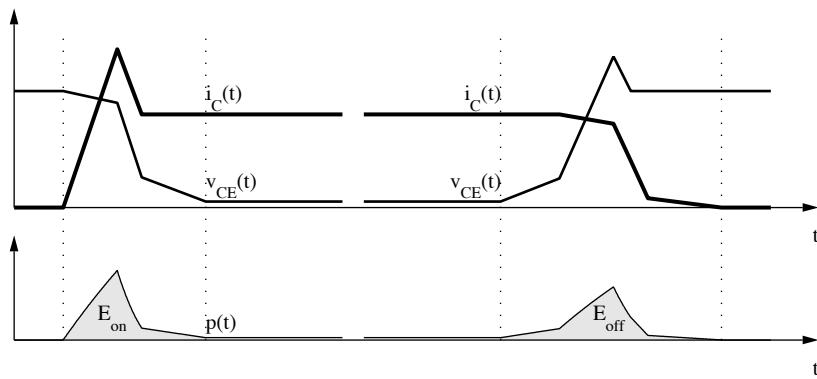


A setting of 0 V for a single temperature and current value means no conduction losses. If you do not specify a thermal description in the device parameters, the default will be used, i.e. the losses are calculated from the electrical device parameters.

Note If you specify the **Thermal description** parameter, the dissipated thermal power does not correspond to the electrical power that is consumed by the device. This must be taken into account when you use the thermal losses for estimating the efficiency of a circuit.

Switching Losses

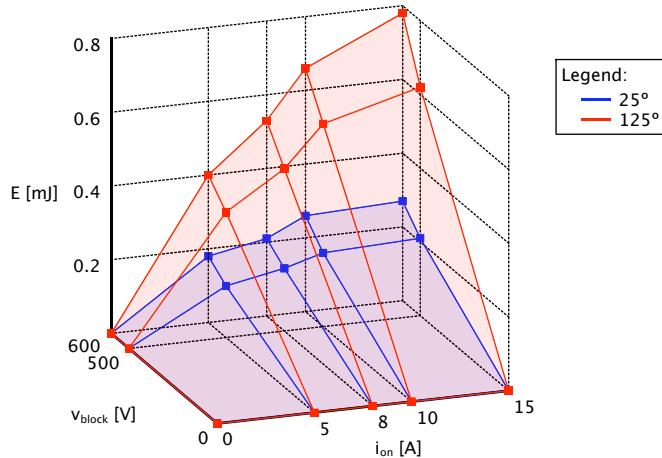
Switching losses occur because the transitions from on-state to off-state and vice versa do not occur instantaneously. During the transition interval both the current through and the voltage across the device are substantially larger than zero which leads to large instantaneous power losses. This is illustrated in the figure below. The curves show the simplified current and voltage waveforms and the dissipated power during one switching cycle of an IGBT in an inverter leg.



In other simulation programs the computation of switching losses is usually challenging because it requires very detailed and accurate semiconductor models. Furthermore, very small simulation time-steps are needed since the duration of an individual switching transition is in the order of a few hundred nanoseconds.

In PLECS this problem is bypassed by using the fact that for a given circuit the current and voltage waveforms during the transition and therefore the total loss energy are principally a function of the pre- and post-switching conditions and the device temperature: $E = E_{\text{on}}(v_{\text{block}}, i_{\text{on}}, T)$, $E = E_{\text{off}}(v_{\text{block}}, i_{\text{on}}, T)$. These functions are defined in the tabs **Turn-on loss** and **Turn-off loss** of the thermal editor as 3D look-up tables (see “Thermal Editor” (on page 90)).

A setting of 0 J for a single voltage, current and temperature value means no switching losses.



Note Due to the instantaneous nature of the switching transitions, the dissipated thermal energy cannot be consumed electrically by the device. This must be taken into account when you use the thermal losses for estimating the efficiency of a circuit.

Semiconductor components that implement this loss model are

- the Diode (see page 232),
- the Thyristor (see page 430),
- the GTO (see page 257),
- the GTO with Diode (see page 259),
- the IGBT (see page 269),
- the IGBT with Diode (see page 275),
- the Reverse Blocking IGCT (see page 281),
- the Reverse Conducting IGCT (see page 283),
- the MOSFET (see page 326),
- the MOSFET with Diode (see page 329) and
- the TRIAC (see page 451).

In addition, the Set/Reset Switch (see page 380) is also included in this group to enable you to build your own semiconductor models.

Ohmic Losses

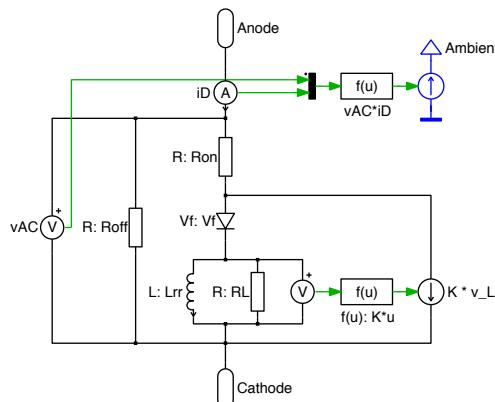
Ohmic losses are calculated as $i^2 \cdot R$ resp. u^2/R . They are dissipated by the following components:

- the Resistor (see page 363),
- the Variable Resistor with Variable Series Inductor (see page 470),
- the Variable Resistor with Constant Series Inductor (see page 467),
- the Variable Resistor with Variable Parallel Capacitor (see page 468) and
- the Variable Resistor with Constant Parallel Capacitor (see page 466).

Heat Sinks and Subsystem

By default, if you place a subsystem on a heat sink, the heat sink temperature is propagated recursively into all subschematics of the subsystem. All thermal losses dissipated in all subschematics flow into the heat sink. In some cases this is not desirable.

The implicit propagation mechanism is disabled if a subschematic contains one or more heat sinks or the Ambient Temperature block (see page 197). This latter block provides a thermal connection to the heat sink enclosing the parent subsystem block.



As an example the figure above shows the subschematic of the Diode with Reverse Recovery (see page 234). By default, this diode model would only dissipate the ohmic losses from the three resistors and the conduction losses of the

internal ideal diode. However, the losses from the reverse recovery current injected by the current source would be neglected because current sources (and also voltage sources) do not dissipate thermal losses.

The Diode with Reverse Recovery therefore uses a Controlled Heat Flow block (see page 222) to inject the electrical power loss into the thermal model via the Ambient Temperature block. The power loss is calculated by multiplying the device voltage and the device current.

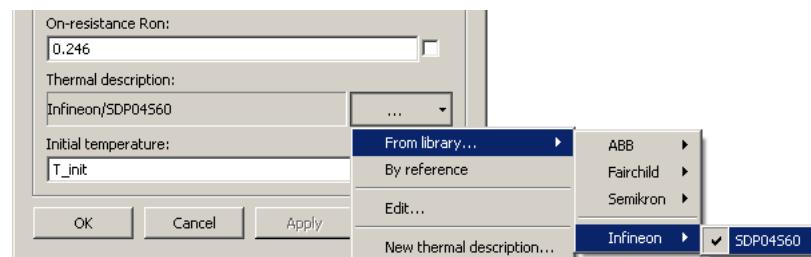
Thermal Description Parameter

Most semiconductor components in PLECS have a parameter **Thermal description**. The parameter can be used in two ways:

- to assign a data sheet from the thermal library to the component or
- to assign a data sheet from a reference variable that is defined either as a thermal mask parameter or in the MATLAB workspace.

Assigning Thermal Data Sheets

Thermal data sheets can be assigned to semiconductors with the menu entry **From library....** PLECS only displays data sheets that match the device type; e.g. in the dialog box of a thyristor only those data sheets appear that have their **Type** field set to Thyristor.



Selecting a data sheet from a thermal library

If no data sheet is available the menu entry is disabled. In thermal parameters of masked subsystem all data sheets are accessible, regardless of their type. See section “Thermal Library” (on page 88) for more information on how to create new data sheets.

Using Reference Variables

To use a reference variable in the **Thermal description** parameter select the menu entry **By reference** from the parameter menu. Afterwards the reference variable can then be entered in the text field.

The reference variable must either be defined in a subsystem mask or in the MATLAB workspace. If a MATLAB workspace variable is used it must specify the name of a thermal description file or a structure that defines the thermal loss data.

Referencing thermal data sheets

If the reference variable refers to a thermal data sheet, it must be specified as a string beginning with `file:` followed by the name of the datasheet. It is possible to use an absolute file path to a thermal description file, for example:

```
thLosses = 'file:C:\Thermal\Vendor\mydiode.xml'
```

Alternatively, the name of a data sheet from the thermal library can be specified. In this case the data sheet must be on the thermal search path. Its name must be provided as a relative path without the `.xml` extension, for example:

```
thLosses = 'file:Vendor/mydiode'
```

Referencing data loss structures

The reference variable can contain a data structure that defines the thermal losses with the fields `Von`, `Eon`, `Eoff` and `CauerChain`. The fields are described as follows:

Von This field is a 2D lookup table for the voltage drop in form of a struct with two index vectors `i`, `T` and an output matrix `v`.

Eon, **Eoff** These fields are 3D lookup tables of the turn-on and turn-off losses in form of structs with three index vectors `v`, `i`, `T` and an output array `E`.

CauerChain This field is a struct of two arrays, R and C which must have the same length. The elements specify the respective values of the resistances and capacitances in the thermal Cauer chain.

Any of the index vectors may be omitted if the lookup value is not dependent on the corresponding variable. The number of dimensions of the output table must correspond to the number of index vectors. If none of the index vectors is specified, the output table must be a scalar. In this case the output can be specified directly as a scalar rather than as a struct with a single scalar field.

An example for constructing a workspace variable containing loss data is given below:

```

von.i = [0 5 15 35 50];
von.T = [25 125];
von.v = [[0.8 1.3 1.7 2.3 2.7]' [0.6 1.1 1.6 2.6 3.2]';
eon.v = [0 200 300];
eon.i = [0 13 23 32 50];
eon.T = [25 125];
eon.E = 1e-3 * ...
    [0.000 0.000 0.000 0.000 0.000
     0.000 0.167 0.333 0.500 1.333
     0.000 0.250 0.500 0.750 1.700];
eon.E(:,:,2) = 1e-3 * ...
    [0.000 0.000 0.000 0.000 0.000
     0.000 0.333 0.667 1.000 2.267
     0.000 0.500 1.000 1.500 3.400];
cc.C = [0.95 2.4];
cc.R = [0.118 0.172];
thLosses = struct('Von', von, 'Eon', eon, 'Eoff', 0, ...
    'CauerChain', cc);

```

In PLECS Blockset, workspace variables can also be constructed from thermal data sheets using the command line interface (see “Converting Thermal Descriptions” (on page 153)).

Thermal Library

PLECS uses a library of thermal data sheets for semiconductors. The data sheets of the thermal library are created and edited with the thermal editor (see “Thermal Editor” (on page 90)). By separating the thermal descriptions of semiconductors from their electrical behavior it is possible to use specific parameters from semiconductor manufactures for thermal simulations in conjunction with the generic electrical switch models from PLECS.

Library Structure

PLECS uses directory names to hierarchically organize the data sheets in the thermal library. The reference to a data sheet consists of its relative path and its filename starting from the directories on the thermal search path.

The search path for thermal libraries is specified in the PLECS preferences (see section “Configuring PLECS” (on page 35)). Each search path entry is the root directory for a library tree. On program startup PLECS searches each root directory in the search path recursively for .xml files and merges the available descriptions into one logical structure. The accessible data sheets can be updated manually by pressing the **Rescan** button in the PLECS preferences window. If a new data sheet is created and saved below a directory which is already on the search path the library is updated automatically.

A common way to organize data sheets within a thermal library is to use the manufacturer name as the first directory level and the part number as the filename of the data sheet.

Global and Local Data Sheets

In addition to the global library search paths specified in the Preferences window PLECS searches a private directory for each model. This allows for sharing models with other users without the need to synchronize the whole thermal library. The private directory is located in the same directory as the model file. Its name is the name of the model file (without the .mdl extension) plus a suffix _plecs, e.g. p1SMPs_CCM_plecs for model p1SMPs_CCM.mdl.

If a library file with the same relative path is found both in the global and the local library the file from the local library is used.

Creating New Data Sheets

New thermal data sheets are created by selecting **New... + Thermal description...** from the **File** menu.

The data sheet should be saved on the thermal search path, otherwise it will not be added to the thermal library and cannot be accessed.

Note It is also possible to import thermal descriptions from PLECS 1.x using the command line interface (see section “Command Line Interface” (on page 153)).

Browsing the Thermal Library

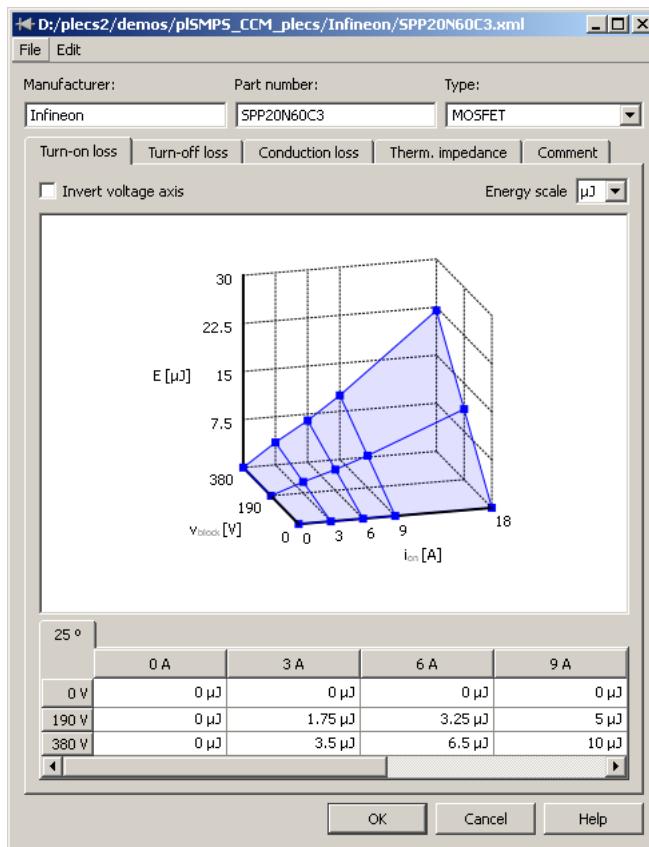
PLECS allows for browsing the thermal library with the **Thermal library browser**. It is invoked from the **View** menu.

The tree view on the left shows all local and global data sheets of the thermal library for the current model.

Thermal Editor

The Thermal Editor is used for creating, viewing and editing thermal data sheets. To open a new editor window select **New... + Thermal description...** from the **File** menu. Existing library data sheets can be edited either in the **Thermal library browser** (accessible from the **View** menu) or by assigning a data sheet to a semiconductor in the **Thermal description** parameter and then selecting the menu entry **Edit....**

The Thermal Editor facilitates editing switching losses, conduction losses and the thermal equivalent circuit of a component.



The text entries **Manufacturer**, **Part number** and **Comment** are for documentation purposes only. The **Type** selector serves as a filter for the **Thermal**

description menu entry. It must be set according to the semiconductor type it is intended to be used with.

In order to access the data sheet in a PLECS model it must be saved in a sub-directory on the thermal search path. See section “Thermal Library” (on page 88) for details of the structure of the thermal library.

Editing Switching Losses

Switching losses are defined as a 3D lookup-table in the **Turn-on loss** and **Turn-off loss** tabs. The energy for each switching event depends on the blocking voltage, the device current and the device temperature. PLECS uses a linear interpolation technique to calculate the actual losses from the given values.

New interpolation points for temperature, voltage and current are added and removed with the **Edit** menu or the context menu in the table. Multiple values can be added separated by semicolons or spaces.

To rotate and tilt the 3D view move the mouse within the view while keeping the left mouse button pressed.

Editing Conduction Losses

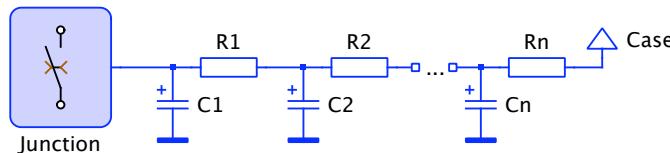
Conduction losses are defined by means of the on-state voltage drop as a 2D lookup-table in the **Conduction loss** tab. The voltage drop depends on the device current and the device temperature. PLECS uses linear interpolation to calculate the actual voltage drop from the given values.

New interpolation points for temperature and current are added and removed with the **Edit** menu or the context menu in the table. Multiple values can be added separated by semicolons or spaces.

Editing the Thermal Equivalent Circuit

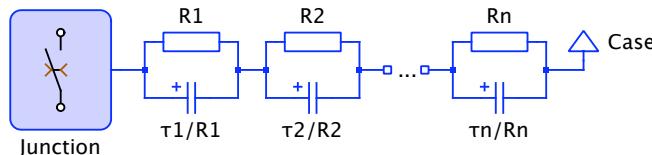
The thermal equivalent circuit of a component describes its physical structure in terms of thermal transitions from the junction to the case. Each transition consists of a thermal resistor and a thermal capacitor. They can be edited in the **Therm. impedance** tab of the thermal editor. The thermal equivalent circuit is specified either in Cauer or Foster form.

The structure of a Cauer network is shown in the figure below. In the thermal editor the number of chain elements n and the values for R_i (in K/W) and C_i (in J/K) for each chain element need to be entered.



Cauer network

The figure below illustrates the structure of a Foster network. In the thermal editor the number of chain elements n and the values for R_i in (K/W) and τ_i (in s) for each chain element need to be entered. Foster networks can be converted to Cauer networks by pressing the button **Convert to Cauer**.

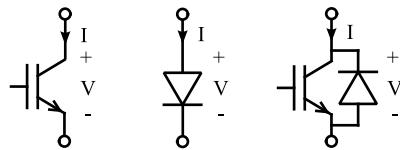


Foster network

Note Internally, PLECS always uses the Cauer network to calculate the thermal transitions. Foster networks are converted to Cauer networks at simulation start. Strictly speaking, this conversion is only accurate if the temperature at the outer end of the network, i.e. the case, is held constant. For practical purposes the conversion should yield accurate results if the external thermal capacitance is much bigger than the capacitances within the network.

Semiconductor Loss Specification

Care must be taken to ensure the polarity of the currents and voltages are correct when specifying conduction and switching loss data for semiconductor switches and diodes. If one or both polarities are in the wrong direction, the losses will be zero or incorrect. The voltage and current polarities of a single semiconductor switch, diode and semiconductor switch with diode are defined in PLECS as shown in the figure below.



Voltage and current polarity of single semiconductor switch, diode and semiconductor switch with diode

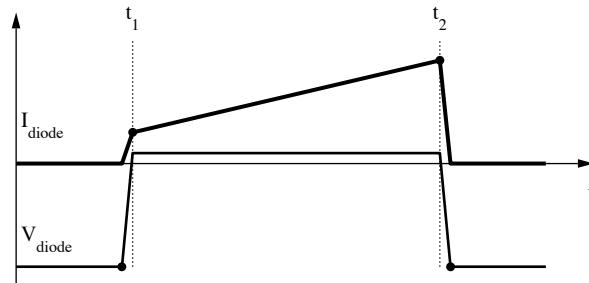
Single Semiconductor Switch Losses

The blocking voltage experienced by a single semiconductor switch is positive; therefore, switching losses are defined in the positive voltage/positive current region. Conduction losses are also defined in the positive voltage/positive current region.

Diode Losses

The voltage and current waveforms during a typical diode switching cycle are shown in the next figure. Turn on losses occur at $t = t_1$ and turn off losses at $t = t_2$. The switching energy loss in both cases is calculated by PLECS using the negative blocking voltage and positive conducting current at the switching instant. These values are shown in the figure as dots. Therefore, the lookup tables for the turn-on and turn-off switching losses must be specified in the negative voltage/positive current region.

Conduction losses occur when $t_1 < t < t_2$. During this time period, the current and voltage are both positive. Therefore the conduction loss profile must be specified in the positive voltage/positive current region.

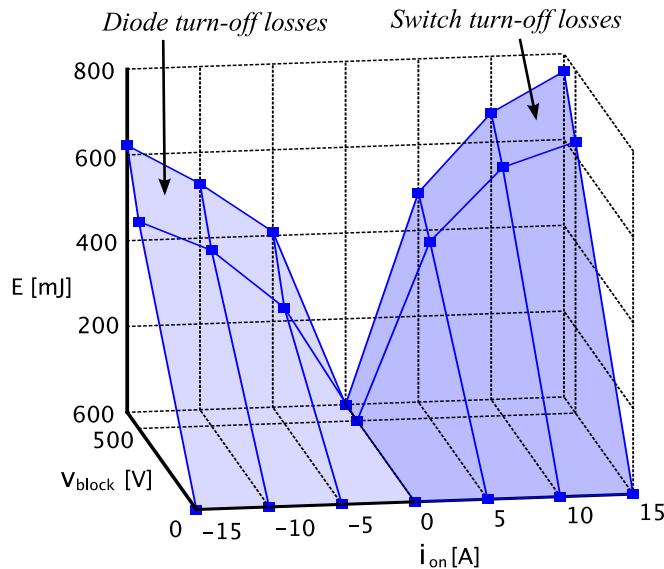


Diode voltage and current during switching

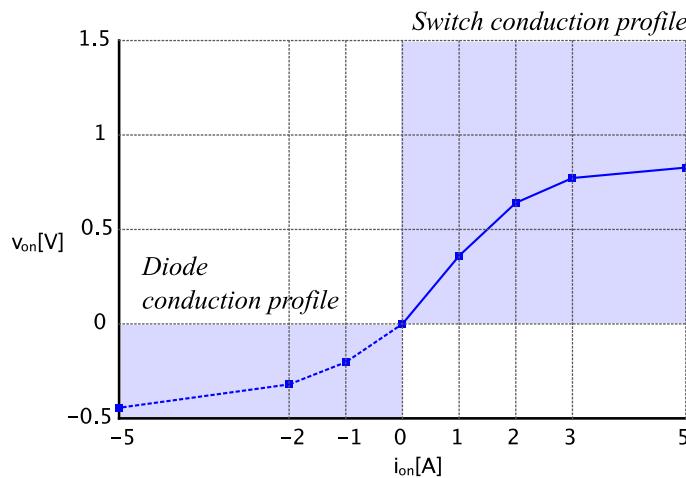
Losses of Semiconductor Switch with Diode

Semiconductor switches with an integrated diode such as the IGBT with Diode model allow losses for both the semiconductor switch and diode to be individually specified using a single set of lookup tables. The conduction and switching loss tables for the semiconductor switch are specified for the same voltage/current regions as for the single semiconductor switch without diode. Due to the polarity reversal of the diode, the diode losses are appended to the loss tables of the semiconductor switch by extending the tables in the negative voltage/negative current direction for the diode conduction losses, and in the positive voltage/negative current direction for the diode switching losses. An example turn-off loss table and conduction loss profile for a semiconductor switch with diode are shown in the next two figures. A summary of the valid voltage and current regions for defining conduction and switching losses for the different types of semiconductors is given below:

	Diode		Switch		Switch with Diode			
	Diode		Switch		Switch		Diode	
	V	I	V	I	V	I	V	I
Conduction Loss	+	+	+	+	+	+	-	-
Switching Loss	-	+	+	+	+	+	+	-



Turn-off loss lookup table for semiconductor switch with diode



Conduction loss profile for semiconductor switch with diode

Magnetic Modeling

Inductors and transformers are key components in modern power electronic circuits. Compared to other passive components they are rather difficult to model for the following reasons:

- Magnetic components, especially transformers with multiple windings can have complex geometric structures. The flux in the magnetic core may be split into several paths with different magnetic properties. In addition to the core flux, each winding has its own leakage flux.
- Core materials such as iron alloy and ferrite express a highly non-linear behavior. At high flux densities, the core material saturates leading to a greatly reduced inductor impedance. Moreover, hysteresis effects and eddy currents cause frequency-depending losses.

In PLECS, the user can build complex magnetic components in a special magnetic circuit domain. Primitives such as windings, cores and air gaps are provided in the Magnetics Library. The available core models include saturation and hysteresis. Frequency dependent losses can be modeled with magnetic resistances. Windings form the interface between the electrical and the magnetic domain.

Alternatively, less complex magnetic components such as saturable inductors and single-phase transformers can be modeled directly in the electrical domain.

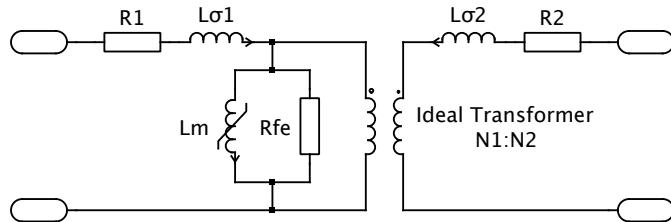
Equivalent circuits for magnetic components

To model complex magnetic structures with equivalent circuits, three different approaches exist: Coupled-inductors, the resistance-reluctance analogy and the capacitance-permeance analogy.

Coupled inductors

In the coupled inductor approach, the magnetic component is modeled directly in the electrical domain as an equivalent circuit, in which inductances represent magnetic flux paths and losses incur at resistors. Magnetic coupling between windings is realized either with mutual inductances or with ideal transformers.

Using coupled inductors, magnetic components can be implemented in any circuit simulator since only electrical components are required. This approach is most commonly used for representing standard magnetic components such as transformers. The figure below shows an example for a two-winding transformer, where $L_{\sigma 1}$ and $L_{\sigma 2}$ represent the leakage inductances, L_m the non-linear magnetization inductance and R_{fe} the iron losses. The copper resistances of the windings are modeled with R_1 and R_2 .



Transformer implementation with coupled inductors

However, the equivalent circuit bears little resemblance to the physical structure of the magnetic component. For example, parallel flux paths in the magnetic structure are modeled with series inductances in the equivalent circuit. For non-trivial magnetic components such as multiple-winding transformers or integrated magnetic components, the equivalent circuit can be difficult to derive and understand. In addition, equivalent circuits based on inductors are impossible to derive for non-planar magnetic components.

Reluctance-resistance analogy

The traditional approach to model magnetic structures with equivalent electrical circuits is the reluctance-resistance analogy. The magnetomotive force (MMF) F is regarded as analogous to voltage and the magnetic flux Φ as analogous to current. As a consequence, magnetic reluctance of the flux path \mathcal{R} corresponds to electrical resistance:

$$\mathcal{R} = \frac{\mathcal{F}}{\Phi}$$

The magnetic circuit is simple to derive from the core geometry: Each section of the flux path is represented by a reluctance and each winding becomes an MMF source.

To link the external electrical circuit with the magnetic circuit, a magnetic interface is required. The magnetic interface represents a winding and establishes a relationship between flux and MMF in the magnetic circuit and voltage v and current i at the electrical ports:

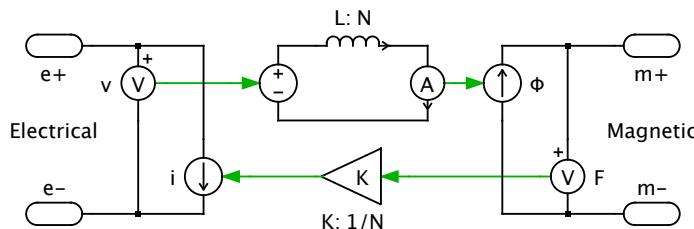
$$v = N \frac{d\Phi}{dt}$$

$$i = \frac{F}{N}$$

where N is the number of turns. If the magnetic interface is implemented with an integrator it can be solved by an ODE solver for ordinary differential equations:

$$\Phi = \frac{1}{N} \int v dt$$

The schematic below outlines a possible implementation of the magnetic interface in PLECS.



Implementation of magnetic interface

Although the reluctance-resistance duality may appear natural and is widely accepted, it is an awkward choice for multiple reasons:

- Physically, energy is stored in the magnetic field of a volume unit. In a magnetic circuit model with lumped elements, the reluctances should therefore be storage components. However, with the traditional choice of mmf and flux as magnetic system variables, reluctances are modeled as resistors, i.e. components that would usually dissipate energy. It is also confusing that the magnetic interface is a storage component.

- To model energy dissipation in the core material, inductors must be employed in the magnetic circuit, which is even less intuitive.
- Magnetic circuits with non-linear reluctances generate differential-algebraic equations (DAE) resp. algebraic loops that cannot be solved with the ODE solvers offered in PLECS.
- The use of magnetic interfaces results in very stiff system equations for closely coupled windings.

Permeance-capacitance analogy

To avoid the drawbacks of the reluctance-resistance analogy the alternative permeance-capacitance analogy is most appropriate. Here, the MMF F is again the across-quantity (analogous to voltage), while the *rate-of-change* of magnetic flux $\dot{\Phi}$ is the through-quantity (analogous to current). With this choice of system variables, magnetic permeance \mathcal{P} corresponds to capacitance:

$$\dot{\Phi} = \mathcal{P} \frac{dF}{dt}$$

Hence it is convenient to use permeance \mathcal{P} instead of the reciprocal reluctance \mathcal{R} to model flux path elements. Because permeance is modeled with storage components, the energy relationship between the actual and equivalent magnetic circuit is preserved. The permeance value of a volume element is given by:

$$\mathcal{P} = \frac{1}{\mathcal{R}} = \frac{\mu_0 \mu_r A}{l}$$

where $\mu_0 = 4\pi \times 10^{-7} \text{ N/A}^2$ is the magnetic constant, μ_r is the relative permeability of the material, A is the cross-sectional area and l the length of the flux path.

Magnetic resistors (analogous to electrical resistors) can be used in the magnetic circuit to model losses. They can be connected in series or in parallel to a permeance component, depending on the nature of the specific loss. The energy relationship is maintained as the power

$$P_{\text{loss}} = F \dot{\Phi}$$

converted into heat in a magnetic resistor corresponds to the power loss in the electrical circuit.

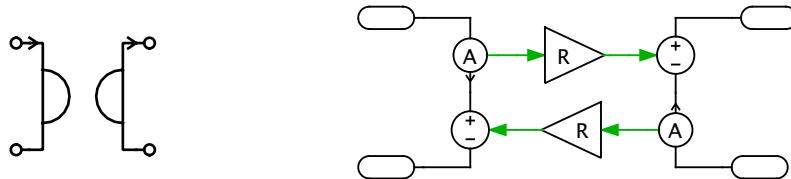
Windings form the interface between the electrical and the magnetic domain. A winding of N turns is described with the equations below. The left-hand

side of the equations refers to the electrical domain, the right-hand side to the magnetic domain.

$$v = N\dot{\Phi}$$

$$i = \frac{F}{N}$$

Because a winding converts through-quantities ($\dot{\Phi}$ resp. i) in one domain into across-quantities (v resp. F) in the other domain, it can be implemented with a gyrator, in which N is the gyrator resistance R . The figure below shows the symbol for a gyrator and a possible implementation in PLECS.



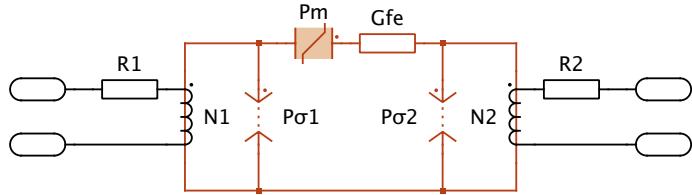
Gyrator symbol and implementation

In principle, the gyrator component could be used with regular capacitors to build magnetic circuits. However, neither the gyrator symbol nor the capacitor adequately resemble a winding respectively a flux path. Moreover, any direct connection between the electrical and magnetic domain made by mistake would lead to non-causal systems that are very difficult to debug. Therefore, dedicated magnetic components should be used when modeling magnetic circuits.

Magnetic Circuit Domain in PLECS

The magnetic domain provided in PLECS is based on the permeance-capacitance analogy. The magnetic library comprises windings, constant and variable permeances as well as magnetic resistors. By connecting them according to the physical structure the user can create equivalent circuits for arbitrary magnetic components. The two-winding transformer from above will look like the schematic below when modeled in the magnetic domain.

$\mathcal{P}_{\sigma 1}$ and $\mathcal{P}_{\sigma 2}$ represent the permeances of the leakage flux path, L_m the non-linear permeance of the core, and G_{fe} dissipates the iron losses. The winding resistances R_1 and R_2 are modeled in the electrical domain.



Transformer implementation in the magnetic domain

Modeling Non-Linear Magnetic Material

Non-linear magnetic material properties such as saturation and hysteresis can be modeled using the variable permeance component. The permeance is determined by the signal fed into the input of the component. The flux-rate through a variable permeance $\mathcal{P}(t)$ is governed by the equation:

$$\dot{\Phi} = \frac{d}{dt} (\mathcal{P} \cdot F) = \mathcal{P} \cdot \frac{dF}{dt} + \frac{d}{dt} \mathcal{P} \cdot F$$

Since F is the state variable the equation must be solved for $\frac{dF}{dt}$. Therefore, the control signal must provide the values of both $\mathcal{P}(t)$ and $\frac{d}{dt} \mathcal{P}(t)$.

The control signals must also provide the flux $\Phi(t)$ through the permeance. This enables the solver to enforce Kirchhoff's current law for all branches k of a node:

$$\sum_{k=1}^n \Phi_k = 0$$

When specifying the characteristic of a non-linear permeance, we need to distinguish carefully between the *total* permeance $\mathcal{P}_{\text{tot}}(F) = \Phi/F$ and the *differential* permeance $\mathcal{P}_{\text{diff}}(F) = d\Phi/dF$.

If the *total* permeance $\mathcal{P}_{\text{tot}}(F)$ is known the flux-rate $\dot{\Phi}$ through a time-varying permeance is calculated as:

$$\begin{aligned}
\dot{\Phi} &= \frac{d\Phi}{dt} \\
&= \frac{d}{dt} (\mathcal{P}_{\text{tot}} \cdot F) \\
&= \mathcal{P}_{\text{tot}} \cdot \frac{dF}{dt} + \frac{d\mathcal{P}_{\text{tot}}}{dt} \cdot F \\
&= \mathcal{P}_{\text{tot}} \cdot \frac{dF}{dt} + \frac{d\mathcal{P}_{\text{tot}}}{dF} \cdot \frac{dF}{dt} \cdot F \\
&= \left(\mathcal{P}_{\text{tot}} + \frac{d\mathcal{P}_{\text{tot}}}{dF} \cdot F \right) \cdot \frac{dF}{dt}
\end{aligned}$$

In this case, the control signal for the variable permeance component is:

$$\begin{bmatrix} \mathcal{P}(t) \\ \frac{d}{dt}\mathcal{P}(t) \\ \Phi(t) \end{bmatrix} = \begin{bmatrix} \mathcal{P}_{\text{tot}} + \frac{d}{dF}\mathcal{P}_{\text{tot}} \cdot F \\ 0 \\ \mathcal{P}_{\text{tot}} \cdot F \end{bmatrix}$$

In most cases, however, the *differential* permeance $\mathcal{P}_{\text{diff}}(F)$ is provided to characterize magnetic saturation and hysteresis. With

$$\begin{aligned}
\dot{\Phi} &= \frac{d\Phi}{dt} \\
&= \frac{d\Phi}{dF} \cdot \frac{dF}{dt} \\
&= \mathcal{P}_{\text{diff}} \cdot \frac{dF}{dt} ,
\end{aligned}$$

the control signal is

$$\begin{bmatrix} \mathcal{P}(t) \\ \frac{d}{dt}\mathcal{P}(t) \\ \Phi(t) \end{bmatrix} = \begin{bmatrix} \mathcal{P}_{\text{diff}} \\ 0 \\ \mathcal{P}_{\text{tot}} \cdot F \end{bmatrix}$$

Saturation Curves for Soft-Magnetic Material

Curve fitting techniques can be employed to model the properties of ferromagnetic material. As an example, a saturation curve adapted from the modified Langevin equation for bulk magnetization without interdomain coupling is used, which is referred to as the coth function:

$$B = B_{\text{sat}} \left(\coth \frac{3H}{a} - \frac{a}{3H} \right) + \mu_{\text{sat}} H$$

The coth function has three degrees of freedom which are set by the coefficients B_{sat} , a and μ_{sat} . These coefficients can be found e.g. using a least-squares fitting procedure. Calculating the derivate of B with respect to H yields

$$\frac{dB}{dH} = B_{\text{sat}} \left(\frac{\tanh^2(H/a) - 1}{a \tanh^2(H/a)} - \frac{a}{H^2} \right) + \mu_{\text{sat}}$$

With the relationships $\Phi = B \cdot A$ and $F = H \cdot l$ the control signal $\mathcal{P}_{\text{diff}}$ for the variable permeance is easily derived from the equation above.

References

- S. El-Hamamsy and E. Chang, "Magnetics modeling for computer-aided design of power electronics circuits," in *Power Electronics Specialists Conference*, vol. 2, pp. 635–645, 1989.
- R. W. Buntenbach, "Improved circuit models for inductors wound on dissipative magnetic cores," in *Proc. 2nd Asilomar Conf. Circuits Syst.*, Pacific Grove, CA, Oct. 1968, pp. 229–236 (IEEE Publ. No. 68C64-ASIL).
- R. W. Buntenbach, "Analogs between magnetic and electrical circuits," in *Electron. Products*, vol. 12, pp. 108–113, 1969.
- D. Hamill, "Lumped equivalent circuits of magnetic components: the gyrator-capacitor approach," in *IEEE Transactions on Power Electronics*, vol. 8, pp. 97–103, 1993.
- D. Hamill, "Gyrator-capacitor modeling: A better way of understanding magnetic components," in *APEC Conference Proceedings* pp. 326–332, 1994.

6

Analysis Tools

Steady-State Analysis

Many specifications of a power electronic system are often given in terms of steady-state characteristics. A straight-forward way to obtain the steady-state operating point of a system is to simulate over a sufficiently long time-span until all transients have faded out. The drawback of this brute-force approach is that it can be very time consuming. Usually a system has time constants that are much longer than the switching period. This applies in particular to electro-thermal models.

Algorithm

The steady-state analysis of a periodic system is based on a quasi-Newton method with Broyden's update. In this approach the problem is formulated as finding the roots of the function

$$\mathbf{f}(\mathbf{x}) = \mathbf{x} - \mathbf{F}_T(\mathbf{x})$$

where \mathbf{x} is an initial vector of state variables and $\mathbf{F}_T(\mathbf{x})$ is the final vector of state variables one period T later.

Evaluating $\mathbf{f}(\mathbf{x})$ or $\mathbf{F}_T(\mathbf{x})$ therefore involves running a simulation from t_{start} to $t_{\text{start}} + T$. The period, T , must be the least common multiple of the periods of all sources (signal or electrical) in the model.

The above problem can be solved iteratively using

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}_k^{-1} \cdot \mathbf{f}(\mathbf{x}_k) \quad , \quad \mathbf{J}_k = \left. \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}_k}$$

The Jacobian \mathbf{J} is calculated numerically using finite differences. If n is the number of state variables, calculating the Jacobian requires $n + 1$ simulation

runs where each state variable in turn is slightly perturbed and the difference between the perturbed and unperturbed solution is computed to obtain one column of \mathbf{J} :

$$\mathbf{j}_i = \frac{\mathbf{f}(\mathbf{x} + \Delta\mathbf{x}_i) - \mathbf{f}(\mathbf{x})}{|\Delta\mathbf{x}_i|}, \quad i = 1 \dots n$$

Because this is computationally expensive, only the first Jacobian is actually computed this way. In subsequent iterations, the Jacobian is updated using Broyden's method, which does not require any additional simulations.

The convergence criterion of the iterations is based on the requirement that both the maximum relative error in the state variables and the maximum relative change from one iteration to the next are smaller than a certain limit *rtol*:

$$\left| \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{\mathbf{x}_k} \right| < \text{rtol} \quad \text{and} \quad \frac{|f_i(\mathbf{x})|}{\max|x_i(\tau)|} < \text{rtol} \quad \text{for all } i = 1, \dots, n$$

A steady-state analysis comprises the following steps:

- 1** Simulate until the final switch positions after one cycle are equal to the initial switch positions. This is called a *circular topology*.
- 2** Calculate the Jacobian matrix \mathbf{J}_0 for the initial state.
- 3** Iterate until the convergence criterion is satisfied. If during the iterations the final switch positions after one cycle differ from the initial switch positions, go back to step **1**.

Fast Jacobian Calculation for Thermal States

To reduce the number of simulation runs and thus save computation time PLECS can calculate the Jacobian matrix entries pertaining to thermal states directly from the state-space matrices rather than using finite differences.

There is a certain error involved with this method since it neglects the feedback from the thermal states to the electrical states (or Simulink states). While this will not affect the accuracy of the final result of the steady-state analysis it may slow down the convergence. Normally, however, the overall performance will be much faster than calculating the full Jacobian matrix.

The calculation method is controlled by the parameter `JacobianCalculation` (see below).

Limitations

Hidden state variables

In the PLECS Blockset, the steady-state analysis depends on the fact that a model can be completely initialized with the `InitialState` parameter of the `sim` command. However, certain Simulink blocks that clearly have an internal memory do not store this memory in the state vector and therefore cannot be initialized. Among these blocks are the Memory block, the Relay block, the Transport Delay block and the Variable Transport Delay block. If a model contains any block with hidden states, the algorithm may be unable to find a solution.

State variable windup

If the effect of a state variable on the system is limited in some way but the state variable itself is not limited, it might wind up towards infinity. In this case the algorithm may fail to converge or return a false solution. In order to avoid this problem you should limit the state variable itself, e.g. by enabling the **Limit output** checkbox of an Integrator block.

Reference

- D. Maksimović, "Automated steady-state analysis of switching power converters using a general-purpose simulation tool", Proc. IEEE Power Electronics Specialists Conference, June 1997, pp. 1352-1358.

AC Analysis

The AC Analysis uses the Steady-State Analysis to compute the transfer function of a periodic system at discrete analysis frequencies. For each frequency the following steps are executed:

- 1** Apply a sinusoidal perturbation to the system under study.
- 2** Find the periodic steady-state operating point of the perturbed system.
- 3** Extract the system response at the perturbation frequency using Fourier analysis.

The perturbation frequencies are defined by specifying the sweep range and the number of points to be placed within this range on a linear or logarithmic scale.

Note The period length of the perturbed system is the least common multiple of the unperturbed system period and the perturbation period. In order to keep this number and thus the simulation time small the algorithm may slightly adjust the individual perturbation frequencies.

Impulse Response Analysis

An alternative and faster method to determine the open loop transfer function of a system is the Impulse Response Analysis. Instead of perturbing a system with sinusoidal stimuli of different frequencies, one at a time, a single impulse is applied when the system is in steady state. The system transfer function can then be calculated very efficiently over a wide frequency range (from zero to half the system frequency) by computing the Laplace transform of the transient impulse response.

Algorithm

The impulse response analysis is performed in three steps:

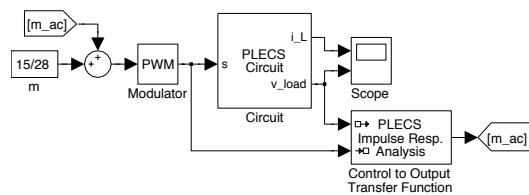
- 1** Find the steady-state operating point of the system under study.
- 2** Apply a perturbation in form of a discrete impulse for the duration of one period.
- 3** Calculate the Laplace transform of the transient impulse response.

Compensation for Discrete Pulse

Theoretically, in order to compute the system transfer function from the Laplace transform of the system response, the system must be perturbed with a unit Dirac impulse (also known as delta function). This is not practical for numerical analysis, so the algorithm applies a finite rectangular pulse instead.

For transfer functions such as the line-to-output transfer function or the output impedance this can be compensated for by dividing the Laplace transform of the system response by the Laplace transform of the rectangular pulse. This is achieved by setting the parameter **Compensation for discrete pulse** to **discrete pulse**, which is the default.

However, when calculating control-to-output transfer functions that involve the duty cycle of a switched converter, the rectangular input signal interferes with the sampling of the modulator. In this case the compensation type should be set to **external reference**. This causes the Impulse Response Analysis block to have two input signals that should be connected as shown in this figure.



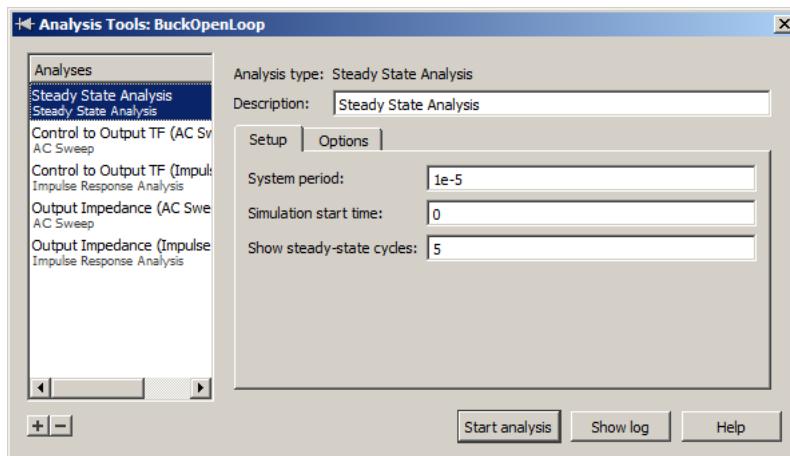
Finally, you can set the compensation type to **none** which means that the computed transfer function is taken as is. Use this setting if the modulator uses regular sampling and the sampling period is identical to the system period.

Reference

- D. Maksimović, "Automated small-signal analysis of switching power converters using a general-purpose time-domain simulator", Proc. Applied Power Electronics Conference, February 1998.

Usage in PLECS Standalone

In PLECS Standalone all analyses are managed in the Analysis Tools Dialog shown below. To open the dialog, select **Analysis tools...** from the **Simulation** menu of the schematic editor.



The left hand side of the dialog window shows a list of the analyses that are currently configured for the model. To add a new analysis, click the button marked + below the list and select the desired analysis type. To remove the currently selected analysis, click on the button marked -. You can reorder the analyses by clicking and dragging an entry up and down in the list.

The right hand side of the dialog window shows the parameter settings of the currently selected analysis. Each analysis must have a unique **Description**. The other parameters available for the different analysis types are described further below.

The button **Start analysis/Abort analysis** starts the currently selected analysis or aborts the analysis that is currently running. The button **Show log/Hide log** shows or hides a log window that displays the progress of an analysis and diagnostic messages.

Steady-State Analysis

System period

The system period is the least common multiple of the periods of all sources (signal or electrical) in the model. If the parameter setting does

not reflect the true system period or an integer multiple thereof, the analysis will yield meaningless results or fail to converge altogether.

Simulation start time

The start time t_{start} to be used in the transient simulation runs. Simulations run from t_{start} to $t_{\text{start}} + T$, where T is the system period specified above. The default is 0.

Show steady-state cycles

The number of steady-state cycles, for which a transient simulation is run at the end of an analysis. The default is 1.

Number of init. cycles

The number of cycle-by-cycle simulations to be performed *before* the Newton iterations are started. When an analysis fails to converge because the starting point was too far from the steady-state solution, this parameter can help to get better starting conditions. The default is 0.

Termination tolerance

The relative error bound. The analysis continues until both the maximum relative error in the state variables and the maximum relative change from one iteration to the next are smaller than this bound for each state variable.

Max. number of iterations

Maximum number of Newton iterations allowed.

Rel. perturbation for Jacobian

Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.

Jacobian calculation

Controls whether Jacobian matrix entries for thermal state variables are calculated via finite differences (**full**) or directly from the state-space matrices (**fast**). The default is **fast**.

AC Sweep

In order to perform an AC sweep, you need to insert a Small Signal Perturbation (see page 393) and a Small Signal Response (see page 394) block in order to define the points at which the perturbation is injected and the response is measured. The Small Signal Gain (see page 392) block can be used to obtain the closed loop gain of a feedback loop.

At the end of an analysis, a scope window will open and display the Bode diagram of the transfer function. You can also open the scope manually by clicking one **Show results** button.

System period

The system period is the least common multiple of the periods of all sources (signal or electrical) in the model. If the parameter setting does not reflect the true system period or an integer multiple thereof, the analysis will yield meaningless result or fail to converge altogether.

Frequency range

A vector containing the lowest and highest perturbation frequency.

Amplitude

A vector containing the amplitudes of the perturbation signal at the lowest and highest frequency. The amplitudes at intermediate frequencies are interpolated linearly. If a scalar is entered, the amplitude will be constant for all frequencies.

Perturbation

The Small Signal Perturbation block that will be active during the analysis. All other perturbations blocks will output 0.

Response

The Small Signal Response block that will record the system response during the analysis.

Simulation start time

The start time t_{start} to be used in the transient simulation runs. Simulations run from t_{start} to $t_{start} + T$, where T is the system period specified above. The default is 0.

Frequency scale

Specifies whether the sweep frequencies should be distributed on a linear or logarithmic scale.

Number of points

The number of automatically distributed frequencies.

Additional frequencies

A vector specifying frequencies to be swept *in addition* to the automatically distributed frequencies.

For a description of the steady-state options please refer to “Steady-State Analysis” (on page 110).

Impulse Response Analysis

In order to perform an impulse response analysis, you need to insert a Small Signal Perturbation (see page 393) and a Small Signal Response (see page 394) block in order to define the points at which the perturbation is injected and the response is measured.

At the end of an analysis, a scope window will open and display the Bode diagram of the transfer function. You can also open the scope manually by clicking one **Show results** button.

For a description of the parameters please refer to “AC Sweep” (on page 111). In an impulse response analysis, the computational effort for an individual frequency is very cheap. Therefore, the parameter **Additional frequencies** is omitted; instead, the **Number of points** can be set to a large value in order obtain smooth curves.

Extraction of State-Space Matrices

PLECS allows you to extract the state-space matrices describing the linear portion of a circuit model for a given combination of switch positions. The commands used for this purpose are listed below. These commands can be used both in a Simulation Script (see page 155) and on the Octave console. In each of the commands *circuit* is the name of the circuit model.

```
names = plecs('get', circuit, 'StateSpaceOrder');
```

returns a struct containing the names of the components associated with the circuit model's inputs, outputs, states and switches.

```
plecs('set', circuit, 'SwitchVector', switchpos);
```

sets the vector of switch positions for the subsequent analysis to *switchpos*.

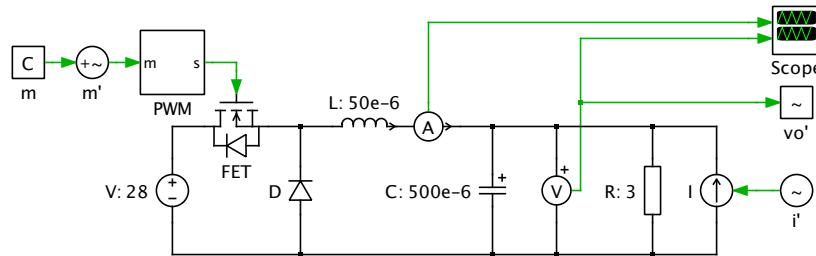
```
t = plecs('get', circuit, 'Topology');
```

returns a struct with the state space matrices **A**, **B**, **C**, **D** and **I** for the vector of switch positions specified by the previous command. The matrix **I** is the identity matrix if all electrical states are independent. Otherwise it specifies the relationship between the dependent variables.

Above commands can also be invoked via the XML-RPC interface (see page 159) using an analogous syntax.

Application Example

The demo model BuckOpenLoop implements the buck converter shown below. It operates at a switching frequency of 100 kHz with a fixed duty-cycle of 15/28. To run a transient simulation from zero initial conditions, select **Start** from the **Simulation** menu.



To view the analyses configured in this model select **Analysis tools...** from the **Simulation** menu. The only periodic source in the model is the carrier signal used in the modulator. Hence, the parameter **System period** for all analyses is specified as $T = 1/100 \text{ kHz} = 10^{-5} \text{ s}$.

Steady-State Operation

To view the steady-state operation of the converter, select **Steady-State Analysis** from the list and click on **Start analysis**. After the analysis has found the periodic operating point, the scope will show five steady-state cycles.

Control-to-Output Transfer Function

For the calculation of the control-to-output transfer function, a small perturbation needs to be added to the modulation index. This is done with the Small Signal Perturbation block m' , which has the **Show feed-through input** setting enabled. The system output in this case is defined as the output voltage of the converter. The output signal of the voltmeter is therefore connected to the Small Signal Response block vo' .

To calculate the transfer function using the AC Sweep, select **Control to Output TF (AC Sweep)** from the list and click on **Start analysis**. The analysis sweeps the frequency range between 100 Hz and 50 kHz. 21 points are placed logarithmically within this range; to obtain a smoother output, additional data points are generated between 800 and 1400 Hz.

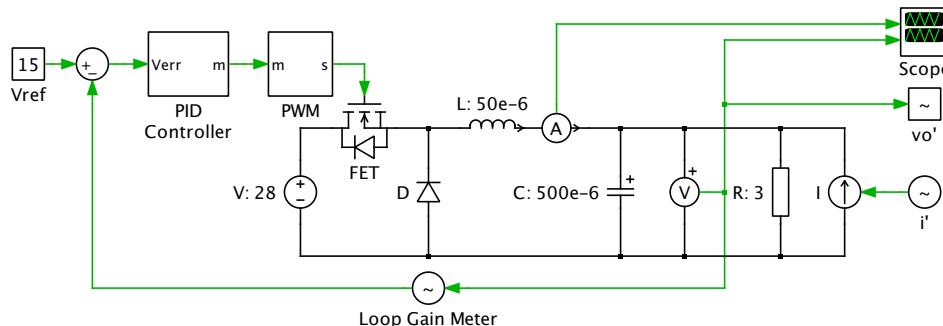
To calculate the transfer function using the Impulse Response Analysis, select **Control to Output TF (Impulse Response)** from the list and click on **Start analysis**.

Output Impedance

For the calculation of the output impedance, a small perturbation current is injected into the converter output using a current source that is controlled by the Small Signal Perturbation block i' and the output voltage response is measured. As above, two analyses have been configured that calculate the impedance using the AC Sweep and the Impulse Response Analysis.

Loop Gain

The demo model BuckClosedLoop implements the controlled buck converter shown below. A PID controller regulates the output voltage to 15 volts.



For the calculation of the voltage loop gain, the Small Signal Gain block Loop Gain Meter has been inserted into the feedback path. If you look under the mask of the Small Signal Gain, you can see how the block both injects a small perturbation and measures the system response.

To calculate the loop gain, select **Analysis tools...** from the **Simulation** menu, then choose **Closed Loop Gain** from the list of analyses and click **Start analysis**.

Usage in the PLECS Blockset

In the PLECS Blockset, you configure analyses by copying the appropriate blocks from the **Analysis Tools** library in **PLECS Extras** into your model.

Steady-State Analysis

To perform a steady-state analysis, copy the Steady-State Analysis block (see page 492) into your model. An analysis can be run interactively from the block dialog or via a MATLAB command. The calling syntax is

```
plsteadystate(block);
```

where *block* is the Simulink handle or the full block path of the Steady-State Analysis block. The block handle or path can be followed by parameter/value pairs. Otherwise, the settings specified in the block dialog are used.

The following table lists the parameters of the Steady-State Analysis block. The **Parameter** column shows the parameter names to be used with the `plsteadystate` command. The **Description** column indicates whether and where you can set the value in the dialog box. Parameters that are not accessible in the dialog box can be modified using the `set_param` command.

Steady-State Analysis Parameters

Parameter	Description
TimeSpan	For a fixed system period, the period length; this is the least common multiple of the periods of independent sources in the system. For a variable system period, the maximum time span during which to look for a trigger event marking the end of a period. Set by the System period length/Max simulation time span field.
TStart	Simulation start time. Set by the Simulation start time field.

Steady-State Analysis Parameters (contd.)

Parameter	Description
Tolerance	Relative error tolerance used in the convergence criterion. Set by the Termination tolerance field.
MaxIter	Maximum number of iterations allowed. Set by the Max number of iterations field.
Display	Specifies the level of detail of the diagnostic messages displayed in the command window (iteration, final, off). Set by the Display drop-down list.
HideScopes	Hide all Simulink scope windows during an analysis in order to save time.
HiddenStates	Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (error, warning, none). Set by the Hidden model states drop-down list.
FinalStateName	Name of a MATLAB variable used to store the steady-state vector at the end of an analysis. Set by the Steady-state variable field.
NCycles	Number of steady-state cycles that should be simulated at the end of an analysis. Set by the Show steady-state cycles field.
JPert	Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.
JacobianCalculation	Controls the way the Jacobian matrix is calculated (full, fast). The default is fast.
NInitCycles	Number of cycle-by-cycle simulations that should be performed before the actual steady-state analysis. This parameter can be used to provide the algorithm with a better starting point. The default is 0.

These examples show how to run analyses for the block Steady State in the model `mymodel`:

```
plsteadystate('mymodel/Steady State');
```

starts an analysis using the parameters specified in the dialog box.

```
plsteadystate('mymodel/Steady State','TStart',0,...  
'FinalStateName','x0');  
plsteadystate('mymodel/Steady State','TStart',1,...  
'FinalStateName','x1');
```

performs two analyses with different start times and assigns the resulting steady-state vectors to two different variables `x0` and `x1`. This is useful e.g. if the model has a reference signal with a step change and you want to determine the steady state before and after the change.

AC Sweep / Loop Gain Analysis

To perform an AC sweep, copy the AC Sweep block (see page 484) into your model. The block outputs a perturbation signal, which must be injected into the system. The system response must be fed back into the block input.

To perform a loop gain analysis, copy the Loop Gain Analysis block (see page 490) into your model and insert it into the path of a feedback loop.

An analysis can be run interactively from the block dialogs or via a MATLAB command. The calling syntax is

```
placsweep(block);
```

where *block* is the Simulink handle or the full block path of the AC Sweep or Loop Gain Analysis block. The block handle or path can be followed by parameter/value pairs. Otherwise, the settings specified in the block dialog are used.

The following table lists the parameters of the AC Sweep and Loop Gain Analysis blocks. The **Parameter** column shows the parameter names to be used with the `placsweep` command. The **Description** column indicates whether and where you can set the value in the dialog box. Parameters that are not accessible in the dialog box can be modified using the `set_param` command.

AC Analysis Parameters

Parameter	Description
TimeSpan	Period length of the unperturbed system. Set by the System period length field.
TStart	Simulation start time. Set by the Simulation start time field.
FreqRange	Range of the perturbation frequencies. Set by the Frequency sweep range field.
FreqScale	Specifies whether the sweep frequencies should be distributed on a linear or logarithmic scale. Set by the Frequency sweep scale field.
NPoints	Number of data points generated. Set by the Number of points field.
InitialAmplitude	Perturbation amplitude at the first perturbation frequency. Set by the Amplitude at first freq field.
Method	Method used for obtaining the periodic steady-state operating point of the perturbed system: Brute force simulation - start from model initial state, Brute force simulation - start from unperturbed steady state, Steady-state analysis - start from model initial state, Steady-state analysis - start from unperturbed steady state. Set by the Method drop-down list.
Tolerance	Relative error tolerance used in the convergence criterion. Set by the Termination tolerance field.
MaxIter	Maximum number of iterations allowed. Set by the Max number of iterations field.
Display	Specifies the level of detail of the diagnostic messages displayed in the command window (iteration, final, off). Set by the Display drop-down list.

AC Analysis Parameters (contd.)

Parameter	Description
HideScopes	Hide all Simulink scope windows during an analysis in order to save time.
HiddenStates	Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (error, warning, none). Set by the Hidden model states drop-down list.
OutputName	Name of a MATLAB variable used to store the transfer function at the end of an analysis. Set by the Output variable field.
BodePlot	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the Plot Bode diagram drop-down list.
JPert	Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.
NInitCycles	If a steady-state analysis is used to obtain the starting point of the ac analysis (see parameter Method above), this parameter specifies the number of cycle-by-cycle simulations that should be performed before the steady-state analysis. This parameter can be used to provide the algorithm with a better starting point. The default is 0.

These examples show how to run analyses for the block AC Sweep in the model mymodel:

```
placsweep('mymodel/AC Sweep');
```

starts an analysis using the parameters specified in the dialog box.

```
placsweep('mymodel/AC Sweep','TStart',0,...  
'OutputName','T0');  
placsweep('mymodel/AC Sweep','TStart',1,...  
'OutputName','T1');
```

performs two analyses with different start times and assigns the resulting transfer functions to two different variables T_0 and T_1 . This is useful e.g. if the model has a reference signal with a step change and you want to determine the transfer function before and after the change.

Impulse Response Analysis

To perform an impulse response analysis, copy the Impulse Response Analysis block (see page 488) into your model. The block outputs a perturbation signal, which must be injected into the system. The system response must be fed back into the block input.

An analysis can be run interactively from the block dialogs or via a MATLAB command. The calling syntax is

```
plimpulseresponse(block);
```

where *block* is the Simulink handle or the full block path of the Impulse Response Analysis block. The block handle or path can be followed by parameter/value pairs. Otherwise, the settings specified in the block dialog are used.

The following table lists the parameters of the Impulse Response Analysis block. The **Parameter** column shows the parameter names to be used with the `plimpulseresponse` command. The **Description** column indicates whether and where you can set the value in the dialog box. Parameters that are not accessible in the dialog box can be modified using the `set_param` command.

Impulse Response Analysis Parameters

Parameter	Description
TimeSpan	Period length of the unperturbed system. Set by the System period length field.
TStart	Simulation start time. Set by the Simulation start time field.
FreqRange	Range of the perturbation frequencies. Set by the Frequency sweep range field.

Impulse Response Analysis Parameters (contd.)

Parameter	Description
FreqScale	Specifies whether the sweep frequencies should be distributed on a linear or logarithmic scale. Set by the Frequency sweep scale field.
NPoints	Number of data points generated. Set by the Number of points field.
Perturbation	Perturbation amplitude of the discrete impulse. Set by the Perturbation field.
Compensation	Specifies whether and how the effect of the sampling should be compensated (none, discrete pulse, external reference). Set by the Compensation for discrete pulse drop-down list.
Tolerance	Relative error tolerance used in the convergence criterion of the initial steady-state analysis. Set by the Termination tolerance field.
MaxIter	Maximum number of iterations allowed during the initial steady-state analysis. Set by the Max number of iterations field.
Display	Specifies the level of detail of the diagnostic messages displayed in the command window (iteration, final, off). Set by the Display drop-down list.
HideScopes	Hide all Simulink scope windows during an analysis in order to save time.
HiddenStates	Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (error, warning, none). Set by the Hidden model states drop-down list.
OutputName	Name of a MATLAB variable used to store the transfer function at the end of an analysis. Set by the Output variable field.

Impulse Response Analysis Parameters (contd.)

Parameter	Description
BodePlot	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the Plot Bode diagram drop-down list.
JPert	Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.
NInitCycles	Number of cycle-by-cycle simulations that should be performed before the initial steady-state analysis. This parameter can be used to provide the algorithm with a better starting point. The default is 0.

Extraction of State-Space Matrices

PLECS allows you to extract the state-space matrices describing the linear portion of a circuit model for a given combination of switch positions. The commands used for this purpose are listed below. In each of the commands *circuit* is the full Simulink path of a PLECS Circuit block.

```
names = plecs('get', circuit, 'StateSpaceOrder');
```

returns a struct containing the names of the components associated with the circuit model's inputs, outputs, states and switches.

```
plecs('set', circuit, 'SwitchVector', switchpos);
```

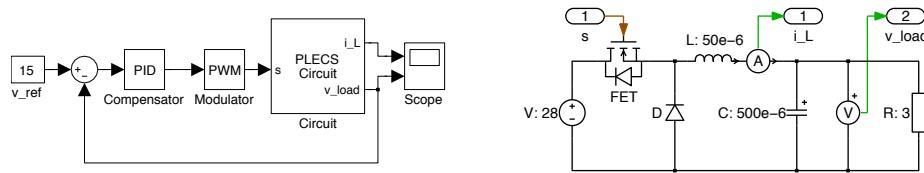
sets the vector of switch positions for the subsequent analysis to *switchpos*.

```
t = plecs('get', circuit, 'Topology');
```

returns a struct with the state space matrices **A**, **B**, **C**, **D** and **I** for the vector of switch positions specified by the previous command. The matrix **I** is the identity matrix if all electrical states are independent. Otherwise it specifies the relationship between the dependent variables.

Application Example

This section demonstrates the application of the analysis tools in the PLECS Blockset for the design of the regulated buck converter system operating at a switching frequency of 100 kHz shown in the figure below. The converter shall supply a regulated 15 volts to a resistive load at a nominal load current of 5 amperes.



The examples used in this section follow the design example in [Erickson], chapter 9. They have been implemented in the demo models `p1BuckSweep`, `p1BuckImpulseResponse` and `p1BuckLoop`.

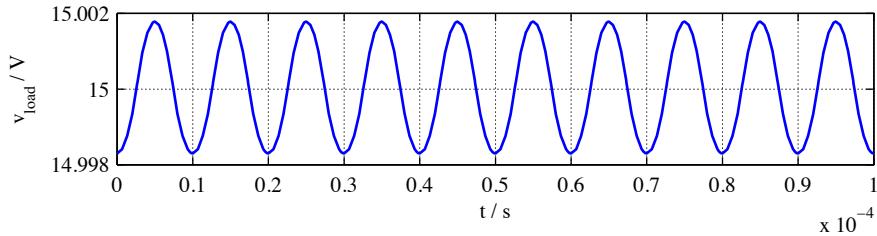
Steady-State Analysis

We first examine the open-loop behavior of the system. In order to get the desired output voltage we need to apply a fixed duty-cycle of $V_{out}/V_{src} = 15V/28V$. You can verify this by using the Steady-State Analysis block to obtain the steady-state waveform of the output voltage.

For this purpose you copy the block into the model and double-click it to open the dialog box. The parameter **System period length** is already set to the correct value, i.e. $1e-5$. Set the parameter **Show steady-state cycles** to e.g. 10 so that you can more easily check that the system is indeed in the steady state when the analysis finishes. Then click on **Start analysis**. The algorithm should converge after the first iteration, and the scope should show the waveform in the figure below.

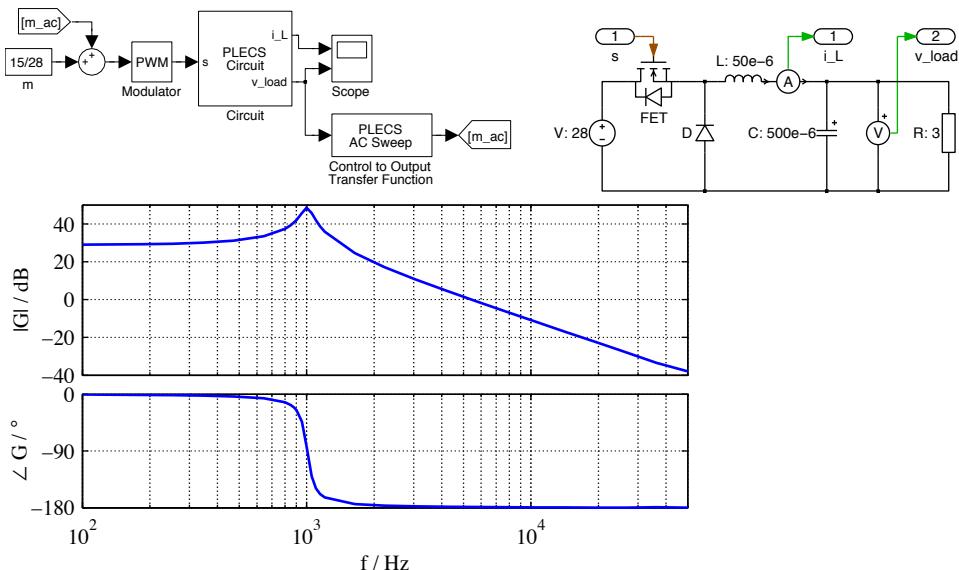
AC Sweep

Open-loop control-to-output transfer function In order to determine the control-to-output transfer function you need to perturb the steady-state duty-cycle and measure the corresponding perturbation of the output voltage. This

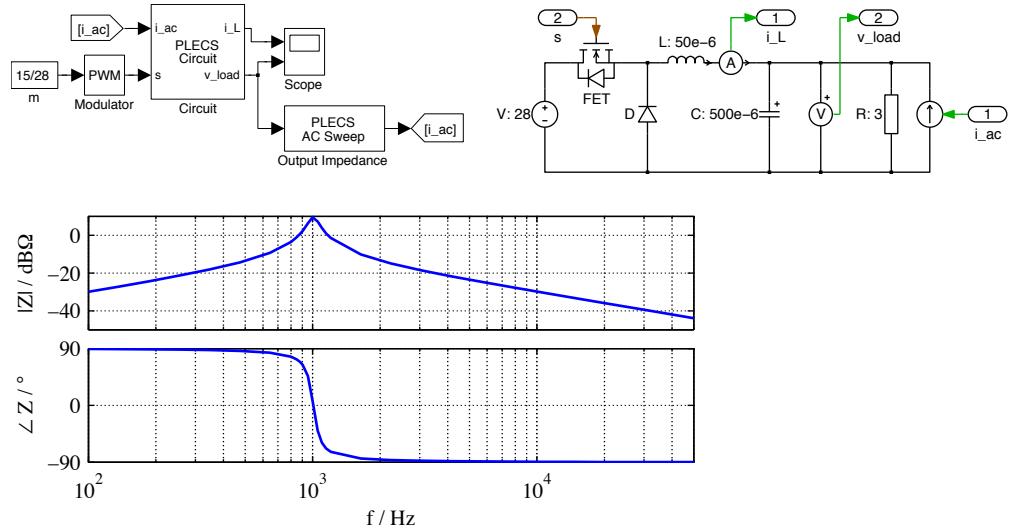
**Steady-state output voltage**

is achieved by connecting an AC Sweep block as shown below. The block output is the perturbation signal; it is added to the steady-state duty cycle. The block input is connected to the load voltage signal.

The initial amplitude of the perturbation is set to $1e-3$ which is approx. $2/1000$ of the duty cycle. We want to sweep a frequency range between 100Hz and 50kHz with a few extra points between 800Hz and 1200Hz. This is achieved by setting the parameter to [100 800:50:1200 50000]. As expected, the resulting bode plot of the transfer function shows a double pole at $f_0 = 1/(2\pi\sqrt{LC}) \approx 1\text{kHz}$ and a dc gain of $G_0 = 28\text{V} \approx 29\text{dB}$.

**Open-loop control-to-output transfer function**

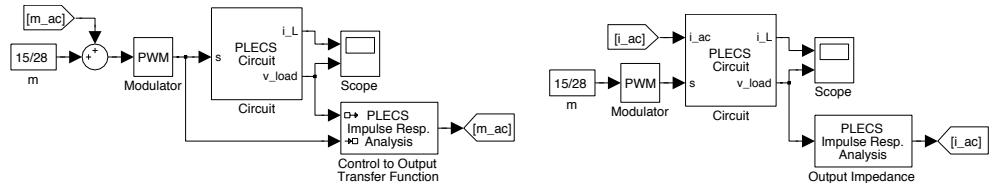
Open-loop output impedance Although not required for the compensator design we will now calculate the output impedance for demonstration purposes. To do so we need to inject a small ac current into the converter output and measure the resulting perturbation of the output voltage. We therefore connect a controlled current source in parallel with the load resistor as shown below. This current source is controlled by the perturbation signal of the AC Sweep block. The block input is again connected to the load voltage signal. The average steady-state output current is 5 amperes; we therefore set the initial perturbation amplitude to $1e-2$.



Open-loop output impedance

Impulse Response Analysis

Alternatively you can determine the open-loop transfer functions using the Impulse Response Analysis block as shown in the figure below. In this analysis method the calculation of an individual output point is relatively inexpensive; we therefore set the number of points to 300 and extend the sweep range to [10 50000]. In order to compensate for the discrete rectangular pulse used to perturb the system, we choose the setting external reference for the control-to-output transfer function and discrete pulse for the output impedance.

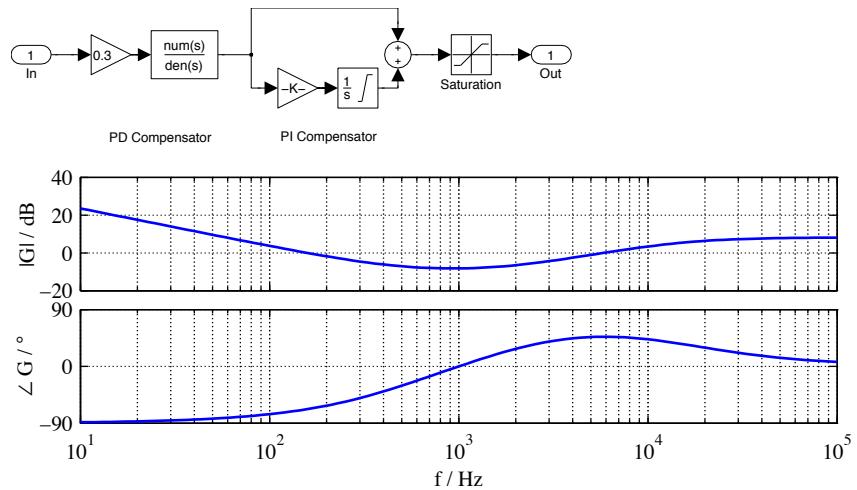


Using the Impulse Response Analysis block

Loop Gain Analysis

Compensator settings The compensator should attain a crossover frequency of $f_c = 5\text{kHz}$. At this frequency the open-loop control-to-output transfer function has a phase of nearly -180° . It should be lifted by 52° to get a peak overshoot of 16%. This is achieved using a PD compensator with a zero at $f_z = 1.7\text{kHz}$, a pole at $f_p = 14.5\text{kHz}$ and a dc gain of $k = (f_c/f_0)^2 \sqrt{f_z/f_p}/G_0 \approx 0.3$. For a zero stationary error a PI compensator with an inverted zero at $f_Z = 500\text{Hz}$ is added.

The compensator is implemented as shown above. The compensator output is limited to $0.1\dots 0.9$. In order to prevent windup problems during the steady-



PID compensator and transfer function

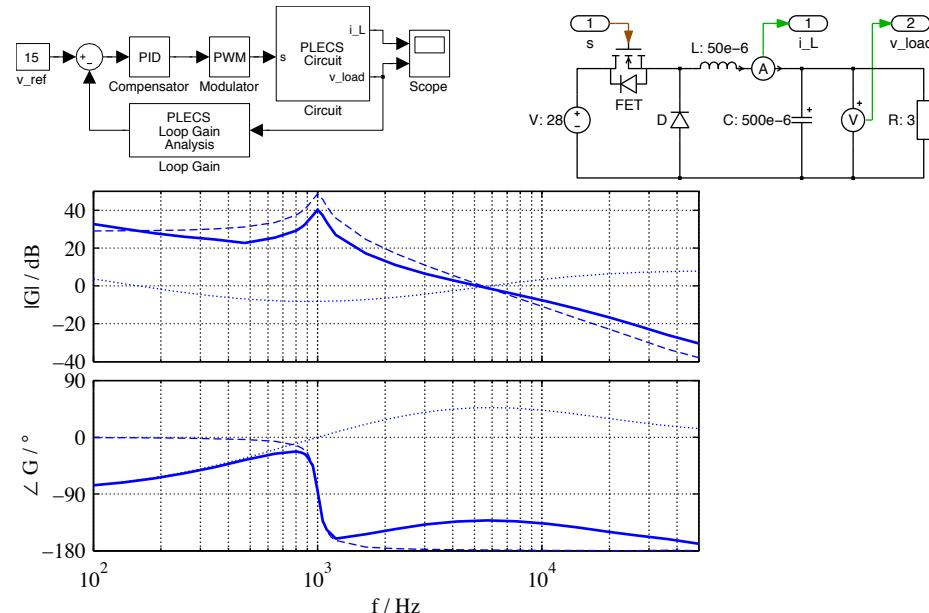
state analysis the integrator is limited to the same range.

Loop gain The gain of the closed control loop is measured by inserting the Loop Gain Analysis block into the loop path. A good place is the feedback path as shown below. The average steady-state load voltage is 15 volts; the initial perturbation amplitude is therefore chosen as $1e-2$. The convergence of the initial steady-state analysis can be accelerated by pre-charging the capacitor to its average steady-state voltage.

The resulting bode plot of the closed-loop gain shown in the figure below. Also shown are the open-loop control-to-output function with a dashed line and the PID compensator transfer function with a dotted line. As you can see, the design goals for crossover frequency and phase margin have been reached.

State-Space Averaging

Another method for obtaining the open-loop transfer functions of a circuit is a technique called state-space averaging. This topic is fairly complex and could



Closed-loop gain

easily fill a book of its own. This manual therefore assumes that you are familiar with the concept and just highlights how to use PLECS in the process. The code examples given here are collected in the demo M-file `p1SSADemo`.

The small-signal ac model of a dc converter operating in continuous conduction mode (CCM) is described by the equation system

$$\begin{aligned}\frac{d}{dt}\tilde{\mathbf{x}}(t) &= \bar{\mathbf{A}}\tilde{\mathbf{x}}(t) + \bar{\mathbf{B}}\tilde{\mathbf{u}}(t) + \{(\mathbf{A}_1 - \mathbf{A}_2)\bar{\mathbf{x}} + (\mathbf{B}_1 - \mathbf{B}_2)\bar{\mathbf{u}}\}\tilde{m}(t) \\ \tilde{\mathbf{y}}(t) &= \bar{\mathbf{C}}\tilde{\mathbf{x}}(t) + \bar{\mathbf{D}}\tilde{\mathbf{u}}(t) + \{(\mathbf{C}_1 - \mathbf{C}_2)\bar{\mathbf{x}} + (\mathbf{D}_1 - \mathbf{D}_2)\bar{\mathbf{u}}\}\tilde{m}(t)\end{aligned}$$

where the quantities $\tilde{\mathbf{x}}(t)$, $\tilde{\mathbf{u}}(t)$, $\tilde{\mathbf{y}}(t)$ and $\tilde{m}(t)$ are small ac variation around the operating point $\bar{\mathbf{x}}$, $\bar{\mathbf{u}}$, $\bar{\mathbf{y}}$ and \bar{m} . The averaged state-space matrices $\bar{\mathbf{A}}$, $\bar{\mathbf{B}}$, $\bar{\mathbf{C}}$ and $\bar{\mathbf{D}}$ are defined as

$$\begin{aligned}\bar{\mathbf{A}} &= \bar{m}\mathbf{A}_1 + (1 - \bar{m})\mathbf{A}_2 \\ \bar{\mathbf{B}} &= \bar{m}\mathbf{B}_1 + (1 - \bar{m})\mathbf{B}_2 \\ \bar{\mathbf{C}} &= \bar{m}\mathbf{C}_1 + (1 - \bar{m})\mathbf{C}_2 \\ \bar{\mathbf{D}} &= \bar{m}\mathbf{D}_1 + (1 - \bar{m})\mathbf{D}_2\end{aligned}$$

where the subscript 1 denotes the interval when the switch is conducting and the diode blocking, and the subscript 2 denotes the interval when the switch is blocking and the diode conducting.

You can use PLECS to calculate the different matrices \mathbf{A}_1 , \mathbf{A}_2 etc. and from these the various transfer functions. Using the buck converter from the previous example, the first step is to determine the internal order of the switches:

```
load_system('p1BuckSweep');
names = plecs('get', 'p1BuckSweep/Circuit', ...
    'StateSpaceOrder');
names.Switches

ans =
'Circuit/FET'
'Circuit/D'
```

Next you retrieve the state-space matrices for the two circuit topologies:

```

plecs('set', 'plBuckSweep/Circuit', 'SwitchVector', [1 0]);
t1 = plecs('get', 'plBuckSweep/Circuit', 'Topology');

plecs('set', 'plBuckSweep/Circuit', 'SwitchVector', [0 1]);
t2 = plecs('get', 'plBuckSweep/Circuit', 'Topology');

```

Now you can calculate the averaged state-space matrices:

```

m = 15/28;
A = t1.A*m + t2.A*(1-m);
B = t1.B*m + t2.B*(1-m);
C = t1.C*m + t2.C*(1-m);
D = t1.D*m + t2.D*(1-m);

```

Output impedance The output impedance is the transfer function from a state-space input (the current source I_{ac}) to a state-space output (the voltmeter V_m). Such a transfer function is given by:

$$\frac{\tilde{Y}(s)}{\tilde{U}(s)} = \bar{C}(sI - \bar{A})^{-1}\bar{B} + \bar{D}$$

Since the circuit model is a MIMO (multi-input multi-output) model, you need to specify the indices of the proper elements in the input and output vector. You can identify them using the fields `Inputs` and `Outputs` of the struct names that you retrieved earlier:

```

names.Inputs

ans =
'Circuit/V_dc'
'Circuit/I_ac'

names.Outputs

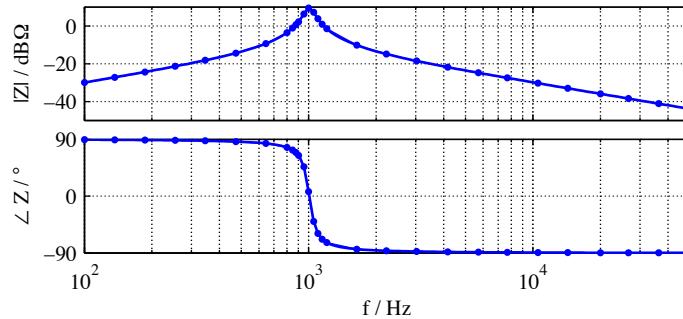
ans =
'Circuit/Vm'
'Circuit/Am'
'Circuit/FET'
'Circuit/FET'
'Circuit/D'
'Circuit/D'

```

So, the output impedance is the transfer function from input 2 to output 1. If you have the Control System Toolbox you can now display the Bode diagram:

```
bode(ss(A,B(:,2),C(1,:),D(1,2)), {2*pi*100, 2*pi*50000})
```

The figure below shows the output impedance drawn with a solid line. The dots represent the data points returned by the ac sweep.



Open-loop output impedance

Open-loop control-to-output transfer function The control-to-output transfer function describes the effect of the small ac variation \tilde{m} on the system outputs. From the small-signal ac model equations we find that

$$\frac{\tilde{\mathbf{Y}}(s)}{\tilde{\mathbf{M}}(s)} = \mathbf{C}_{co}(s\mathbf{I} - \mathbf{A}_{co})^{-1}\mathbf{B}_{co} + \mathbf{D}_{co}$$

with

$$\begin{aligned}\mathbf{A}_{co} &= \bar{\mathbf{A}} \\ \mathbf{B}_{co} &= \{ -(\mathbf{A}_1 - \mathbf{A}_2)\bar{\mathbf{A}}^{-1}\bar{\mathbf{B}} + (\mathbf{B}_1 - \mathbf{B}_2) \} \bar{\mathbf{u}} \\ \mathbf{C}_{co} &= \bar{\mathbf{C}} \\ \mathbf{D}_{co} &= \{ -(\mathbf{C}_1 - \mathbf{C}_2)\bar{\mathbf{A}}^{-1}\bar{\mathbf{B}} + (\mathbf{D}_1 - \mathbf{D}_2) \} \bar{\mathbf{u}}\end{aligned}$$

Note that \mathbf{B}_{co} and \mathbf{D}_{co} are column vectors since there is only one scalar input variable, \tilde{m} . The vector $\bar{\mathbf{u}}$ is a column vector consisting of the dc input voltage and the small-signal ac current.

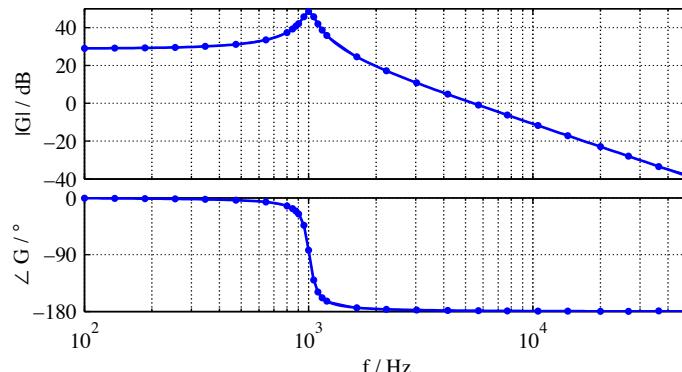
This leads to the following program code:

```
u = [28; 0];

B_co = (-(t1.A-t2.A)*(A\B)+(t1.B-t2.B))*u;
D_co = (-(t1.C-t2.C)*(A\B)+(t1.D-t2.D))*u;

bode(ss(A,B_co,C(1,:),D_co(1)), {2*pi*100, 2*pi*50000})
```

The figure below shows the control-to-output transfer function drawn with a solid line. The dots represent the data points returned by the ac sweep.



Open-loop control-to-output transfer function

Reference

R.W. Erickson, D. Maksimović, "Fundamentals of Power Electronics, 2nd Ed.", Kluwer Academic Publishers, 2003.

C-Scripts

C-Scripts provide a powerful and comfortable mechanism for implementing custom control blocks in the C programming language. They enable you to interact with the solver engine on a level very similar to that of built-in blocks.

Typical applications where C-Scripts are useful include:

- Implementing complex nonlinear and/or piecewise functions. These would otherwise need to be modeled with complex block diagrams that are hard to read and maintain.
- Implementing modulators or pulse generators that require exact but flexible time step control.
- Incorporating external C code, e.g. for a DSP controller, into a simulation model.

There is no need to manually compile any code or even to install a compiler. A built-in compiler translates your C code on-the-fly to native machine code and links it dynamically into PLECS.

A detailed description of how C-Scripts work is given in the following section. For a quick start you can also have a look at the C-Script examples further below.

How C-Scripts Work

Since C-Scripts interact so closely with the solver engine, a good understanding of how a dynamic system solver works is advantageous. This is described in detail in the chapter “How PLECS Works” (on page 25).

C-Script Functions

A C-Script block, like any other control block, can be described as a mathematical (sub-)system having a set of inputs u , outputs y and state variables x_c , x_d that are related to each other by a set of equations:

$$\begin{aligned}y &= f_{\text{output}}(t, u, x_c, x_d) \\x_d^{\text{next}} &= f_{\text{update}}(t, u, x_c, x_d) \\ \dot{x}_c &= f_{\text{derivative}}(t, u, x_c, x_d)\end{aligned}$$

A C-Script block has an individual code section for each of these functions and two additional sections for code to be executed at the start and termination of a simulation. The C code that you enter in these sections is automatically wrapped into C functions; the actual function interface is hidden to allow for future extensions. You can access block variables such as inputs, outputs and states by means of special macros that are described further below. The solver calls these C functions as required during the different stages of a simulation (see “Model Execution” on page 30).

Start Function

The start function is called at the beginning of a simulation. If the C-Script has continuous or discrete state variables they should be initialized here using the macros `ContState(i)` and `DiscState(i)`.

Output Function

The output function is called during major and minor time steps in order to update the output signals of the block. The block inputs and outputs and the current time can be accessed with the macros `Input(i)`, `Output(i)` and `CurrentTime`.

If you need to access any input signal during the output function call, you must check the **Input has direct feedthrough** box on the **Setup** pane of the C-Script dialog. This flag influences the block execution order and the occurrence of algebraic loops (see “Block Sorting” on page 29).

In general, output signals should be *continuous and smooth* during minor time steps; *discontinuities or sharp bends* should only occur during major time steps. Whether or not the call is made for a major time step can be inquired with the `IsMajorStep` macro. For details see “Modeling Discontinuities” below.

Note It is not safe to make any assumptions about the progression of time between calls to the output function. The output function may be called multiple times during the same major time step, and the time may jump back and forth between function calls during minor time steps. Code that should execute exactly *once* per major time step should be placed in the *update function*.

Update Function

If the block has discrete state variables, the update function is called once during a major time step after the output functions of all blocks have been processed. During this call, the discrete state variables should be updated using the `DiscState` macro.

Derivative Function

If the block has continuous state variables, the derivative function is called during the *integration loop* of the solver. During this call, the continuous state derivatives should be updated using the `ContDeriv` macro.

Derivatives should be *continuous and smooth* during minor time steps; *discontinuities or sharp bends* should only occur during major time steps. For details see “Modeling Discontinuities” below.

Terminate Function

The terminate function is called at the end of a simulation – regardless of whether the simulation stop time has been reached, the simulation has been stopped interactively, or an error has occurred. Use this function to free any resources that you may have allocated during the start function (e.g. file handles, memory etc.).

Code Declarations

This code section is used for global declarations and definitions (that is, global in the scope of the C-Script block). This is the place to include standard library headers (e.g. `math.h` or `stdio.h`) and to define macros, static variables and helper functions that you want to use in the C-Script functions.

You can also include external source files. The directory containing the model file is automatically added to the included search path, so you can specify the source file path relative to the model file.

Modeling Discontinuities

If the behavior of your C-Script block changes abruptly at certain instants, you must observe the following two rules in order to obtain accurate results:

- 1 If the time, at which a discontinuity or event occurs, is not known a priori but depends on the block inputs and/or states, you must define one or more zero-crossing signals, which aid the solver in locating the event. Failure to do so may result in a jitter on the event times.
- 2 During minor time steps, continuous state derivatives and output signals must be *continuous and smooth* functions. Failure to observe this may lead to gross numerical integration errors.

Defining Zero-crossing Functions

To define zero-crossing signals, register the required number of signals on the **Setup** pane of the C-Script dialog. In the output function, use the macro `ZCSignal(i)` to assign values to the individual zero-crossing signals depending e.g. on the block inputs or states or the current simulation time. The solver constantly monitors all zero-crossing signals of all blocks. If any one signal changes its sign during the current integration step, the step size is reduced so that the next major time step occurs *just after* the first zero-crossing. (See also “Event Detection Loop” on page 31.)

For instance, to model a comparator that must change its output when the input crosses a threshold of 1, you should define the following zero-crossing signal:

```
ZCSignal(0) = Input(0) - 1.;
```

Without the aid of the zero-crossing signal, the solver might make one step at a time when the input signal is e.g. 0.9 and the next step when the input signal has already increased to e.g. 1.23, so that the C-Script block would change its output too late.

With the zero-crossing signal, and provided that the input signal is continuous, the solver will be able to adjust the step size so that the C-Script output will change at the correct time.

Note If a zero-crossing signal depends solely on the simulation time, i.e. if an event time *is* known *a priori*, it is recommended to use a *discrete-variable sample time* and the `NextSampleHit` macro instead. (See “Discrete-Variable Sample Time” below.)

Keeping Functions Continuous During Minor Time Steps

The solver integrates the continuous state derivatives over a given interval (i.e. the current time step) by evaluating the derivatives at different times in the interval. It then fits a polynomial of a certain order to approximate the integral. (See also “Integration Loop” on page 31.) The standard Dormand-Prince solver, for instance, uses 6 derivative evaluations and approximates the integral with a polynomial of 5th order.

Obviously, the derivative of this polynomial is again a polynomial of one order less. On the other hand, to approximate a discontinuous or even just a non-smooth derivative function, a polynomial of infinite order would be required. This discrepancy may lead to huge truncation errors. It is therefore vital to describe the continuous state derivatives as *piecewise smooth* functions and make sure that only one subdomain of these functions is active throughout one integration step.

The output signal of a C-Script block might be used as the input signal of an integrator and thus might become the derivative of a continuous state variable. Therefore, output signals should be described as piecewise smooth functions as well.

Returning to the example of the comparator above, the complete output function code should look like this:

```
if (IsMajorStep)
```

```
{  
    if (Input(0) >= 1.)  
        Output(0) = 1.;  
    else  
        Output(0) = 0.;  
}  
  
ZCSignal(0) = Input(0) - 1.;
```

The condition `if (IsMajorStep)` ensures that the output signal can only change in major steps. It remains constant during the integration loop regardless of the values that the input signal assumes during these minor time steps. The zero-crossing signal, however, is also updated in minor time steps during the event detection loop of the solver.

Sample Time

A C-Script block can model a continuous system, a discrete system, or even a hybrid system having both continuous and discrete properties. Depending on which kind of system you want to model, you need to specify an appropriate **Sample time** on the **Setup** pane of the C-Script dialog. The sample time determines at which time steps (and at which stages) the solver calls the different C-Script functions.

Continuous Sample Time

Blocks with a continuous sample time (setting 0 or [0, 0]) are executed at every major and minor time step. You must choose a continuous sample time if

- the C-Script models a continuous (or piecewise continuous) function,
- the C-Script has continuous states or,
- the C-Script registers one or more zero-crossing signals for event detection.

Semi-Continuous Sample Time

Blocks with a semi-continuous sample time (setting [0, -1]) are executed at every major time step but not at minor time steps. You can choose a semi-continuous instead of a continuous sample time if the C-Script produces only discrete output values and does *not* need zero-crossing signals.

Discrete-Periodic Sample Time

Blocks with a discrete-periodic sample time (setting T_p or $[T_p, T_o]$) are executed at regularly spaced major time steps. The sample period T_p must be a positive real number. The sample offset T_o must be a positive real number in the interval $0 \leq T_o < T_p$; it may be omitted if it is zero.

The time steps, at which the output and update functions are executed, are calculated as $n \cdot T_p + T_o$ with an integer n .

Discrete-Variable Sample Time

Blocks with a discrete-variable sample time (setting -2 or [-2, 0]) are executed at major time steps that are specified by the blocks themselves.

In a C-Script you assign the time, when the block should be executed next, to the macro `NextSampleHit`. This can be done either in the output or update function. At the latest, after the update function call, the `NextSampleHit` must be greater than the current simulation time. Otherwise, the simulation will be aborted with an error.

If a C-Script *only* has a discrete-variable sample time, the time of the first sample hit must be assigned in the start function. Otherwise, the C-Script will never be executed. During the start function, the simulation start time is available via the macro `CurrentTime`.

Note For discrete-variable sample times, the PLECS Blockset can control the time steps taken by the Simulink solvers only indirectly by using an internal zero-crossing signal. Therfore, the actual simulation time at a discrete-variable sample hit may be slightly larger than the value that was specified as the next sample hit.

The solvers of PLECS Standalone, however, can evaluate the sample hit requests directly and are therefore guaranteed to meet the requests exactly.

Multiple Sample Times

If you want to model a hybrid system you can specify multiple sample times in different rows of an $n \times 2$ matrix. For example, if your C-Script has continuous

states but you must also ensure that it is executed every 0.5 seconds with an offset of 0.1 seconds, you would enter [0, 0; 0.5, 0.1].

You can use the macro `IsSampleHit(i)` in the output and update functions in order to inquire which of the registered sample times has a hit in the current time step. The index `i` is a zero-based row number in the sample time matrix. In the above example, if your C-Script should perform certain actions only at the regular sampling intervals, you would write

```
if (IsSampleHit(1))
{
    // this code is only executed at t == n*0.5 + 0.1
}
```

User Parameters

If you want to implement generic C-Scripts that can be used in different contexts, you can pass external parameters into the C functions.

External parameters are entered as a comma-separated list in the **Parameters** field on the **Setup** pane of the C-Script dialog. The individual parameters can be specified as MATLAB expressions and can reference workspace variables. They must evaluate to real scalars, vectors, matrices or 3d-arrays.

Within the C functions you can inquire the number of external parameters with the macro `NumParameters`. The macros `ParamNumDims(i)` and `ParamDim(i, j)` return the number of dimensions of the individual parameters and their sizes.

To access the actual parameter values, use the macro `ParamRealValue(i, j)`, where `j` is a *linear index* into the data array. For example, to access the value in a certain row, column and page of a 3d-array, you write:

```
int rowIdx = 2;
int colIdx = 0;
int pageIdx = 1;
int numRows = ParamDim(0, 0);
int numCols = ParamDim(0, 1);
int elIdx = rowIdx + numRows*(colIdx + numCols*pageIdx);
double value = ParamRealData(0, elIdx);
```

Runtime Checks

If the box **Enable runtime checks** on the **Setup** pane of the C-Script dialog is checked, C-Script macros that access block data (e.g. signals values, states, parameters etc.) are wrapped with protective code to check whether an array index is out of range. Also, the C-Script function calls are wrapped with code to check for solver policy violations such as modifying states during minor time steps or accessing input signals in the output function without enabling direct feedthrough.

These runtime checks have a certain overhead, so once you are sure that your C-Script is free of errors you can disable them in order to increase the simulation speed. This is not recommended, however, because in this case access violations in your C-Script may cause PLECS to crash.

Note The runtime checks cannot guard you against access violations caused by direct memory access.

C-Script Examples

This section presents a collection of simple examples that demonstrate the different features of the C-Script and that you can use as starting points for your own projects. Note that the functionality of the example blocks (with the exception of the Wrapping Integrator) is already available from blocks in the PLECS library.

A Simple Function – Times Two

The first example implements a block that simply multiplies a signal with 2. This block is described by the following system equation:

$$y = f_{\text{output}}(t, u, x_c, x_d) = 2 \cdot u$$

Block Setup The block has one input, one output, no states and no zero-crossing signals. It has direct feedthrough because the output function depends on the current input value. Since the output signal is continuous (provided that the input signal is) the sample time is also continuous, i.e. [0, 0] or simply 0.

Output Function Code

```
Output(0) = 2.*Input(0);
```

In every major and minor time step, the output function retrieves the current input value, multiplies it with 2 and assigns the result to the output.

Discrete States – Sampled Delay

This example implements a block that samples the input signals regularly with a period of one second and outputs the samples with a delay of one period. Such a block is described by the following set of system equations:

$$\begin{aligned}y &= f_{\text{output}}(t, u, x_c, x_d) = x_d \\x_d^{\text{next}} &= f_{\text{update}}(t, u, x_c, x_d) = u\end{aligned}$$

Remember that in a major time step the solver first calls the block output function and then the block update function.

Block Setup The block has one input and one output. One discrete state variable is used to store the samples. The block does not have direct feedthrough because the input signal is not used in the output function but only in the update function. The sample time is [1, 0] or simply 1.

Output Function Code

```
Output(0) = DiscState(0);
```

Update Function Code

```
DiscState(0) = Input(0);
```

Continuous States – Integrator

This example implements a block that continuously integrates the input signal and outputs the value of the integral. Such a block is described by the following set of system equations:

$$\begin{aligned}y &= f_{\text{output}}(t, u, x_c, x_d) = x_c \\ \dot{x}_c &= f_{\text{derivative}}(t, u, x_c, x_d) = u\end{aligned}$$

Block Setup The block has one input and one output. One continuous state variable is used to integrate the input signal. The block does not have direct feedthrough because the input signal is not used in the output function but only in the derivative function. The sample time is continuous, i.e. [0, 0] or simply 0.

Output Function Code

```
Output(0) = ContState(0);
```

Derivative Function Code

```
ContDeriv(0) = Input(0);
```

Event Handling – Wrapping Integrator

This examples extends the previous one by implementing an integrator that wraps around when it reaches an upper or lower boundary (e.g. 2π and 0). Such an integrator is useful for building e.g. a PLL to avoid round-off errors that would occur if the phase angle increased indefinitely. This wrapping property can actually not be easily described with mathematical functions. However, the C code turns out to be fairly simple.

Block Setup The block has the same settings as in the previous example. Additionally, it requires two zero-crossing signals, in order to let the solver find the exact instants, at which the integrator state reaches the upper or lower boundary.

Output Function Code

```
#define PI 3.141592653589793
if (IsMajorStep)
{
    while (ContState(0) > 2*PI)
        ContState(0) -= 2*PI;
    while (ContState(0) < 0)
        ContState(0) += 2*PI;
}
ZCSignal(0) = ContState(0);
ZCSignal(1) = ContState(0) - 2*PI;

Output(0) = ContState(0);
```

In every major time step, if the integrator state has gone beyond the upper or lower boundary, 2π is added to or subtracted from the state until it lies within the boundaries again. In every major and minor time step, the zero-crossing signals are calculated so that they become zero when the state is 0 resp. 2π . Finally, the integrator state is assigned to the output.

Note, that the state may *not* be modified during minor time steps, because then the solver is either itself updating the state (while integrating it) or trying to find the zeros of the zero-crossing functions, which in turn depend on the state. In either case an external modification of the state will lead to unpredictable results.

Derivative Function Code

```
ContDeriv(0) = Input(0);
```

Piecewise Smooth Functions – Saturation

This example implements a saturation block that is described by the following piecewise system equation:

$$y = f_{\text{output}}(t, u, x_c, x_d) = \begin{cases} 1, & \text{for } u \geq 1 \\ u, & \text{for } -1 < u < 1 \\ -1, & \text{for } 1 \leq u \end{cases}$$

When implementing this function, care must be taken to ensure that the active output equation does not change during an integration loop in order to avoid numerical errors (see “Modeling Discontinuities” on page 136).

Block Setup The block has one input, one output and no state variables. In order to make sure that a major step occurs whenever the input signal crosses the upper or lower limit, two zero-crossing signals are required.

Output Function Code

```
static enum { NO_LIMIT, LOWER_LIMIT, UPPER_LIMIT } mode;

if (IsMajorStep)
{
    if (Input(0) > 1.)
        mode = UPPER_LIMIT;
    else if (Input(0) < -1.)
        mode = LOWER_LIMIT;
```

```

        else
            mode = NO_LIMIT;
    }

switch (mode)
{
    case NO_LIMIT:
        Output(0) = Input(0);
        break;
    case UPPER_LIMIT:
        Output(0) = 1.;
        break;
    case LOWER_LIMIT:
        Output(0) = -1.;
        break;
}
ZCSignal(0) = Input(0) + 1.;
ZCSignal(1) = Input(0) - 1.;

```

Ensuring that only one output equation will be used throughout an entire integration step requires a static mode variable that will retain its value between function calls. The active mode is determined in major time steps depending on the input signal. In the subsequent minor time steps, the equation indicated by the mode variable will be used *regardless of the input signal*.

If the step size were not properly limited and the input signal went beyond the limits during minor time steps, so would the output signal. This is prevented by the two zero-crossing signals that enable the solver to reduce the step size as soon as the input signal crosses either limit.

Note Instead of the static mode variable, a discrete state variable could also be used to control the active equation. In this particular application a static variable is sufficient because information needs to be passed only from one major time step to the subsequent *minor* time steps.

However, if information is to be passed from one major time step to a later *major* time step, a discrete state variable should be used, so that it can also be stored between multiple simulation runs.

Multiple Sample Times – Turn-on Delay

A turn-on delay is often needed for inverter controls in order to prevent short-circuits during commutation. When the input signal changes from 0 to 1, the output signal will follow after a prescribed delay time, provided that the input signal is still 1 at that time. When the input signal changes to 0, the output is reset immediately.

Block Setup The block has one input and one output. One discrete state variable is required to store the input signal value from the previous major time step.

Two sample times are needed: a semi-continuous sample time so that the input signal will be sampled at *every major time step*, and a discrete-variable sample time to enforce a major time step *exactly* after the prescribed delay time. The **Sample time** parameter is therefore set to [0, -1; -2, 0].

As an additional feature the delay time is defined as an external user parameter.

Code Declarations

```
#include <float.h>
#define PREV_INPUT DiscState(0)
#define DELAY ParamRealData(0, 0)
```

The standard header file `float.h` defines two numerical constants, `DBL_MAX` and `DBL_EPSILON`, that will be needed in the output function. Additionally, two convenience macros are defined in order to make the following code more readable.

Start Function Code

```
if (NumParameters != 1)
{
    SetErrorMessage("One parameter required (delay time).");
    return;
}
if (ParamNumDims(0) != 2
    || ParamDim(0, 0) != 1 || ParamDim(0, 1) != 1
    || DELAY <= 0.)
{
    SetErrorMessage("Delay time must be a positive scalar.");
    return;
}
```

The start function checks whether the proper number of external parameters (i.e. one) has been provided, and whether this parameter has the proper dimensions and value.

Output Function Code

```

if (Input(0) == 0)
{
    Output(0) = 0;
    NextSampleHit = DBL_MAX;
}
else if (PREV_INPUT == 0)
{
    NextSampleHit = CurrentTime + DELAY;
    if (NextSampleHit == CurrentTime)
        NextSampleHit = CurrentTime * (1.+DBL_EPSILON);
}
else if (IsSampleHit(1))
{
    Output(0) = 1;
    NextSampleHit = DBL_MAX;
}

```

If the input signal is 0, the output signal is also set to 0 according to the block specifications. The next discrete-variable hit is set to some large number (in fact: the largest possible floating point number) because it is not needed in this case.

Otherwise, if the input signal is *not* 0 but it has been in the previous time step, i.e. if it just changed from 0 to 1, a discrete-variable sample hit is requested at *DELAY* seconds later than the current time.

Finally, if *both* the current and previous input signal values are nonzero and the discrete-variable sample time has been hit, i.e. if the delay time has just passed and the current input is still nonzero, the output is set to 1 and the next discrete-variable hit time is again reset to the largest possible floating point number.

The condition `if (NextSampleHit == CurrentTime)` requires special explanation: If *DELAY* is very small and the current time is very large, the sum of these two floating point numbers might again yield the current time value due to roundoff errors, which would lead to a simulation error. In this case the next sample hit is increased to the smallest possible floating point number that is still larger than the current time. Admittedly, this problem will only occur when the current time and the delay time are more than *15 decades* apart, and so it might be considered academic.

Update Function Code

```
PREV_INPUT = Input(0);
```

In the update function, the current input value is stored as the previous input value for the following time step.

C-Script Macros

The following table summarizes the macros that can be used in the C-Script function code sections.

C-Script Data Access Macros

Macro	Type	Access	Description
NumInputs	int	R	Returns the number of input signals.
NumOutputs	int	R	Returns the number of output signals.
NumContStates	int	R	Returns the number of continuous states.
NumDiscStates	int	R	Returns the number of discrete states.
NumZCSignals	int	R	Returns the number of zero-crossing signals.
NumParameters	int	R	Returns the number of user parameters.
CurrentTime	double	R	Returns the current simulation time (resp. the simulation start time during the start function call).
IsMajorStep	int	R	Returns 1 during major time steps, else 0.
IsSampleHit(int i)	int	R	Returns 1 if the ith sample time currently has a hit, else 0.
NextSampleHit	double	R/W	Specifies the next simulation time when the block should be executed. This is relevant only for blocks that have registered a discrete-variable sample time.
Input(int i)	double	R	Returns the value of the ith input signal.

C-Script Data Access Macros (contd.)

Macro	Type	Access	Description
Output(int i)	double	R/W	Provides access to the value of the ith output signal. Output signals may <i>only</i> be changed during the output function call.
ContState(int i)	double	R/W	Provides access to the value of the ith continuous state. Continuous state variables may <i>not</i> be changed during minor time steps.
ContDeriv(int i)	double	R/W	Provides access to the derivative of the ith continuous state.
DiscState(int i)	double	R/W	Provides access to the value of the ith discrete state. Discrete state variables may <i>not</i> be changed during minor time steps.
ZCSignal(int i)	double	R/W	Provides access to the ith zero-crossing signal.
ParamNumDims(int i)	int	R	Returns the number of dimensions of the ith user parameter.
ParamDim(int i, j)	int	R	Returns the jth dimension of the ith user parameter.
ParamRealData(int i, j)	double	R	Returns the value of the jth element of the ith user parameter. The index j is a linear index into the parameter elements. Indices into multi-dimensional arrays must be calculated using the information provided by the ParamNumDims and ParamDim macros.
SetErrorMessage(char *msg)	void	W	Use this macro to report errors that occur in your code. The simulation will be terminated as soon as the current C-Script function returns. In general, this macro should be followed by a return statement. The pointer msg must point to static memory.

Note The values of the input and output signals are not stored in contiguous memory. Therefore, signal values may only be accessed by using the macros, not by pointer arithmetic. For example, trying to access the second output using the following code will fail:

```
double *output = &Output(0); // not recommended
output[1] = 1;           // fails
*(output + 1) = 1;      // fails
Output(1) = 1;          // ok
```

Simulation Scripts

Running simulations from a script allows you to examine the effect of varying parameters or to post-process the simulation results to extract relevant information.

In PLECS Blockset scripts are written in the Matlab environment. Simulink offers a scripting interface to modify parameters and run simulations from a script. A detailed description of the Simulink scripting options is out of the scope of this manual, please refer to the documentation for Simulink instead. PLECS Blockset offers additional commands to control the parameters of PLECS circuits.

PLECS Standalone offers two different scripting methods:

- Scripts can be executed directly in PLECS Standalone. The scripts use a syntax which is very similar to Matlab.
- PLECS offers an XML-RPC interface that allows any other program that can send XML-RPC requests to control PLECS. Many scripting languages support XML-RPC out of the box, for example Python or Ruby. Other scripting language extensions for XML-RPC support are available for free on the internet.

The scripting options for PLECS Standalone are described in section “Simulation Scripts in PLECS Standalone” (on page 155).

Command Line Interface in PLECS Blockset

PLECS offers a Command Line Interface (CLI) to access component and circuit parameters from scripts or, in case of PLECS Blockset, also directly from the MATLAB command line. The command syntax is

```
plecs('cmd', 'parameter1', 'parameter2', ...)
```

where *cmd* is one of the following commands: get, set, scope, thermal, export, version, hostid.

Reading and Setting Parameters of Components

The command

```
plecs('get', 'componentPath')
plecs('get', 'componentPath', 'parameter')
```

returns the value of *parameter* of the PLECS component indicated by the *componentPath* as a string. If *parameter* is omitted a cell array with all available parameters is returned.

```
plecs('set', 'componentPath', 'parameter', 'value')
```

sets the value of *parameter* of the PLECS component indicated by the *componentPath* to *value*.

The special parameter 'CurrentCircuit' can be used to query the path to the current PLECS Circuit. The component path has to be an empty string:

```
plecs('get', '', 'CurrentCircuit')
```

This command can only be used in the initialization commands of subsystems.

Holding and Clearing Traces in Scopes

```
plecs('scope', 'scopePath', 'HoldTrace')
plecs('scope', 'scopePath', 'HoldTrace', 'traceName')
```

saves the values of the last simulation run to a new trace in the scope indicated by the *scopePath*. If given, *traceName* is used as the name for the new trace, otherwise a default name is assigned.

```
plecs('scope', 'scopePath', 'ClearTraces')
```

clears all traces in the scope indicated by the *scopePath*.

Converting Thermal Descriptions

```
plecs('thermal', 'import', valVon, valEon, valEoff)
plecs('thermal', 'import', valVon, valEon, valEoff, ...
      valCauer)
```

imports the on-state voltage drop *valVon*, the switching losses *valEon*, *valEoff* and the cauer chain elements *valCauer* into the thermal editor. The parameter *valVon* has to be a struct with two index vectors *i*, *T* and an output matrix *v*. The parameters *valEon* and *valEoff* have to be structs with three index vectors *v*, *i*, *T* and an output array *E*. The optional argument *valCauer* has to be a struct with two array elements, *C* and *R*. The command can be used to import thermal descriptions as used in PLECS 1.x into the thermal library of PLECS 2.x and later.

```
plecs('thermal', 'export', 'filename')
```

reads the thermal data sheet from *filename* and returns a struct with the fields Von, Eon, Eoff, CauerChain and Comment containing the respective data from the thermal data sheet. The parameter *filename* has to be an absolute filename to the data sheet including the .xml extension.

```
plecs('thermal', 'export', 'filename', 'modelName')
```

reads the thermal data sheet from *filename* and returns a struct with the fields Von, Eon, Eoff, CauerChain and Comment containing the respective data from the thermal data sheet. The parameter *filename* has to be a relative filename to the data sheet without the .xml extension.

Export for PLECS Viewer

```
plecs('export', 'modelName')
plecs('export', 'modelName', hideSchematics)
plecs('export', 'modelName', hideSchematics, 'filename')
```

exports the model *modelName* for the PLECS Viewer. By setting the optional argument *hideSchematics* to true all PLECS circuits are marked as non-viewable. If argument *filename* is given it is used as the exported model's filename, otherwise you are prompted for a filename.

Other CLI Commands

To retrieve the version information from PLECS as a string, enter

```
plecs('version')
```

To retrieve a struct with host ID and MATLAB license information, enter

```
plecs('hostid')
```

To check out a license for PLECS, enter

```
[success,message] = plecs('checkout')
```

If the check-out succeeds, the return variable `success` will be set to 1 and `message` will be an empty string. Else, `success` will be set to 0 and `message` will contain a detailed error message. When called without left-hand side arguments, the command will raise a MATLAB error upon an unsuccessful check-out and else execute silently.

Examples

Some examples for using the command line interface in PLECS Blockset:

```
plecs('get', 'mdl/Circuit1')
```

returns the parameters of `Circuit1` in the simulink model `mdl`.

```
plecs('get', 'mdl/Circuit1', 'Name')
```

returns the name of `Circuit1`.

```
plecs('get', 'mdl/Circuit1', 'CircuitModel')
```

returns the circuit simulation method of `Circuit1`.

```
plecs('get', 'mdl/Circuit1/R1')
```

returns the parameters of component `R1` in circuit `Circuit1`.

```
plecs('set', 'mdl/Circuit1/R1', 'R', '2')
```

sets the resistance of component R1 in circuit Circuit1 to 2.

```
plecs('export', 'mdl', true, 'exported.mdl')
```

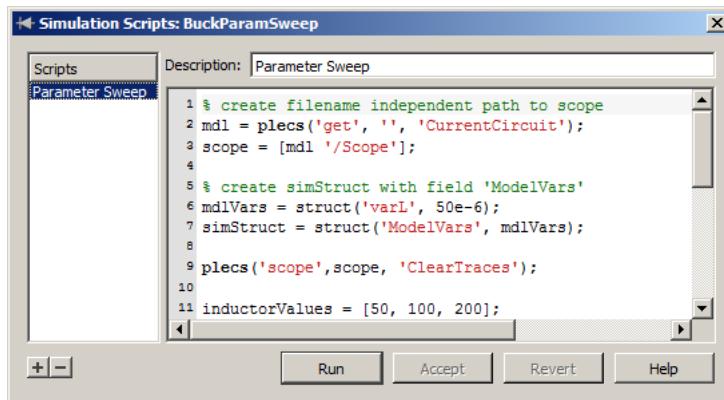
exports the model `mdl` to the model file `exported.mdl` which can be opened with the PLECS Viewer. The contents of all PLECS schematics are hidden.

```
data = plecs('thermal', 'export', 'Infineon/SDP04S60', ...
    'p1SMPS_CCM')
```

assigns the thermal description from library element Infineon/SDP04S60 in model `p1SMPS_CCM` to the variable `data` in the MATLAB workspace.

Simulation Scripts in PLECS Standalone

Simulation scripts are managed in the Simulation Scripts dialog shown below. To open the dialog, select **Simulation scripts...** from the **Simulation** menu of the schematic editor.



The left hand side of the dialog window shows a list of the scripts that are currently configured for the model. To add a new script, click the button marked **+** below the list. To remove the currently selected script, click on the button marked **-**. You can reorder the scripts by clicking and dragging an entry up and down in the list.

The right hand side of the dialog window shows the script in an editor window. Each script must have a unique **Description**.

The button **Run/Stop** starts the currently selected script or aborts the script that is currently running.

To make changes to the script without running it, press the **Accept** button. The **Revert** button takes back any changes that have been made after the **Accept** or **Run** button was pressed.

PLECS Standalone uses GNU Octave to execute simulation scripts. The Octave language is very similar to Matlab. A full syntax description of the Octave scripting language is available in the Octave documentation, <http://www.gnu.org/software/octave/doc/interpreter/>.

Overview of PLECS Scripting Extensions

In addition to generic Octave commands you can use the following commands to control PLECS from within a simulation script.

Reading and Setting Component Parameters

The command

```
plecs('get', 'componentPath')
plecs('get', 'componentPath', 'parameter')
```

returns the value of *parameter* of the PLECS component indicated by the *componentPath* as a string. If *parameter* is omitted a struct array with all available parameters is returned.

```
plecs('set', 'componentPath', 'parameter', 'value')
```

sets the value of *parameter* of the PLECS component indicated by the *componentPath* to *value*.

The special parameter 'CurrentCircuit' can be used to query the name of the model that contains the script which is currently executed. The component path has to be an empty string:

```
plecs('get', '', 'CurrentCircuit')
```

This command is useful for constructing a component path that does not depend on the model name.

Holding and Clearing Traces in Scopes

```
plecs('scope', 'scopePath', 'HoldTrace')
plecs('scope', 'scopePath', 'HoldTrace', 'traceName')
```

saves the values of the last simulation to a new trace in the scope indicated by the *scopePath*. If given, *traceName* is used as the name for the new trace, otherwise a default name is assigned.

```
plecs('scope', 'scopePath', 'ClearTraces')
```

clears all traces in the scope indicated by the *scopePath*.

Starting a Simulation

```
plecs('simulate')
plecs('simulate', optStruct)
```

starts a simulation. The optional argument *optStruct* can be used to override model parameters; for detailed information see section “Scripted Simulation and Analysis Options” (on page 162).

If any outports exist on the top level of the simulated model, the command returns a struct consisting of two fields, Time and Values. Time is a vector that contains the simulation time for each simulation step. The rows of the array Values consist of the signal values at the outports. The order of the signals is determined by the port numbers.

Starting an Analysis

```
plecs('analyze', 'analysisName')
plecs('analyze', 'analysisName', optStruct)
```

starts the analysis defined in the Analysis Tools dialog under the name *analysisName*. The optional argument *optStruct* can be used to override model parameters; for detailed information see section “Scripted Simulation and Analysis Options” (on page 162).

For a Steady-State Analysis, if any outports exist on the top level of the simulated model, the command returns a struct consisting of two fields, Time and

Values as described above. The signal values at the outports are captured after a steady-state operating point has been obtained.

For an AC Sweep or an Impulse Response Analysis, the command returns a struct consisting of three fields, F, Gr and Gi. F is a vector that contains the perturbation frequencies of the analysis. The rows of the arrays Gr and Gi consist of the real and imaginary part of the transfer function as defined in the analysis.

Example Script

The following script runs a parameter sweep by setting the variable varL to the values in inductorValues. It is used in the demo model BuckParamSweep.

```
mdl = plecs('get', '', 'CurrentCircuit');
scope = [mdl '/Scope'];

mdlVars = struct('varL', 50e-6);
opts = struct('ModelVars', mdlVars);

plecs('scope', scope, 'ClearTraces');

inductorValues = [50, 100, 200];
for ix = 1:length(inductorValues)
    opts.ModelVars.varL=inductorValues(ix) * 1e-6;
    out = plecs('simulate', opts);
    plecs('scope', scope, 'HoldTrace', ...
        ['L=' mat2str(inductorValues(ix)) 'uH']);
    [maxv, maxidx] = max(out.Values(1,:));
    printf('Max current for L=%duH: %f at %fs\n', ...
        inductorValues(ix), maxv, out.Time(maxidx));
end
```

The first two lines construct the path to the component *Scope*. By using the *CurrentCircuit* to build the path the script does not depend on a specific model name.

The next two lines create a struct *ModelVars* with one field, *varL*. The struct is embedded into another struct named *opts*, which will be used later to initialize the simulation parameters.

Inside of the for-loop each value of *inductorValues* is assigned successively to the structure member variable *varL*. A new simulation is started, the result is

saved in variable `out` for post-processing. By holding the trace in the scope the scope output will remain visible when a new simulation is started. The name of the trace is the inductance value.

The script then searches for the peak current in the simulation results and outputs the value and the time, at which it occurred, in the Octave Console.

XML-RPC Interface in PLECS Standalone

The XML-RPC interface allows you to control PLECS Standalone from an external program. PLECS acts as an XML-RPC server which processes requests from an XML-RPC client.

XML-RPC is a lightweight protocol that is supported by numerous scripting languages. For the following description, Python 2.x syntax and script excerpts are used.

Establishing an XML-RPC Connection to PLECS

The XML-RPC interface in PLECS is disabled by default. It must be enabled in the PLECS preferences before a connection can be established. The TCP port to use can also be configured in the PLECS preferences.

The following Python code initiates an XML-RPC connection to PLECS:

```
import xmlrpclib
server = xmlrpclib.Server("http://localhost:1080/RPC2")
```

The code assumes that PLECS is configured to use TCP port 1080 for XML-RPC. Note that the URL must end with "/RPC2", which is an XML-RPC convention.

Note XML-RPC connections to PLECS are only allowed from clients running on the same machine as PLECS. Therefore, the connection should always be initiated using `localhost` in the server URL.

Overview of XML-RPC Commands

Commands for PLECS start with `plecs` followed by a dot. In Python they are invoked on the `server` object, for example

```
server.plecs.load("myModel.plecs")
```

Opening and Closing a Model

The command

```
plecs.load('mdlFileName')
```

opens the model with the given `mdlFileName`. The filename should contain the absolute path to the file.

The command

```
plecs.close('mdlName')
```

closes the model with the given name. The model will be closed unconditionally without being saved, even when changes have been made.

Reading and Setting Component Parameters

The command

```
plecs.get('componentPath')
plecs.get('componentPath', 'parameter')
```

returns the value of `parameter` of the PLECS component indicated by the `componentPath` as a string. If `parameter` is omitted a struct array with all available parameters is returned.

```
plecs.set('componentPath', 'parameter', 'value')
```

sets the value of `parameter` of the PLECS component indicated by the `componentPath` to `value`.

Holding and Clearing Traces in Scopes

```
plecs.scope('scopePath', 'HoldTrace')
plecs.scope('scopePath', 'HoldTrace', 'traceName')
```

saves the values of the last simulation to a new trace in the scope indicated by the *scopePath*. If given, *traceName* is used as the name for the new trace, otherwise a default name is assigned.

```
plecs.scope('scopePath', 'ClearTraces')
```

clears all traces in the scope indicated by the *scopePath*.

Starting a Simulation

The command

```
plecs.simulate('mdlName')
plecs.simulate('mdlName', optStruct)
```

starts a simulation of the model named *mdlName*. The optional argument *optStruct* can be used to override model parameters; for detailed information see section “Scripted Simulation and Analysis Options” (on page 162).

If any outports exist on the top level of the simulated model, the command returns a struct consisting of two fields, *Time* and *Values*. *Time* is a vector that contains the simulation time for each simulation step. The rows of the array *Values* consist of the signal values at the outports. The order of the signals is determined by the port numbers.

Starting an Analysis

The command

```
plecs.analyze('mdlName', 'analysisName')
plecs.analyze('mdlName', 'analysisName', optStruct)
```

starts the analysis named *analysisName* in the model named *mdlName*. The optional argument *optStruct* can be used to override model parameters; for detailed information see section “Scripted Simulation and Analysis Options” (on page 162).

For a Steady-State Analysis, if any outports exist on the top level of the simulated model, the command returns a struct consisting of two fields, Time and Values as described above. The signal values at the outports are captured after a steady-state operating point has been obtained.

For an AC Sweep or an Impulse Response Analysis, the command returns a struct consisting of three fields, F, Gr and Gi. F is a vector that contains the perturbation frequencies of the analysis. The rows of the arrays Gr and Gi consist of the real and imaginary part of the transfer function as defined in the analysis.

Example Script

The following Python script establishes an XML-RPC connection, loads a model and simulates it twice. The scope output from each simulation is preserved by holding the traces in the scope.

```
import xmlrpclib
server = xmlrpclib.Server("http://localhost:1080/RPC2")

server.plecs.load("C:/Models/BuckParamSweep.plecs")
server.plecs.scope('BuckParamSweep/Scope', 'ClearTraces')

opts = {'ModelVars' : { 'varL' : 50e-6 } }
result = server.plecs.simulate("BuckParamSweep", opts)
server.plecs.scope('BuckParamSweep/Scope',
                    'HoldTrace', 'L=50uH')

opts['ModelVars']['varL'] = 100e-6;
result = server.plecs.simulate("BuckParamSweep", opts)
server.plecs.scope('BuckParamSweep/Scope',
                    'HoldTrace', 'L=100uH')
```

Scripted Simulation and Analysis Options

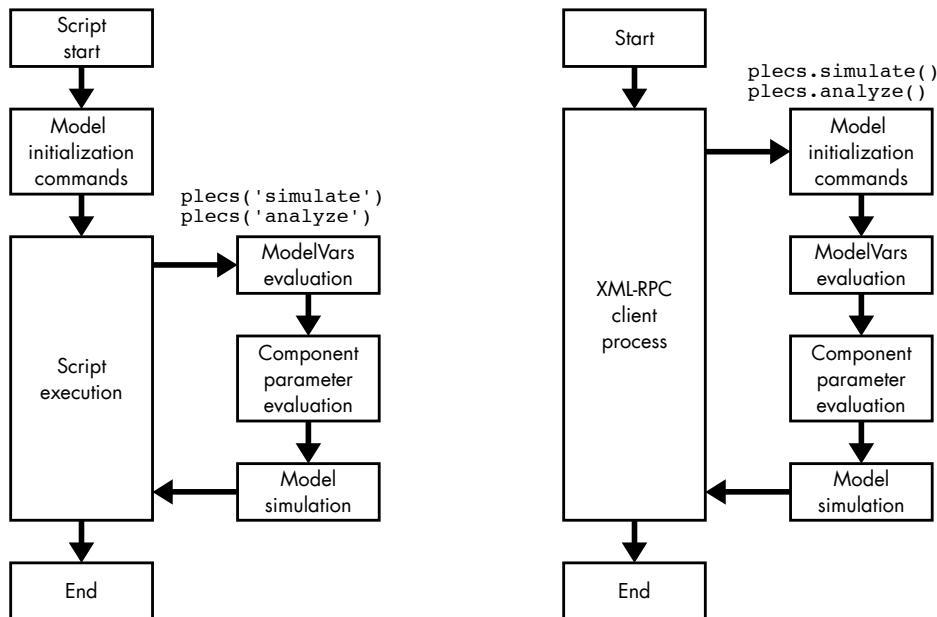
When you start a simulation or analysis from a Simulation Script or via XML-RPC, you can pass an optional argument *optStruct* in order to override param-

eter settings defined in the model. This enables you to run simulations for different scenarios without having to modify the model file.

The argument *optStruct* is a struct that may contain the fields `ModelVars`, `SolverOpts` and – when starting an analysis – `AnalysisOpts`, which are again structs as described below.

ModelVars The optional field `ModelVars` is a struct variable that allows you to override variable values defined by the model initialization commands. Each field name is treated as a variable name; the field value is assigned to the corresponding variable.

The override values are applied *after* the model initialization commands have been evaluated and *before* the component parameters are evaluated as shown in the figure below.



Execution order for Simulation Scripts (left) and XML-RPC (right)

SolverOpts The optional field `SolverOpts` is a struct variable that allows you to override the solver settings specified in the Simulation Parameters dialog. Each field name is treated as a solver parameter name; the field value is

assigned to the corresponding solver parameter. The following table lists the possible parameters.

Solver Options in Scripted Simulations

Parameter	Description
Solver	The solver to use for the simulation. Possible values are <code>dopri</code> for a non-stiff variable step solver, <code>radau</code> for a stiff variable step solver and <code>discrete</code> for a fixed step solver. See section “Standalone Parameters” (on page 75) for more details.
StartTime	The start time specifies the initial value of the simulation time variable t at the beginning of a simulation, in seconds.
StopTime	The simulation ends when the simulation time has advanced to the specified stop time.
MaxStep	See parameter Max Step Size in section “Standalone Parameters” (on page 75). This parameter is only evaluated for variable step solvers.
InitStep	See parameter Initial Step Size in section “Standalone Parameters” (on page 75). This parameter is only evaluated for variable step solvers.
FixedStep	This parameter specifies the fixed time increments for the solver and also the sample time used for the state-space discretization of the physical model. It is only evaluated for the fixed step solver.
AbsTol	See the description for Tolerances in section “Standalone Parameters” (on page 75).
RelTol	See the description for Tolerances in section “Standalone Parameters” (on page 75).
Refine	See parameter Refine factor in section “Standalone Parameters” (on page 75).

AnalysisOpts For an analysis the optional field `AnalysisOpts` is a struct variable that allows you to override the analysis settings defined in the Analysis Tools dialog. Each field name is treated as an analysis parameter name,

the field value is assigned to the corresponding analysis parameter. The following tables list the possible parameters

Analysis Options in Scripted Analyses

Parameter	Description
TimeSpan	System period length; this is the least common multiple of the periods of independent sources in the system.
StartTime	Simulation start time.
Tolerance	Relative error tolerance used in the convergence criterion of a steady-state analysis.
MaxIter	Maximum number of iterations allowed in a steady-state analysis.
JacobianPerturbation	Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.
JacobianCalculation	Controls the way the Jacobian matrix is calculated (full, fast). The default is fast.
InitCycles	Number of cycle-by-cycle simulations that should be performed before the actual analysis. This parameter can be used to provide the initial steady-state analysis with a better starting point.
ShowCycles	Number of steady-state cycles that should be simulated at the end of an analysis. This parameter is evaluated only for a steady-state analysis.
FrequencyRange	Range of the perturbation frequencies. This parameter is evaluated only for a small-signal analysis.
FrequencyScale	Specifies whether the sweep frequencies should be distributed on a linear or logarithmic scale. This parameter is evaluated only for a small-signal analysis.

Analysis Options in Scripted Analyses (contd.)

Parameter	Description
AdditionalFreqs	A vector specifying frequencies to be swept <i>in addition</i> to the automatically distributed frequencies. This parameter is evaluated only for a small-signal analysis.
NumPoints	The number of automatically distributed perturbation frequencies. This parameter is evaluated only for a small-signal analysis.
Perturbation	The full block path of the Small Signal Perturbation block that will be active during an analysis. This parameter is evaluated only for a small-signal analysis.
Response	The full block path of the Small Signal Response block that will record the system response during an analysis. This parameter is evaluated only for a small-signal analysis.
AmplitudeRange	The amplitude range of the sinusoidal perturbation signals for an ac sweep. This parameter is evaluated only for an ac sweep.
Amplitude	The amplitude of the discrete pulse perturbation for an impulse response analysis. This parameter is evaluated only for an impulse response analysis.

Code Generation with Real-Time Workshop

This section applies only to the PLECS Blockset.

The PLECS Blockset fully integrates with Real-Time Workshop to generate C code for your simulation model. Whenever you start the build process, PLECS automatically generates the code for a circuit block and inserts it at the appropriate places.

Note Scopes which are placed in PLECS schematics are not updated during a simulation using code generation. To view the simulation results all scopes must be placed in the Simulink model.

Code Generation Targets

PLECS can generate code for two different targets: the Rapid Simulation target (or RSim target) and the Real-Time target. These two targets are described in detail in the following two sections. The table below highlights the differences between the targets.

Comparison of Code Generation Targets

	RSim Target	Real-Time Target
Purpose	Rapid, non-real-time simulations.	Real-time simulations.
Technique	A compressed description of the circuit schematic is embedded in the code and interpreted at run time.	Signal and state-space equations are inlined as ANSI C code.
Limitations	none	Limited support for semiconductors and non-linear components.
Inlining	Parameters may be declared tunable, so that they are evaluated at run time.	All parameters are inlined, i.e. evaluated at compile time and embedded into the generated code.
Deployment	Requires distribution of the PLECS RSim module.	Generated code does not have external dependencies.
Licensing	Requires a PLECS license at run time.	Does not require a PLECS license at run time.

By default, PLECS automatically selects the appropriate target depending on the target settings of Real-Time Workshop. This selection can be overruled with the circuit parameter `RTWTarget`, see section “Code Generation Options” (on page 172).

Rapid Simulation Target

The RSim target is selected by default when you run a simulation using Simulink’s Rapid Accelerator mode or when you generate an executable using the RSim target or the S-Function target of Real-Time Workshop. The resulting code links against the RSim module of PLECS, a shared library which is part of the standard installation.

Deploying Rapid Simulation Executables

To deploy the generated executable you need to copy the appropriate shared library file onto the target computer. The following table lists the library files for the supported platforms.

Library Files for Rapid Simulations

Platform	Library File
Windows 32-bit	plecs\bin\win32\plecsrsim.dll
Windows 64-bit	plecs\bin\win64\plecsrsim.dll
Mac Intel 32-bit	plecs/bin/maci/libplecsrsim.dylib
Mac Intel 64-bit	plecs/bin/maci64/libplecsrsim.dylib
Linux 32-bit	plecs/bin/glnx86/libplecsrsim.so
Linux 64-bit	plecs/bin/glnxa64/libplecsrsim.so

The library file must be copied into the same directory as the executable. Alternatively, you can define the appropriate environment variable for your target computer such that it includes the directory where you have installed the library file.

Licensing Protocols for the PLECS RSim Module

The RSim module checks out a PLECS license for the duration of execution. It uses the environment variable `PLEXIM_LICENSE_FILE` to locate the license file. If the module is unable to check out a PLECS license, it issues an error message and stops the simulation.

Note The PLECS RSim module does not link against MATLAB. Therfore, it cannot accept license files that use MATLAB-based host IDs.

Tunable Circuit Parameters in Rapid Simulations

By default, PLECS evaluates the parameters of all circuit components at *compile time* and inlines them into the circuit description. However, for certain applications – such as rapid simulations on different parameter sets or parameterized S-Functions – it is desirable that the parameters be evaluated at *simulation start* instead. This can be achieved by declaring the circuit parameters tunable.

To declare circuit parameters tunable,

- 1 Mask the PLECS Circuit block and define all parameters that you wish to keep tunable as mask variables. Mask variables can either be mask parameters (that appear in the parameter dialog) or variables defined in the mask initialization commands. For more information see “Customizing the Circuit Block” (on page 37).
- 2 On the **Advanced** pane of the PLECS simulation parameters, uncheck the option **Inline circuit parameters for RSim target**.
- 3 Include the variable names in the list of tunable model parameters. Please see the Real-Time Workshop User’s Guide for details.

Limitations on Tunable Circuit Parameters

If you declare circuit parameters tunable, the RSim module uses its own parser to evaluate parameter expressions at simulation start; it currently cannot handle mask initialization commands. You will receive runtime errors if your circuit contains masked subsystems using mask initialization commands, or if a parameter expression contains a MATLAB function call.

Other limitations apply due to the way the Real-Time Workshop handles tunable parameters:

- Circuit parameters must be double-precision, 2-dimensional, non-sparse arrays.
- The first four characters of the parameter names must be unique.

Real-Time Target

The Real-Time Target is selected by default when you generate code using any of the real-time targets of Real-Time Workshop. Code generation for the Real-Time target requires a separate license for the PLECS Real-Time Coder.

Currently the code-generation capability of PLECS is subject to certain restrictions that are described below.

Unsupported Components

PLECS does not support code generation for the following components:

- Variable Inductor (see page 460)
- Variable Resistor with Variable Series Inductor (see page 470)
- Variable Resistor with Constant Series Inductor (see page 467)
- Variable Capacitor (see page 457)
- Variable Resistor with Variable Parallel Capacitor (see page 468)
- Variable Resistor with Constant Parallel Capacitor (see page 466)
- Brushless DC Machine (see page 203)
- Switched Reluctance Machine (see page 405)

Maximum Number of Switches

A single circuit may not contain more than **32 ideal switches**. This is due to the fact that the switching states of the individual switches are stored internally in a single unsigned integer variable. Note that some switch components, such as the Double Switch (see page 248) or Triple Switch (see page 455), consist of more than one ideal switch.

Limiting the Code Size

By default, PLECS will generate code for all 2^n possible combinations of switch states, where n is the number of the ideal switches that a circuit contains. (As a special case, the Double Switch and Triple Switch account only for 2 and 3 rather than 2^2 and 2^3 combinations each.) This can lead to large source and executable files and long compile times.

You can reduce the size of the generated code and the required compile time by specifying the combinations for which code should be generated in the circuit parameter `RTWTopologies`. If at run-time a combination is encountered for which no code has been produced, the execution is aborted with an error message. You *must* specify the combinations of switch states explicitly if PLECS finds that there are over 1024 (2^{10}) possible combinations.

Natural Commutation

By default, PLECS will not generate code for circuits containing naturally commutated switches, i.e. switches whose turn-on or turn-off conditions depend on voltage or current measurements. This includes all power semiconductors and the circuit breaker in the PLECS library. The reason for this is that the natural commutation instants of such switches typically do not coincide with the discrete sample times of a real-time program.

You can override the default behavior and generate code for naturally commutated switches by setting the circuit parameter `RTWAllowNaturalCommutation` to on. Note, however, that switching will still take place *only* at the discrete sample times. This can reduce the accuracy or even lead to inconsistent conditions after switching.

As an example, consider the turn-off of a diode connected in series with an inductor. In general the zero-crossing of the inductor current will not occur exactly at a simulation step; it will be either positive or negative. If it is negative, the diode will turn off. Unless there is another path for the inductor current, the circuit state will become inconsistent and the program execution is aborted.

You can override this type of error and instruct PLECS to enforce consistent conditions by setting the circuit parameter `RTWSuppressStateInconsistencies` to on. In the above example PLECS would then reset the inductor current to zero after the diode turn-off so as to obtain a consistent circuit state. Note, however, that this happens at the cost of a *discontinuity* of the inductor current which is not physically possible.

Code Generation Options

The following table lists the parameters that can be used to customize the code generation process.

Code Generation Options

Parameter	Description
RTWTarget	Specifies the code generation target. Possible values are auto, RSim and RealTime. The default is auto meaning that PLECS selects the target depending on the Real-Time Workshop target.
RTWTopologies	A matrix specifying the combinations of switch states for which code should be generated for the Real-Time target. The matrix must have n columns, where n is the number of ideal switches in the circuit. The default is [] meaning that PLECS will generate code for <i>all</i> possible combinations.
RTWAllowNaturalCommutation	Specifies for the Real-Time target whether code should be generated for circuits containing naturally commutated switches (such as diodes). Possible values are on and off. The default is off.
RTWSuppressStateInconsistencies	Specifies for the Real-Time target whether persistent state inconsistencies should be suppressed (on) or lead to a run-time error (off). The default is off.
RSimInlineCircuitParams	Specifies for the RSim target whether parameters should be evaluated at compile time and inlined into the code (on) or evaluated at run time (off). The default is on.

With the exception of RSimInlineCircuitParams these parameters can currently only be set via the command line interface. For example, the command below enables the code-generation for a circuit with naturally commutated switches:

```
plecs('set', circuit, 'RTWAllowNaturalCommutation', 'on');
```

In this example *circuit* is the full Simulink path of the circuit block.

Components by Category

This chapter lists the blocks of the Component library by category.

System

Configurable Subsystem	Provide subsystem with exchangeable implementations
Electrical Ground	Connect to common electrical ground
Electrical Label	Connect electrical potentials by name
Electrical Port	Add electrical connector to subsystem
Scope	Display simulation results versus time
Signal Demultiplexer	Split vectorized signal
Signal From	Reference signal from Signal Goto block by name
Signal Goto	Make signal available by name
Signal Import	Add signal input connector to subsystem
Signal Multiplexer	Combine several signals into vectorized signal
Signal Outport	Add signal output connector to subsystem
Signal Selector	Select or reorder elements from vectorized signal
Signal Switch	Select one of two input signals depending on control signal
Subsystem	Create functional entity in hierarchical simulation model

To File	Write time and signal values to file
Wire Multiplexer	Bundle several wires into bus
Wire Selector	Select or reorder elements from wire bus
XY Plot	Display correlation between two signals

Control

Sources

Clock	Provide current simulation time
Constant	Generate constant signal
Pulse Generator	Generate periodic rectangular pulses
Ramp	Generate constantly rising or falling signal
Sine Wave	Generate time-based sine wave with optional bias
Step	Generate constant signal with instantaneous step change
Triangular Wave Generator	Generate periodic triangular or sawtooth waveform

Math

Abs	Calculate absolute value of input signal
Gain	Multiply input signal by constant
Math Function	Apply specified mathematical function
Minimum / Maximum	Output input signal with highest resp. lowest value
Product	Multiply and divide scalar or vectorized input signals
Rounding	Round floating point signal to integer values
Signum	Provide sign of input signal
Sum	Add and subtract input signals

Trigonometric Function

Apply specified trigonometric function

Continuous**Integrator**

Integrate input signal with respect to time

State Space

Implement linear time-invariant system as state-space model

Transfer Function

Model linear time-invariant system as transfer function

Delays**Memory**

Provide input signal from previous major time step

Pulse Delay

Delay discrete-value input signal by fixed time

Transport Delay

Delay continuous input signal by fixed time

Turn-on Delay

Delay rising flank of input pulses by fixed dead time

Discontinuous**Comparator**

Compare two input signals with minimal hysteresis

Dead Zone

Output zero while input signal is within dead zone limits

Hit Crossing

Detect when signal reaches or crosses given value

Quantizer

Apply uniform quantization to input signal

Rate Limiter

Limit rising and falling rate of change

Relay

Toggle between on- and off-state with configurable threshold

Saturation

Limit input signal to upper and/or lower value

Discrete

Delay	Delay input signal by given number of samples
Discrete Fourier Transform	Perform discrete Fourier transform on input signal
Discrete Mean Value	Calculate running mean value of input signal
Discrete RMS Value	Calculate root mean square (RMS) value of input signal
Discrete Total Harmonic Distortion	Calculate total harmonic distortion (THD) of input signal
Discrete Transfer Function	Model discrete system as transfer function
Zero-Order Hold	Sample and hold input signal periodically

Filters

Moving Average	Continuously average input signal over specified time period
Periodic Average	Periodically average input signal over specified time
Periodic Impulse Average	Periodically average Dirac impulses over specified time

Functions & Tables

1D Look-Up Table	Compute piece-wise linear function of one input signal
2D Look-Up Table	Compute piece-wise linear function of two input signals
3D Look-Up Table	Compute piece-wise linear function of three input signals
C-Script	Execute custom C code
DLL	Interface with externally generated dynamic-link library

Fourier Series	Synthesize periodic output signal from Fourier coefficients
Function	Apply arbitrary arithmetic expression to scalar or vectorized input signal

Logical

Combinatorial Logic	Use binary input signals to select one row from truth table
D Flip-flop	Implement edge-triggered flip-flop
Edge Detection	Detect edges of pulse signal in given direction
JK Flip-flop	Implement edge-triggered JK flip-flop
Logical Operator	Combine input signals logically
Monoflop	Generate pulse of specified width when triggered
Relational Operator	Compare two input signals
SR Flip-flop	Implement set-reset flip-flop

Modulators

2-Pulse Generator	Generate firing pulses for H-bridge thyristor rectifier
3-Phase Overmodulation	Extend linear range of modulation index for 3-phase inverters
6-Pulse Generator	Generate firing pulses for 3-phase thyristor rectifier
Blanking Time	Generate commutation delay for 2-level inverter bridges
Blanking Time (3-Level)	Generate commutation delay for 3-level inverter bridges
Peak Current Controller	Implement peak current mode control
Sawtooth PWM	Generate PWM signal using sawtooth carrier

Sawtooth PWM (3-Level)	Generate 3-level PWM signal using sawtooth carriers
Space Vector Modulator	Generate PWM signals for 3-phase inverter using space-vector modulation technique
Symmetrical PWM	Generate PWM signal using symmetrical triangular carrier
Symmetrical PWM (3-Level)	Generate 3-level PWM signal using symmetrical triangular carriers

Transformations

Polar to Rectangular	Convert polar coordinates to Cartesian coordinates
Rectangular to Polar	Convert Cartesian coordinates to polar coordinates
Transformation 3ph->RRF	Transform 3-phase signal to rotating reference frame
Transformation 3ph->SRF	Transform 3-phase signal to stationary reference frame
Transformation RRF->3ph	Transform vector in rotating reference frame into 3-phase signal
Transformation RRF->SRF	Transform vector from rotating to stationary reference frame
Transformation SRF->3ph	Transform vector in stationary reference frame into 3-phase signal
Transformation SRF->RRF	Transform vector from stationary to rotating reference frame

Small Signal Analysis

(PLECS Standalone only)

Small Signal Gain	Measure loop gain of closed control loop using small signal analysis
Small Signal Perturbation	Generate perturbation signal for small signal analysis

Small Signal Response

Measure system response for small signal analysis

Electrical

Sources

Current Source (Controlled)	Generate variable current
Current Source AC	Generate sinusoidal current
Current Source DC	Generate constant current
Voltage Source (Controlled)	Generate variable voltage
Voltage Source AC	Generate sinusoidal voltage
Voltage Source AC (3-Phase)	Generate 3-phase sinusoidal voltage
Voltage Source DC	Generate constant voltage

Meters

Ammeter	Output measured current as signal
Meter (3-Phase)	Measure voltages and currents of 3-phase system
Voltmeter	Output measured voltage as signal

Passive Components

Capacitor	Ideal capacitor
Inductor	Ideal inductor
Mutual Inductor	Ideal mutual inductor
Mutual Inductance (2 Windings)	Magnetic coupling between two lossy windings
Mutual Inductance (3 Windings)	Magnetic coupling between three lossy windings

Pi-Section Line	Single-phase pi-section transmission line
Piece-wise Linear Resistor	Resistance defined by voltage-current pairs
Resistor	Ideal resistor
Saturable Capacitor	Capacitor with piece-wise linear saturation
Saturable Inductor	Inductor with piece-wise linear saturation
Variable Capacitor	Capacitance controlled by signal
Variable Inductor	Inductance controlled by signal
Variable Resistor with Constant Capacitor	Controlled resistance in parallel with constant capacitance
Variable Resistor with Constant Inductor	Controlled resistance in series with constant inductance
Variable Resistor with Variable Capacitor	Controlled resistance in parallel with controlled capacitance
Variable Resistor with Variable Inductor	Controlled resistance in series with controlled inductance

Power Semiconductors

Diode	Ideal diode with optional forward voltage and on-resistance
Diode with Reverse Recovery	Dynamic diode model with reverse recovery
GTO	Ideal GTO with optional forward voltage and on-resistance
GTO (Reverse Conducting)	Ideal GTO with ideal anti-parallel diode
IGBT	Ideal IGBT with optional forward voltage and on-resistance
IGBT with Diode	Ideal IGBT with ideal anti-parallel diode
IGBT with Limited di/dt	Dynamic IGBT model with finite current slopes during turn-on and turn-off
IGCT (Reverse Blocking)	Ideal IGCT with optional forward voltage and on-resistance

IGCT (Reverse Conducting)	Ideal IGCT with ideal anti-parallel diode
MOSFET	Ideal MOSFET with optional on-resistance
MOSFET with Diode	Ideal MOSFET with ideal anti-parallel diode
MOSFET with Limited di/dt	Dynamic MOSFET model with finite current slopes during turn-on and turn-off
Thyristor	Ideal thyristor (SCR) with optional forward voltage and on-resistance
Thyristor with Reverse Recovery	Dynamic thyristor (SCR) model with reverse recovery
TRIAC	Ideal TRIAC with optional forward voltage and on-resistance
Zener Diode	Zener diode with controlled reverse breakdown voltage

Switches

Breaker	AC circuit breaker opening at zero current
Double Switch	Changeover switch with two positions
Set/Reset Switch	Bistable on-off switch
Switch	On-off switch
Triple Switch	Changeover switch with three positions

Transformers

Ideal Transformer	Ideally coupled windings without inductance
Linear Transformer (2 Windings)	Single-phase transformer with winding resistance and optional core loss
Linear Transformer (3 Windings)	Single-phase transformer with winding resistance and optional core loss
Saturable Transformers	Single-phase transformers with two resp. three windings and core saturation

Transformers (3ph, 2 Windings)	3-phase transformers in Yy, Yd, Yz, Dy, Dd and Dz connection
Transformers (3ph, 3 Windings)	3-phase transformers in Ydy and Ydz connection

Machines

Brushless DC Machine	Detailed model of brushless DC machine excited by permanent magnets
Brushless DC Machine (Simplified)	Simple model of brushless DC machine excited by permanent magnets
DC Machine	Simple model of DC machine
Induction Machine	Non-saturable induction machine with slip-ring rotor
Induction Machine (Open Stator Windings)	Non-saturable induction machine with squirrel-cage rotor and open stator windings
Induction Machine (Squirrel-Cage)	Non-saturable induction machine with squirrel-cage rotor
Induction Machine with Saturation	Induction machine with slip-ring rotor and main-flux saturation
Permanent Magnet Synchronous Machine	Synchronous machine excited by permanent magnets
Switched Reluctance Machine	Detailed model of switched reluctance machine with open windings
Synchronous Machine (Round Rotor)	Smooth air-gap synchronous machine with main-flux saturation
Synchronous Machine (Salient Pole)	Salient pole synchronous machine with main-flux saturation

Converters

Diode Rectifier (3ph)	3-phase diode rectifier
------------------------------	-------------------------

Ideal 3-Level Converter (3ph)	Switch-based 3-phase 3-level converter
Ideal Converter (3ph)	Switch-based 3-phase converter
IGBT 3-Level Converter (3ph)	3-phase 3-level neutral-point clamped IGBT converter
IGBT Converter (3ph)	3-phase IGBT converter
MOSFET Converter (3ph)	3-phase MOSFET converter
Thyristor Rectifier/Inverter	3-phase thyristor rectifier/inverter

Electronics

Op-Amp	Ideal operational amplifier with finite gain
Op-Amp with Limited Output	Ideal operational amplifier with limited output voltage

Thermal

Ambient Temperature	Connect to Heat Sink on which component is placed
Constant Heat Flow	Generate constant heat flow
Constant Temperature	Provide constant temperature
Controlled Heat Flow	Generate variable heat flow
Controlled Temperature	Provide variable temperature
Heat Flow Meter	Output measured heat flow as signal
Heat Sink	Isotherm environment for placing components
Thermal Capacitor	Thermal capacitance of piece of material
Thermal Chain	Thermal impedance implemented as RC chain
Thermal Ground	Connect to common reference temperature
Thermal Port	Add thermal connector to subsystem
Thermal Resistor	Thermal resistance of piece of material

Thermometer Output measured temperature as signal

Magnetic

Winding	Ideal winding defining an electro-magnetic interface
Magnetic Permeance	Linear magnetic permeance
Linear Core	Linear magnetic core element
Air Gap	Air gap in a magnetic core
Leakage Flux Path	Permeance of linear leakage flux path
Saturable Core	Magnetic core element with saturation
Hysteretic Core	Magnetic core element with static hysteresis
Variable Magnetic Permeance	Variable permeance controlled by external signal
Magnetic Resistance	Effective magnetic resistance for modeling losses
MMF Meter	Output the measured magneto-motive force
Flux Rate Meter	Output the measured rate-of-change of magnetic flux
MMF Source (Constant)	Generate a constant magneto-motive force
MMF Source (Controlled)	Generate a variable magneto-motive force
Magnetic Port	Add magnetic connector to subsystem

Additional Simulink Blocks

(*PLECS Blockset only*)

AC Sweep	Perform AC sweep
Impulse Response Analysis	Perform impulse response analysis
Loop Gain Analysis	Determine loop gain of closed control loop
Steady-State Analysis	Determine periodic steady-state operating point

Timer

Generate piece-wise constant signal

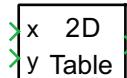
Component Reference

This chapter lists the contents of the Component library in alphabetical order.

1D Look-Up Table

Purpose	Compute piece-wise linear function of one input signal
Library	Control / Functions & Tables
Description	<p>The 1D Look-Up Table block maps an input signal to an output signal. You define the mapping function by specifying a vector of input values and a vector of output values. If the input signal lies within the range of the input vector, the output value is calculated by linear interpolation between the appropriate two points. If the input signal is out of bounds, the block extrapolates using the first or last two points.</p> <p>Step transitions are achieved by repeating an input value with different output values. If the input signal exactly matches the input value of such a discontinuity, the output signal will be the output value of the mapping function that is first encountered when moving away from the origin. If the discontinuity is at input value 0, the output signal will be the average of the two output values. This behavior can be overridden by defining <i>three</i> output values for the same input value; in this case the middle output value will be chosen.</p> <p>Use the 2D Look-Up Table block (see page 191) to map two input signals to an output signal.</p>
Parameters	<p>Vector of input values x The vector of input values x. This vector must be the same size as the output vector and monotonically increasing. It should not contain more than three identical values.</p> <p>Vector of output values $f(x)$ The vector containing the output values $f(x)$. This vector must be the same size as the input vector.</p>
Probe Signals	<p>Input The block input signal.</p> <p>Output The block output signal.</p>

2D Look-Up Table

Purpose	Compute piece-wise linear function of two input signals
Library	Control / Functions & Tables
Description	<p>The 2D Look-Up Table block maps two input signals to an output signal. You define the mapping function by specifying two vectors of input values and a matrix of output values. The input vector x corresponds to the <i>rows</i> of the output matrix, the input vector y, to the <i>columns</i>.</p> <p>The output value is interpolated or extrapolated from the block parameters using the technique described for the 1D Look-Up Table block (see page 190).</p>
 Parameters	<p>Vector of input values x The vector of input values x. This vector must be the same size as the number of rows in the output matrix and monotonically increasing. It should not contain more than three identical values.</p> <p>Vector of input values y The vector of input values y. This vector must be the same size as the number of columns in the output matrix and monotonically increasing. It should not contain more than three identical values.</p> <p>Matrix of output values $f(x,y)$ The matrix containing the output values $f(x,y)$. The number of rows and columns must match the size of the input vectors.</p>
Probe Signals	<p>Input x The block input signal x.</p> <p>Input y The block input signal y.</p> <p>Output The block output signal.</p>

2-Pulse Generator

Purpose Generate firing pulses for H-bridge thyristor rectifier

Library Control / Modulators

Description This block generates the pulses used to fire the thyristors of an H-bridge rectifier. The inputs of the block are a logical enable signal, a ramp signal φ (produced e.g. by a PLL), and the firing angle alpha.

```
>enable  
>phi  pulses  
>alpha
```

3D Look-Up Table

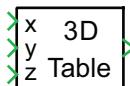
Purpose

Compute piece-wise linear function of three input signals

Library

Control / Functions & Tables

Description



The 3D Look-Up Table block maps three input signals to an output signal. You define the mapping function by specifying three vectors of input values and an array of output values. The input vectors x , y and z correspond to the *first*, *second* and *third* dimension of the output array.

The output value is interpolated or extrapolated from the block parameters using the technique described for the 1D Look-Up Table block (see page 190).

Parameters

Vector of input values x

The vector of input values x . This vector must be the same size as the size of the first dimension in the output array and monotonically increasing. It should not contain more than three identical values.

Vector of input values y

The vector of input values y . This vector must be the same size as the size of the second dimension in the output array and monotonically increasing. It should not contain more than three identical values.

Vector of input values z

The vector of input values z . This vector must be the same size as the size of the third dimension in the output array and monotonically increasing. It should not contain more than three identical values.

3D array of output values $f(x,y,z)$

The array containing the output values $f(x, y, z)$. The dimensions must match the size of the input vectors.

Probe Signals

Input x

The block input signal x .

Input y

The block input signal y .

Input z

The block input signal z .

Output

The block output signal.

3-Phase Overmodulation

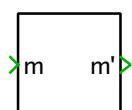
Purpose

Extend linear range of modulation index for 3-phase inverters

Library

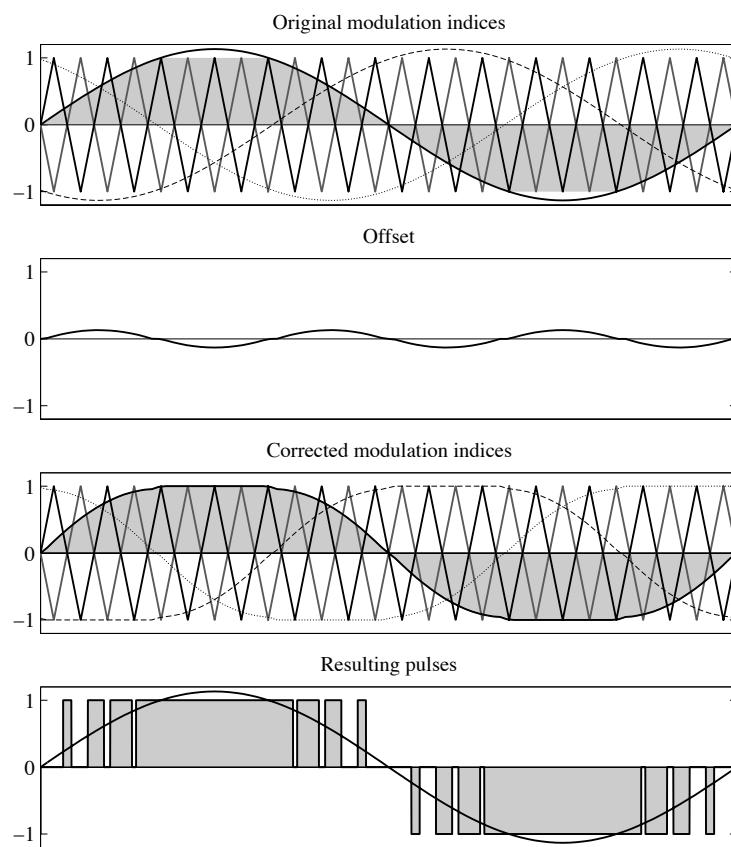
Control / Modulators

Description



For three-phase signals, this block extends the linear range of the modulation index from [-1 1] to [-1.154 1.154] by adding a zero-sequence offset. This block may be used for the control of three-phase converters without neutral point connection such as the IGBT Converter (see page 273).

The figures below illustrates the working principle of the 3-Phase Overmodulation block in conjunction with the Symmetrical PWM (see page 409).



6-Pulse Generator

Purpose

Generate firing pulses for 3-phase thyristor rectifier

Library

Control / Modulators

Description

- >enable
- >phi pulses
- >alpha

This block generates the pulses used to fire the thyristors of a 6-pulse rectifier or inverter. The inputs of the block are a logical enable signal, a ramp signal φ (produced e.g. by a PLL), and the firing angle alpha.

If the “Double pulses” option is selected, each thyristor receives two pulses: one when the firing angle is reached, and a second, when the next thyristor is fired.

Parameters

Pulse width

The width of the firing pulses in radians with respect to one period of fundamental frequency.

Pulse type

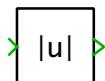
“Double pulses” enables a second firing pulse for each thyristor.

Abs

Purpose Calculate absolute value of input signal

Library Control / Math

Description The Abs block outputs the absolute value of the input signals, $y = |u|$.



Probe Signals

Input

The block input signal.

Output

The block output signal.

Ambient Temperature

Purpose	Connect to Heat Sink on which component is placed
Library	Thermal
Description	The Ambient Temperature is only useful in subsystems. When placed in a subsystem, it provides a thermal connection to the heat sink that encloses the subsystem. For more information see section “Heat Sinks and Subsystems” (on page 84).
	

Note Ambient Temperature blocks may not be used in schematics that contain Thermal Port blocks (see page 427).

Air Gap

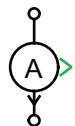
Purpose	Air gap in a magnetic core
Library	Magnetic
Description	This component models an air gap in a magnetic core. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force \mathcal{F}
	$\frac{\Phi}{\mathcal{F}} = \frac{\mu_0 A}{l}$ where $\mu_0 = 4\pi \times 10^{-7} \text{ N/A}^2$ is the magnetic constant, A is the cross-sectional area and l the length of the flux path.
Parameters	<p>Cross-sectional area Effective cross-sectional area A of the air gap, in m^2.</p> <p>Length of flux path Effective length l of the air gap, in m.</p> <p>Initial MMF Magneto-motive force at simulation start, in ampere-turns (A).</p>
Probe Signals	<p>MMF The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).</p> <p>Flux The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.</p> <p>Field strength The magnetic field strength H in the air gap, in A/m.</p> <p>Flux density The magnetic flux density B in the air gap, in teslas (T).</p>

Ammeter

Purpose Output measured current as signal

Library Electrical / Sources

Description



The Ammeter measures the current through the component and provides it as a signal at the output. The direction of a positive current is indicated with a small arrow in the component symbol. The output signal can be made accessible in Simulink with an Output block (see page 387) or by dragging the component into the dialog box of a Probe block.

Note The Ammeter is ideal, i.e. it has zero internal resistance. Hence, if multiple ammeters are connected in parallel the current through an individual ammeter is undefined. This produces a run-time error.

Likewise, if switches connected in parallel are all in closed position the current through the individual switches is not properly defined. Although this does not produce a run-time error it may lead to unexpected simulation results.

Probe Signals

Measured current

The measured current in amperes (A).

Blanking Time

Purpose	Generate commutation delay for 2-level inverter bridges
Library	Control / Modulators
Description	This block generates a blanking time for 2-level inverter bridges so that the turn-on of one switch is delayed with respect to the turn-off of the other switch in the same inverter leg. 
	The input s is a switching function with the values -1 and 1 generated by a 2-level modulator such as the Symmetrical PWM generator (see page 409). The values of the output s' are either -1 (lower switch turned on), 0 (both switches off) or 1 (upper switch on). If the input is a vector, the output is also a vector of the same width.
Parameter	Delay time The delay in seconds (s) between the turn-off of one switch and the turn-on of the other switch in an inverter leg.

Blanking Time (3-Level)

Purpose

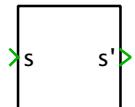
Generate commutation delay for 3-level inverter bridges

Library

Control / Modulators

Description

This block generates a blanking time for 3-level inverter bridges so that the turn-on of one switch is delayed with respect to the turn-off of the other switch in the same inverter leg.

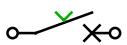


The input s is a switching function with the values -1, 0 and 1 generated by a 3-level modulator such as the Symmetrical PWM (3-Level) generator (see page 411). The values of the output s' are either -1, -0.5, 0, 0.5 or 1. If the input is a vector, the output is also a vector of the same width.

Parameter**Delay time**

The delay in seconds (s) between the turn-off of one switch and the turn-on of another switch in an inverter leg.

Breaker

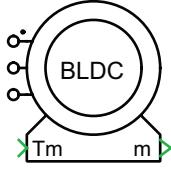
Purpose	AC circuit breaker opening at zero current
Library	Electrical / Switches
Description	This component provides an ideal short or open circuit between its two electrical terminals. The switch closes when the controlling signal becomes non-zero. It opens when both the signal and the current are zero. Therefore, this circuit breaker can be used to interrupt inductive AC currents.
	
Parameter	Initial conductivity Initial conduction state of the breaker. The breaker is initially open if the parameter evaluates to zero, otherwise closed. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.
Probe Signals	Breaker current The current through the component in amperes (A). A positive current flows from the left to the right terminal in the above breaker icon. Breaker conductivity Conduction state of the internal switch. The signal outputs 0 if the breaker is open, and 1 if it is closed.

Brushless DC Machine

Purpose Detailed model of brushless DC machine excited by permanent magnets

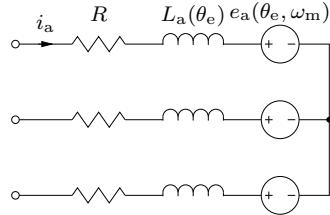
Library Electrical / Machines

Description A brushless DC machine is a type of permanent magnet synchronous machine in which the back electromotive force (EMF) is not sinusoidal but has a more or less trapezoidal shape. Additionally, the variation of the stator inductance with the rotor position is not necessarily sinusoidal.



The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. In the component icon, phase a of the stator winding is marked with a dot.

Electrical System



The back EMF voltages are determined by a shape function k_e and the mechanical rotor speed ω_m . The shape function in turn is expressed as a fourier series of the electrical rotor angle θ_e :

$$e_x(\theta_e, \omega_m) = k_{e,x}(\theta_e) \cdot \omega_m$$

$$k_{e,a}(\theta_e) = \sum_n K_{c,n} \cos(n\theta_e) + K_{s,n} \sin(n\theta_e)$$

$$k_{e,b}(\theta_e) = \sum_n K_{c,n} \cos(n\theta_e - \frac{2\pi n}{3}) + K_{s,n} \sin(n\theta_e - \frac{2\pi n}{3})$$

$$k_{e,c}(\theta_e) = \sum_n K_{c,n} \cos(n\theta_e + \frac{2\pi n}{3}) + K_{s,n} \sin(n\theta_e + \frac{2\pi n}{3})$$

The stator self inductance is also expressed as a fourier series of the electrical rotor angle. The mutual inductance M between the stator phases is assumed to be constant. Since the stator windings are star connected, the mutual inductance can simply be subtracted from the self inductance:

$$L_a(\theta_e) = L_0 - M + \sum_n L_{c,n} \cos(n\theta_e) + L_{s,n} \sin(n\theta_e)$$

Electromechanical System

The electromagnetic torque is a superposition of the torque caused by the permanent magnet and a reluctance torque caused by the non-constant stator inductance:

$$T_e = \sum_{x=a,b,c} k_{e,x} i_x + \frac{p}{2} \frac{dL_x}{d\theta_e} i_x^2$$

The cogging torque is again expressed as a fourier series of the electrical rotor angle:

$$T_{cog}(\theta_e) = \sum_n T_{c,n} \cos(n\theta_e) + T_{s,n} \sin(n\theta_e)$$

Mechanical System

Mechanical rotor speed:

$$\dot{\omega}_m = \frac{1}{J} (T_e + T_{cog}(\theta_e) - F\omega_m - T_m)$$

Mechanical and electrical rotor angle:

$$\dot{\theta}_m = \omega_m$$

$$\theta_e = p \cdot \theta_m$$

Parameters

Back EMF shape coefficients

Fourier coefficients $K_{c,n}$ and $K_{s,n}$ of the back EMF shape function $k_{e,a}(\theta_e)$ in Vs.

Stator resistance

The stator resistance R in ohms (Ω).

Stator inductance

The constant inductance $L_0 - M$ and the fourier coefficients $L_{c,n}$, $L_{s,n}$ of the phase a inductance $L_a(\theta_e)$ in henries (H).

Cogging torque coefficients

Fourier coefficients $T_{c,n}$, $T_{s,n}$ of the cogging torque $T_{\text{cog}}(\theta_e)$ in Nm.

Inertia

Combined rotor and load inertia J in Nms².

Friction coefficient

Viscous friction F in Nms.

Number of pole pairs

Number of pole pairs p .

Initial rotor speed

Initial mechanical speed $\omega_{m,0}$ in radians per second (s⁻¹).

Initial rotor angle

Initial mechanical rotor angle $\theta_{m,0}$ in radians.

Initial stator currents

A two-element vector containing the initial stator currents $i_{a,0}$ and $i_{b,0}$ of phase a and b in amperes (A).

Inputs and Outputs**Mechanical torque**

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector “m” contains the following 7 signals:**(1) Rotor speed**

The rotational speed ω_m of the rotor in radians per second (s⁻¹).

(2) Rotor position

The mechanical rotor angle θ_m in radians.

(3) Electrical torque

The electrical torque T_e of the machine in Nm.

(4) Cogging torque

The cogging torque T_{cog} of the machine in Nm.

(5-7) Back EMF voltages

The back EMF voltages e_a , e_b , e_c in volts (V).

Probe Signals**Stator phase currents**

The three-phase stator winding currents i_a , i_b and i_c , in A. Currents flowing into the machine are considered positive.

Back EMF

The back EMF voltages e_a , e_b , e_c in volts (V).

Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

Rotor position

The mechanical rotor angle θ_m in radians.

Electrical torque

The electrical torque T_e of the machine in Nm.

References

- D. Hanselman, "Brushless permanent magnet motor design, 2nd ed.", The Writers' Collective, Mar. 2003.
- P. Pillay, R. Krishnan, "Modeling, simulation, and analysis of permanent-magnet motor drives, Part II: The brushless DC motor drive", IEEE Trans. on Ind. App., Vol. 25, No. 2, Mar./Apr. 1989.

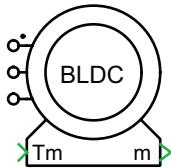
Brushless DC Machine (Simplified)

Purpose Simple model of brushless DC machine excited by permanent magnets

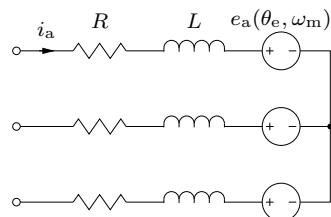
Library Electrical / Machines

Description The simplified Brushless DC Machine is a model of a permanent magnet synchronous machine with sinusoidal or trapezoidal back EMF.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. In the component icon, phase a of the stator winding is marked with a dot.

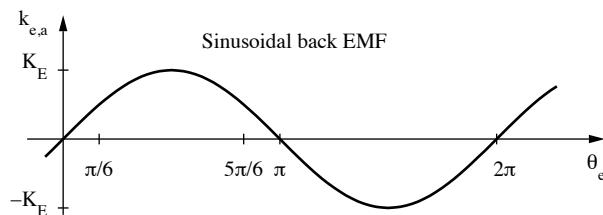


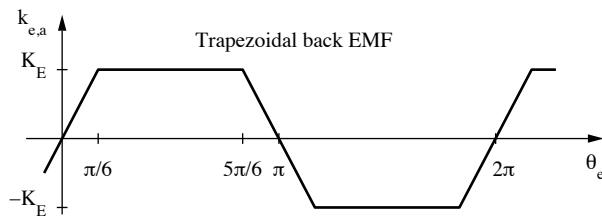
Electrical System



The back EMF voltages are determined by a shape function k_e and the mechanical rotor speed ω_m . The shape function is a sinusoidal or an ideal trapezoidal function scaled with the back EMF constant K_E .

$$e_x(\theta_e, \omega_m) = k_{e,x}(\theta_e) \cdot \omega_m$$





Electromechanical System

The electromagnetic torque is:

$$T_e = \sum_{x=a,b,c} k_{e,x} i_x$$

Mechanical System

Mechanical rotor speed:

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

Mechanical and electrical rotor angle:

$$\dot{\theta}_m = \omega_m$$

$$\theta_e = p \cdot \theta_m$$

Parameters

Back EMF shape

Choose between sinusoidal and trapezoidal back EMF.

Back EMF constant

The back EMF constant K_E in Vs.

Stator resistance

The stator resistance R in ohms (Ω).

Stator inductance

The stator inductance $L - M$ in henries (H).

Inertia

Combined rotor and load inertia J in Nms².

Friction coefficient

Viscous friction F in Nms.

Number of pole pairs

Number of pole pairs p .

Initial rotor speed

Initial mechanical speed $\omega_{m,0}$ in radians per second (s^{-1}).

Initial rotor angle

Initial mechanical rotor angle $\theta_{m,0}$ in radians.

Initial stator currents

A two-element vector containing the initial stator currents $i_{a,0}$ and $i_{b,0}$ of phase a and b in amperes (A).

Inputs and Outputs**Mechanical torque**

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector "m" contains the following 6 signals:**(1) Rotor speed**

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

(2) Rotor position

The mechanical rotor angle θ_m in radians.

(3) Electrical torque

The electrical torque T_e of the machine in Nm.

(4-6) Back EMF voltages

The back EMF voltages e_a, e_b, e_c in volts (V).

Probe Signals**Stator phase currents**

The three-phase stator winding currents i_a, i_b and i_c , in A. Currents flowing into the machine are considered positive.

Back EMF

The back EMF voltages e_a, e_b, e_c in volts (V).

Stator flux (dq)

The stator flux linkages Ψ_d and Ψ_q in the stationary reference frame in Vs.

Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

Rotor position

The mechanical rotor angle θ_m in radians.

Electrical torque

The electrical torque T_e of the machine in Nm.

References

- D. Hanselman, "Brushless permanent magnet motor design, 2nd ed.", The Writers' Collective, Mar. 2003.
- P. Pillay, R. Krishnan, "Modeling, simulation, and analysis of permanent-magnet motor drives, Part II: The brushless DC motor drive", IEEE Trans. on Ind. App., Vol. 25, No. 2, Mar./Apr. 1989.

See also

For back EMF shapes other than sinusoidal or trapezoidal, and/or if the stator inductance L is angle dependent please use the sophisticated model of the Brushless DC Machine (see page 203). The sophisticated BLDC machine can be configured as a simple BLDC machine with sinusoidal back EMF if the parameters are converted as follows:

$$K_{c,n} = [0]$$

$$K_{s,n} = [K_E]$$

$$L_0 - M = L - M$$

$$L_{c,n} = [0]$$

$$L_{s,n} = [0]$$

For machines with sinusoidal back EMF you may also consider to use the Permanent Magnet Synchronous Machine (see page 345). The parameters can be converted as follows provided that the stator inductance L is independent of the rotor angle:

$$[L_d \ L_q] = [L - M \ L - M]$$

$$\varphi'_m = -K_E/p$$

C-Script

Purpose Execute custom C code

Library Control / Functions & Tables

Description The C-Script block allows for custom functionality to be implemented in the C programming language. For a detailed description of C-Scripts see chapter “C-Scripts” (on page 133).

C-Script

The C-Script dialog consists of two tabbed panes that are described below.

Setup

Number of inputs, outputs, cont. states, disc. states, zero-crossings

An integer scalar specifying the sizes of the different data vectors (i.e. input and output signals, continuous and discrete state variables, and zero-crossing signals) that the C-Script registers with the solver. The number of input and output signals may not be zero.

The data vectors can also be sized *dynamically* at simulation start depending on the number of elements in the signal that is connected to the input port. For dynamic sizing set the number of input signals to -1. Any other data vector with a setting of -1 will be expanded to the same width.

Sample time

A scalar or an $n \times 2$ matrix specifying the block sample time(s). The table below lists the valid parameter values for the different sample time types. For a detailed description of the sample time types see “Sample Time” (on page 138).

Type	Value
Continuous	[0, 0] or 0
Semi-Continuous	[0, -1]
Discrete-Periodic	$[T_p, T_o]$ or T_p T_p : Sample period, $T_p > 0$ T_o : Sample offset, $0 \leq T_o < T_p$
Discrete-Variable	[-2, 0] or -2

Direct feedthrough

A vector of zeros and ones specifying the direct feedthrough flags for the input signals. An input signal has direct feedthrough if you need to access the current input signal value during the output function call. This has an influence on the block sorting order and the occurrence of algebraic loops (see “Block Sorting” on page 29). You can also specify a single scalar, which then applies to all input signals.

Language standard

The language standard used by the compiler. Possible values are C90 and C99. The default is C99.

Enable runtime checks

If this box is checked, protective code is added to guard against access violations when working with block data (i.e. signal values, states, zero-crossing signals etc.). The C-Script function calls are also wrapped with protective code to prevent you from violating solver policies such as accessing input signals in the output function without enabling direct feedthrough.

It is strongly recommended to leave the runtime checks enabled.

Parameters

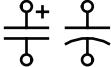
A comma-separated list of expressions that are passed as external parameters into the C functions. The expressions can reference workspace variables and must evaluate to scalars, vectors, matrices or 3d-arrays.

Code

The **Code** pane consists of a combobox for selecting a particular code section and a text editor that lets you edit the currently selected code section. For details on the individual sections see “C-Script Functions” (on page 134). The different macros that you need to use in order to access block data such as input/output signals and states are listed in “C-Script Macros” (on page 148).

If you have made changes to the C code, it will be compiled when you click on **Apply** or **OK**. Any errors or warnings that occur during compilation are listed in a diagnostic window. Small badges next to the line numbers indicate the problematic code lines. If you move the mouse cursor near such a badge, a tooltip with the diagnostics for that line will appear.

Capacitor

Purpose	Ideal capacitor
Library	Electrical / Passive Components
Description	This component provides one or more ideal capacitors between its two electrical terminals. If the component is vectorized, a coupling can be modeled between the internal capacitors. Capacitors may be switched in parallel only if their momentary voltages are equal.
	See section “Configuring PLECS” (on page 35) for information on how to change the graphical representation of capacitors.

Note A capacitor may not be connected in parallel with an ideal voltage source. Doing so would create a dependency between an input variable (the source voltage) and a state variable (the capacitor voltage) in the underlying state-space equations.

Parameters	Capacitance The value of the capacitor, in farads (F). All finite positive and negative values are accepted, including 0. The default is 100e-6. In a vectorized component, all internal capacitors have the same value if the parameter is a scalar. To specify the capacitances individually use a vector $[C_1 \ C_2 \ \dots \ C_n]$. The length n of the vector determines the component's width:
-------------------	---

$$\begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} C_1 & 0 & \cdots & 0 \\ 0 & C_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt}v_1 \\ \frac{d}{dt}v_2 \\ \vdots \\ \frac{d}{dt}v_n \end{bmatrix}$$

In order to model a coupling between the internal capacitors enter a square matrix. The size n of the matrix corresponds to the width of the component:

$$\begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} C_1 & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_2 & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n,1} & C_{n,2} & \cdots & C_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt}v_1 \\ \frac{d}{dt}v_2 \\ \vdots \\ \frac{d}{dt}v_n \end{bmatrix}$$

The capacitance matrix must be invertible, i.e. it may not be singular.

Initial voltage

The initial voltage of the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the width of the component. The positive pole is marked with a “+”. The initial voltage default is 0.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with “+” is greater than the potential at the unmarked terminal.

Capacitor current

The current flowing through the capacitor, in amperes (A).

Clock

Purpose Provide current simulation time

Library Control / Sources

Description The Clock block outputs the current simulation time.



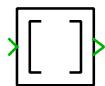
Probe Signals **Output**
The time signal.

Combinatorial Logic

Purpose Use binary input signals to select one row from truth table

Library Control / Logical

Description The Combinatorial Logic block interprets its input signals as boolean values. It calculates a row number $r = 1 + \sum_i 2^i u_i$ where $u_i = 0$ if the i th input signal is 0, $u_i = 1$ otherwise. The output of the Combinatorial Logic block is the r th row of the truth table. For example, when using a truth table



$$\begin{bmatrix} 1.5 & 0 \\ 4 & 2.5 \\ 3 & 1.5 \\ 5.5 & 0 \end{bmatrix}$$

the output is:

Input	Output
[0 0]	[1.5 0]
[0 1]	[4 2.5]
[1 0]	[3 1.5]
[1 1]	[5.5 0]

Parameter **Truth table**

The truth table to calculate the output. The table must have 2^n rows where n is the width of the input signal. The number of columns corresponds to the width of the output signal.

Probe Signals

Input

The input signals.

Output

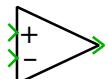
The output signals.

Comparator

Purpose Compare two input signals with minimal hysteresis

Library Control / Discontinuous

Description The Comparator compares two input signals. If the non-inverting input is greater than the inverting input, the output is 1. The output is set to 0 if the non-inverting input is less than the inverting one. The output does not change if both inputs are equal.



Probe Signals

Input	The input signals.
Output	The output signals.

Configurable Subsystem

Purpose	Provide subsystem with exchangeable implementations
Library	System
Description	A configurable subsystem is a subsystem that has multiple, exchangeable configurations. Each subsystem configuration has its own schematic diagram. All subsystem configurations of the configurable subsystem share the same input, output and electrical terminals. Once a port element has been added to one of the internal schematics it becomes available in all other internal schematics. By selecting Look under mask the schematic view of the configurable subsystem is opened. The schematic for each configuration can be accessed by the tabs on top of the schematic view. New configurations can be added and removed from the context menu of the tab bar, accessible by right-click. A double-click on a configuration tab allows for the corresponding configuration to be renamed.
Parameters	Configuration The name of the internal schematic that is used during simulation. Additional parameters for the Configurable Subsystem can be created by masking the block (see “Mask Parameters” (on page 51) for more details).
Probe Signals	Probe signals for the Configurable Subsystem can be created by masking the block (see “Mask Probe Signals” (on page 53) for more details).

Constant

Purpose Generate constant signal

Library Control / Sources

Description The Constant block outputs a constant signal.

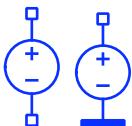


Parameter	Value
	The constant value. This parameter may either be a scalar or a vector defining the width of the component. The default value is 1.
Probe Signals	Output The constant signal.

Constant Heat Flow

Purpose	Generate constant heat flow
Library	Thermal
Description	The Constant Heat Flow generates a constant heat flow between the two thermal ports. The direction of a positive heat flow through the component is marked with an arrow. 
Parameter	Heat flow The magnitude of the heat flow, in watts (W). The default is 1.
Probe Signals	Heat flow The heat flow in watts (W).

Constant Temperature

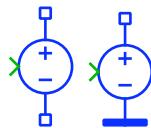
Purpose	Provide constant temperature
Library	Thermal
Description	The Constant Temperature generates a constant temperature difference between its two thermal connectors or between the thermal connector and the thermal reference. The temperature difference is considered positive if the terminal marked with a “+” has a higher temperature.
	
Parameter	Temperature The temperature difference generated by the component, in kelvin (K). The default is 0.
Probe Signals	Temperature The temperature difference in kelvin (K).

Controlled Heat Flow

Purpose	Generate variable heat flow
Library	Thermal
Description	The Controlled Heat Flow generates a variable heat flow between the two thermal ports. The direction of a positive heat flow through the component is marked with an arrow. The momentary heat flow is determined by the signal fed into the input of the component.
	
Probe Signals	Heat flow The heat flow in watts (W).

Controlled Temperature

Purpose	Provide variable temperature
Library	Thermal
Description	The Controlled Temperature generates a variable temperature difference between its two thermal connectors or between the thermal connector and the thermal reference. The temperature difference is considered positive if the terminal marked with a “+” has a higher temperature. The momentary temperature difference is determined by the signal fed into the input of the component.
Probe Signals	Temperature The temperature difference in kelvin (K).

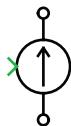


Current Source (Controlled)

Purpose Generate variable current

Library Electrical / Sources

Description The Controlled Current Source generates a variable current between its two electrical terminals. The direction of a positive current through the component is marked with an arrow. The momentary current is determined by the signal fed into the input of the component.



Note A current source may not be open-circuited or connected in series to an inductor or any other current source.

Probe Signals

Source current

The source current in amperes (A).

Source voltage

The voltage measured across the source, in volts (V).

Source power

The instantaneous output power of the source, in watts (W).

Current Source AC

Purpose Generate sinusoidal current

Library Electrical / Sources

Description The AC Current Source generates a sinusoidal current between its two electrical terminals. The direction of a positive current is marked with an arrow. The momentary current i is determined by the equation:



$$i = A \cdot \sin(\omega \cdot t + \varphi)$$

where t is the simulation time.

Note A current source may not be open-circuited or connected in series to an inductor or any other current source.

Parameters Each of the following parameters may either be a scalar or a vector corresponding to the implicit width of the component:

Amplitude

The amplitude A of the current, in amperes (A). The default is 1.

Frequency

The angular frequency ω , in s^{-1} . The default is $2\pi \cdot 50$ which corresponds to 50 Hz.

Phase

The phase shift φ , in radians. The default is 0.

Probe Signals

Source current

The source current in amperes (A).

Source voltage

The voltage measured across the source, in volts (V).

Source power

The instantaneous output power of the source, in watts (W).

Current Source DC

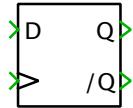
Purpose	Generate constant current
Library	Electrical / Sources
Description	The DC Current Source generates a constant current between its two electrical terminals. The direction of a positive current through the component is marked with an arrow. 
	Note A current source may not be open-circuited or connected in series to an inductor or any other current source.
Parameter	Current The magnitude of the constant current, in amperes (A). This parameter may either be a scalar or a vector defining the width of the component. The default value is 1.
Probe Signals	Source current The source current in amperes (A). Source voltage The voltage measured across the source, in volts (V). Source power The instantaneous output power of the source, in watts (W).

D Flip-flop

Purpose Implement edge-triggered flip-flop

Library Control / Logical

Description The D flip-flop sets its output Q to the value of its input D when an edge on the clock input is detected. The behavior is shown in the following truth table:



D	Clk	Q	/Q
0	0	No change	No change
0	1	No change	No change
1	0	No change	No change
1	1	No change	No change
0	Triggering edge	0	1
1	Triggering edge	1	0

The input D is latched, i.e. when a triggering edge in the clock signal is detected the value of D from the previous simulation step is used to set the output. In other words, D must be stable for at least one simulation step before the flip-flop is triggered by the clock signal.

Parameter **Trigger edge**
The direction of the edge on which the D input is read.

Initial state
The state of the flip-flop at simulation start.

Probe Signals

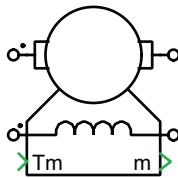
- D** The input signal D .
- Clk** The clock input signal.
- Q** The output signals Q .
- /Q** The output signals $/Q$.

DC Machine

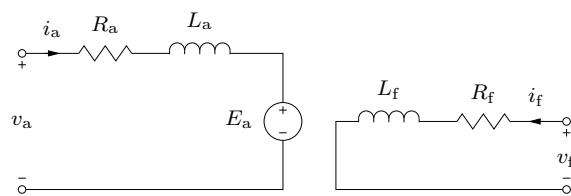
Purpose Simple model of DC machine

Library Electrical / Machines

Description The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. In the component icon, the positive poles of armature and field winding are marked with dots.



Electrical System



Electromagnetic torque:

$$T_e = L_{af} \cdot i_f \cdot i_a$$

Induced voltage of the armature winding:

$$E_a = L_{af} \cdot i_f \cdot \omega_m$$

Mechanical System

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

Parameters

Armature resistance

Armature winding resistance R_a in ohms (Ω).

Armature inductance

Armature winding inductance L_a in henries (H).

Field resistance

Field winding resistance R_f in ohms (Ω).

Inputs and Outputs

Armature inductance

Field winding inductance L_f in henries (H).

Field-armature mutual inductance

Field-armature mutual inductance L_{af} in henries (H).

Inertia

Combined rotor and load inertia J in Nms².

Friction coefficient

Viscous friction F in Nms.

Number of pole pairs

Number of pole pairs p .

Initial rotor speed

Initial mechanical speed $\omega_{m,0}$ in radians per second (s⁻¹).

Initial armature current

Initial current $i_{a,0}$ in the armature winding in amperes (A).

Initial field current

Initial current $i_{f,0}$ in the field winding in amperes (A).

Mechanical torque

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector "m" contains the following 2 signals:

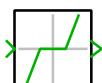
(1) Rotor speed

The rotational speed ω_m of the rotor in radians per second (s⁻¹).

(2) Electrical torque

The electrical torque T_e of the machine in Nm.

Dead Zone

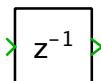
Purpose	Output zero while input signal is within dead zone limits
Library	Control / Discontinuous
Description	The Dead Zone block outputs zero while the input is within the limits of the dead zone. When the input signal is outside of the dead zone limits, the output signal equals the input signal minus the nearest dead zone limit.
	
Parameters	Lower dead zone limit The lower limit of the dead zone. Upper dead zone limit The upper limit of the dead zone.
Probe Signals	Input The block input signal. Output The block output signal.

Delay

Purpose Delay input signal by given number of samples

Library Control / Discrete

Description The Delay block delays the input signal by N sample periods.



Parameters

Delay order

The number of delay periods applied to the input signal.

Initial condition

The initial output value during the first delay period.

Sample time

The length of the sample period in sec. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Probe Signals

Input

The input signal.

Output

The delayed output signal.

Diode

Purpose Ideal diode with optional forward voltage and on-resistance

Library Electrical / Power Semiconductors

Description The Diode is a semiconductor device controlled only by the voltage across it and the current through the device. The Diode model is basically an ideal switch that closes if the voltage between anode and cathode becomes positive and opens again if the current through the component passes through zero. In addition to the ideal switch, a forward voltage and an on-resistance may be specified. These parameters may either be scalars or vectors corresponding to the implicit width of the component. If unsure set both values to 0.



Parameters The following parameters may either be scalars or vectors corresponding to the implicit width of the component:

Forward voltage

Additional dc voltage V_f in volts (V) between anode and cathode when the diode is conducting. The default is 0.

On-resistance

The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.

Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79). If no thermal description is given the losses are calculated based on the voltage drop $v_{on} = V_f + R_{on} \cdot i$.

Initial temperature

Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.

Note Under blocking conditions the diode voltage is *negative*. Hence you should define the turn-on and turn-off loss tables for negative voltages. See chapter “Diode Losses” (on page 93) for more information.

Probe Signals**Diode voltage**

The voltage measured between anode and cathode.

Diode current

The current through the diode flowing from anode to cathode.

Diode conductivity

Conduction state of the internal switch. The signal outputs 0 when the diode is blocking, and 1 when it is conducting.

Diode junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

Diode conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

Diode switching loss

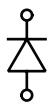
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

Diode with Reverse Recovery

Purpose Dynamic diode model with reverse recovery

Library Electrical / Power Semiconductors

Description This component is a behavioral model of a diode which reproduces the effect of reverse recovery. This effect can be observed when a forward biased diode is rapidly turned off. It takes some time until the excess charge stored in the diode during conduction is removed. During this time the diode represents a short circuit instead of an open circuit, and a negative current can flow through the diode. The diode finally turns off when the charge is swept out by the reverse current and lost by internal recombination.



Note

- Due to the small time-constant introduced by the turn-off transient a stiff solver is recommended for this device model.
- If multiple diodes are connected in series, the off-resistance may not be infinite.

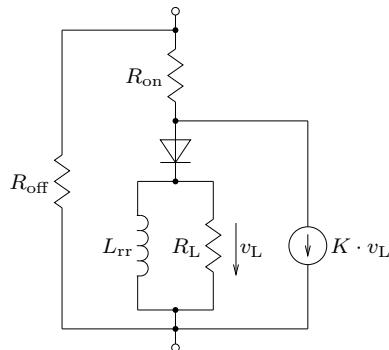
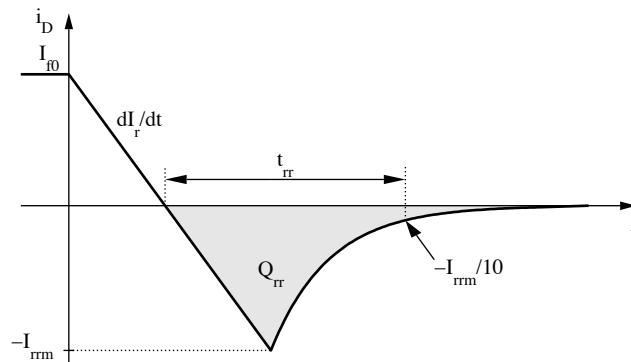
The following figure illustrates the relationship between the diode parameters and the turn-off current waveform. I_{f0} and dI_r/dt denote the continuous forward current and the rated turn-off current slope under test conditions. The turn-off time t_{rr} is defined as the period between the zero-crossing of the current and the instant when it becomes equal to 10% of the maximum reverse current I_{rrm} . The reverse recovery charge is denoted Q_{rr} . Only two out of the three parameters t_{rr} , I_{rrm} , and Q_{rr} need to be specified since they are linked geometrically. The remaining parameter should be set to 0. If all three parameters are given, Q_{rr} is ignored.

The equivalent circuit of the diode model is shown below. It is composed of a resistance, and inductance, and a controlled current source which is linearly dependent on the inductor voltage. The values of these internal elements are automatically calculated from the diode parameters.

Parameters

Forward voltage

Additional dc voltage V_f in volts (V) between anode and cathode when the diode is conducting. The default is 0.



On-resistance

The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.

Off-resistance

The resistance R_{off} of the blocking device, in ohms (Ω). The default is inf . If diodes are connected in series, the off-resistance must have a large finite value.

Continuous forward current

The continuous forward current I_{f0} under test conditions.

Current slope at turn-off

The turn-off current slope dI_r/dt under test conditions.

Reverse recovery time

The turn-off time t_{rr} under test conditions.

Peak recovery current

The absolute peak value of the reverse current I_{rrm} under test conditions.

Reverse recovery charge

The reverse recovery charge Q_{rr} under test conditions. If both t_{rr} and I_{rrm} are specified, this parameter is ignored.

Lrr

This inductance acts as a probe measuring the di/dt . It should be set to a very small value. The default is 10e-10.

Probe Signals**Diode voltage**

The voltage measured between anode and cathode.

Diode current

The current through the diode flowing from anode to cathode.

Diode conductivity

Conduction state of the internal switch. The signal outputs 0 when the diode is blocking, and 1 when it is conducting.

References

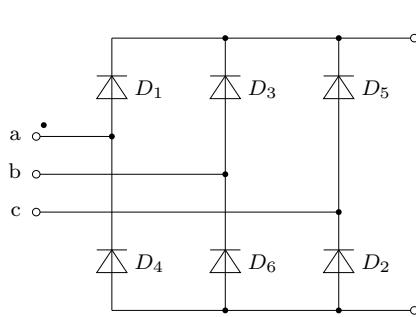
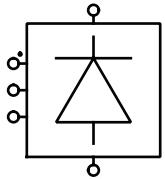
- A. Courty, "MAST power diode and thyristor models including automatic parameter extraction", SABER User Group Meeting Brighton, UK, Sept. 1995.

Diode Rectifier (3ph)

Purpose 3-phase diode rectifier

Library Electrical / Converters

Description Implements a three-phase rectifier based on the Diode model (see page 232). The electrical circuit for the rectifier is given below:



Parameters For a description of the parameters see the documentation of the Diode (on page 232).

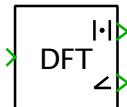
Probe Signals The Diode Rectifier provides six probe signals, each a vector containing the appropriate quantities of the six individual diodes: voltage, current, conductivity, conduction loss and switching loss. The vector elements are ordered according to the natural sequence of commutation.

Discrete Fourier Transform

Purpose Perform discrete Fourier transform on input signal

Library Control / Discrete

Description This block calculates the discrete Fourier transform of a periodic input signal based on discrete samples. The sample time, the number of samples and the harmonic order(s) can be specified. The fundamental frequency f_1 of the running window is:



$$f_1 = \frac{1}{\text{sample time} \times \text{number of samples}}.$$

The outputs of the block are the magnitude and phase angle of the specified harmonics.

If you specify more than one harmonic, the outputs will be vectors with the corresponding width. Alternatively you can specify a single harmonic and feed a vector signal into the block.

Note In Simulink this block is only available for MATLAB 7.0 or newer.

Parameters

Sample time

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Number of samples

The number of samples used to calculate the Fourier transform.

Harmonic orders n

A scalar or vector specifying the harmonic component(s) you are interested in. Enter 0 for the dc component, 1 for the fundamental component, etc.

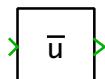
This parameter should be scalar if the input signal is a vector.

Discrete Mean Value

Purpose Calculate running mean value of input signal

Library Control / Discrete

Description This block calculates the running mean of the input signal based on discrete samples. The sample time and the number of samples can be specified. The block is implemented with a shift register. The output of the block is the sum of all register values divided by the number of samples.



Note In Simulink this block is only available for MATLAB 7.0 or newer.

Parameters

Initial condition

The initial condition describes the input signal before simulation start. If the input is a scalar signal the parameter can either be a scalar or a column vector. The number of elements in the vector must match the value of the parameter **Number of samples** - 1. If input and output are vectorized signals a matrix can be used. The number of rows must be 1 or match the number of input signals. The default value of this parameter is 0.

Sample time

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Number of samples

The number of samples used to calculate the mean value.

Discrete RMS Value

Purpose	Calculate root mean square (RMS) value of input signal
Library	Control / Discrete
Description	This block calculates the RMS value of a periodic input signal based on discrete samples. The sample time and the number of samples can be specified. The fundamental frequency f of the running window is



$$f = \frac{1}{\text{sample time} \times \text{number of samples}}.$$

The Discrete RMS Value block is implemented with the Discrete Mean Value block (see page 239).

Note In Simulink this block is only available for MATLAB 7.0 or newer.

Parameters

Initial condition

The initial condition describes the input signal before simulation start. If the input is a scalar signal the parameter can either be a scalar or a column vector. The number of elements in the vector must match the value of the parameter **Number of samples** - 1. If input and output are vectorized signals a matrix can be used. The number of rows must be 1 or match the number of input signals. The default value of this parameter is 0.

Sample time

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Number of samples

The number of samples used to calculate the RMS value.

Discrete Total Harmonic Distortion

Purpose	Calculate total harmonic distortion (THD) of input signal
Library	Control / Discrete
Description	This block calculates the total harmonic distortion of a periodic input signal based on discrete samples. The sample time and the number of samples can be specified. The THD is defined as:



$$\text{THD} = \sqrt{\frac{\sum_{\nu \geq 2} U_\nu^2}{U_1^2}} = \sqrt{\frac{U_{\text{rms}}^2 - U_0^2 - U_1^2}{U_1^2}}$$

where U_ν is the RMS value of the ν th harmonic of the input signal and U_{rms} is its overall RMS value. The fundamental frequency f_1 of the running window is

$$f_1 = \frac{1}{\text{sample time} \times \text{number of samples}}.$$

Note In Simulink this block is only available for MATLAB 7.0 or newer.

Parameters

Sample time

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Number of samples

The number of samples used to calculate the THD.

Discrete Transfer Function

Purpose Model discrete system as transfer function

Library Control / Discrete

Description The Discrete Transfer Function models a discrete time-invariant system that is expressed in the z -domain:

$$\frac{1}{z+1}$$

$$\frac{Y(z)}{U(z)} = \frac{n_n z^n + \dots + n_1 z + n_0}{d_n z^n + \dots + d_1 z + d_0}$$

Parameters

Numerator coefficients

A vector of the z term coefficients $[n_n \dots n_1, n_0]$ for the numerator, written in descending order of powers of z . For example, the numerator $z^3 + 2z$ would be entered as $[1, 0, 2, 0]$.

The output of the Transfer Function is vectorizable by entering a matrix for the numerator.

Denominator coefficients

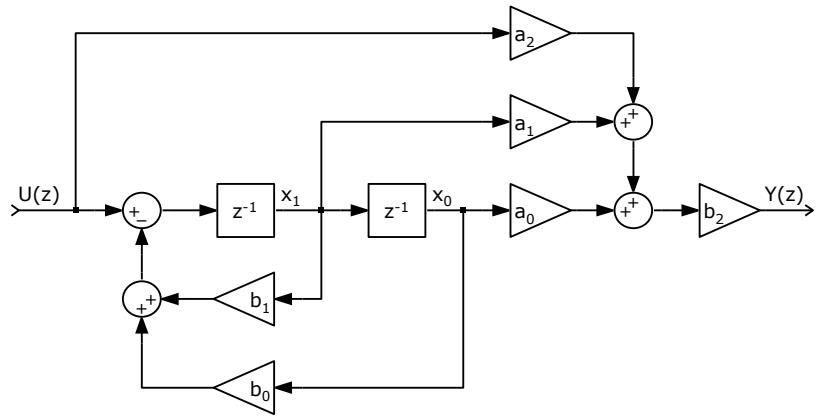
A vector of the z term coefficients $[d_n \dots d_1, d_0]$ for the denominator, written in descending order of powers of z .

Note The order of the denominator (highest power of z) must be greater than or equal to the order of the numerator.

Initial condition

The initial condition vector of the internal states of the Transfer Function in the form $[x_n \dots x_1, x_0]$. The initial conditions must be specified for the controller normal form, depicted below for the transfer function:

$$\frac{Y(z)}{U(z)} = \frac{n_2 z^2 + n_1 z + n_0}{d_2 z^2 + d_1 z + d_0}$$



where

$$b_i = \frac{d_i}{d_n} \quad \text{for } i < n$$

$$b_n = \frac{1}{d_n}$$

$$a_i = n_i - \frac{n_n d_i}{d_n} \quad \text{for } i < n$$

$$a_n = n_n$$

For the normalized transfer function (with $n_n = 0$ and $d_n = 1$) this simplifies to $b_i = d_i$ and $a_i = n_i$.

Probe Signals

Input

The input signal.

Output

The output signal.

DLL

Purpose	Interface with externally generated dynamic-link library
Library	Control / Functions & Tables
Description	<p>The DLL block allows you to load a user generated DLL. The DLL may be implemented in any programming language on any development environment that the system platform supports. For convenience, all code snippets in this description are given in C.</p> <p>The DLL must supply two functions, <code>plecsSetSizes</code> and <code>plecsOutput</code>. Additionally it may implement the functions <code>plecsStart</code> and <code>plecsTerminate</code>.</p> <p>The complete DLL interface is described in the file <code>include/plecs/DllHeader.h</code> in the PLECS installation directory. This file should be included when implementing the DLL.</p>
 DLL	

void plecsSetSizes(struct SimulationSizes* aSizes)

This function is called once during the initialization of a new simulation.

The parameter struct `SimulationSizes` is defined as follows:

```
struct SimulationSizes {  
    int numInputs;  
    int numOutputs;  
    int numStates;  
    int numParameters;  
};
```

In the implementation of `plecsSetSizes` the DLL has to set all the fields of the supplied structure.

numInputs

The width of the input signal that the DLL expects. The length of the input array in the `SimulationState` struct is set to this value.

numOutputs

The number of outputs that the DLL generates. The width of the output signal of the DLL block and the length of the output array in the `SimulationState` struct is set to this value.

numStates

The number of discrete states that the DLL uses. The length of the `states` array in the `SimulationState` struct is set to this value.

numParameters

The length of the parameter vector that the DLL expects. A vector with `numParameters` elements must be supplied in the `Parameters` field of the component parameters of the DLL block. The parameters are passed in the `parameters` array in the `SimulationState` struct.

void plecsOutput(struct SimulationState* aState)

This function is called whenever the simulation time reaches a multiple of the *Sample time* of the DLL block.

The parameter struct `SimulationState` is defined as follows:

```
struct SimulationState {
    const double* const inputs;
    double* const outputs;
    double* const states;
    const double* const parameters;
    const double time;
    const char* errorMessage;
    void* userData;
};
```

inputs

The values of the input signal for the current simulation step. The values are read-only. The array length is the value of the `numInputs` field that was set in the `plecsSetSizes` method.

outputs

The output values for the current simulation step. These values must be set by the DLL. The array length is the value of the `numOutputs` field that was set in the `plecsSetSizes` method.

states

The values of the discrete states of the DLL. These values can be read and modified by the DLL. The array length is the value of the `numStates` field that was set in the `plecsSetSizes` method.

parameters

The values of the parameters that were set in the `Parameters` field in the component parameters of the DLL block. The values are read-only. The

array length is the value of the numParameters field that was set in the plecsSetSizes method.

time

The simulation time of the current simulation step.

errorMessage

The DLL may indicate an error condition by setting an error message. The simulation will be stopped after the current simulation step.

userData

A pointer to pass data from one call into the DLL to another. The value is not touched by PLECS.

void plecsStart(struct SimulationState* aState)

This function is called once at the start of a new simulation. It may be used to set initial outputs or states, initialize internal data structures, acquire resources etc.

The values of the inputs array in the SimulationState struct are undefined in the plecsStart function.

void plecsTerminate(struct SimulationState* aState)

This function is called once when the simulation is finished. It may be used to free any resources that were acquired by the DLL.

Note The processor architecture of the DLL must match the processor architecture of PLECS. If, for example, a 32-bit version of PLECS is used on a 64-bit Windows machine, a 32-bit DLL must be built. The processor architecture used by PLECS is displayed in the *About PLECS ...* dialog, accessible from the *File* menu.

Parameters**Filename**

The filename of the DLL. If the filename does not contain the full path of the DLL, the DLL is searched relative to the directory containing the model file. If no DLL is found with the given filename, a platform specific ending will be attached to the filename and the lookup is retried. The endings and search order are listed in the table below.

Platform	Filename search order
Windows 32-bit	<i>filename</i> , <i>filename.dll</i> , <i>filename_32.dll</i>
Windows 64-bit	<i>filename</i> , <i>filename.dll</i> , <i>filename_64.dll</i>
Mac OS X 32-bit	<i>filename</i> , <i>filename.dylib</i> , <i>filename_32.dylib</i>
Mac OS X 64-bit	<i>filename</i> , <i>filename.dylib</i> , <i>filename_64.dylib</i>
Linux 32-bit	<i>filename</i> , <i>filename.so</i> , <i>filename_32.so</i>
Linux 64-bit	<i>filename</i> , <i>filename.so</i> , <i>filename_64.so</i>

Sample time

The simulation time interval between two successive calls to the output function of the DLL. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Output delay

Allows you to delay the output in each simulation step. This is useful when modeling, for example, a DSP that needs a certain processing time to calculate the new outputs. The output delay must be smaller than the sample time. If the output delay is a positive number, the DLL block has no direct feedthrough, i.e. its outputs can be fed back to its inputs without causing an algebraic loop.

Parameters

Array of parameter values to pass to the DLL. The length of the array must match the value of the numParameters field that the DLL sets in the plecsSetSizes method.

Probe Signals

Input

The input signal.

Output

The output signal.

Double Switch

Purpose	Changeover switch with two positions
Library	Electrical / Switches
Description	This changeover switch provides an ideal short or open circuit. If the input signal is zero the switch is in the upper position. For all other values the switch is in the lower position.
	
Parameter	Initial position Initial position of the switch. The switch is initially in the upper position if the parameter evaluates to zero. For all other values it is in the lower position. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.
Probe Signals	Switch position State of the internal switches. The signal outputs 0 if the switch is in the upper position, and 1 if it is in the lower position.

Edge Detection

Purpose	Detect edges of pulse signal in given direction
Library	Control / Logical
Description	<p>The output of the edge detection block changes to 1 if an edge is detected on the input signal. It returns to 0 in the following simulation step.</p> <p>The block allows you to detect the following edges:</p> <ul style="list-style-type: none">rising The output is set to 1 when the input changes from 0 to a positive value.falling The output is set to 1 when the input changes from a positive value to 0.either The output is set to 1 when the input changes from 0 to a positive value or vice versa.
Parameter	Edge direction The direction of the edges to detect, as described above.
Probe Signals	Input The input signal. Output The output signal.

Electrical Ground

Purpose Connect to common electrical ground

Library System

Description The ground block implements an electrical connection to the ground.



Note PLECS does not require a circuit to be grounded at one or more points. The ground block just provides a convenient means to connect distant points to a common potential.

Electrical Label

Purpose	Connect electrical potentials by name
Library	System
Description	The Electrical Label block provides an electrical connection between other Electrical Label blocks with identical tag names within the same scope.
 tag	
Parameter	Tag name: The tag name is used to find other matching Electrical Label blocks to connect to. Scope: The scope specifies the search depth for the matching Electrical Label blocks. Using the value Global the complete PLECS circuit is searched. When set to Schematic only the schematic containing the Electrical Label block is searched. The setting Masked Subsystem causes a lookup within the hierarchy of the masked subcircuit in which the block is contained. If the block is not contained in a masked subsystem a global lookup is done.

Electrical Port

Purpose	Add electrical connector to subsystem
Library	System
Description	<p>Electrical ports are used to establish electrical connections between a schematic and the subschematic of a subsystem (see page 402). If you copy an Electrical Port block into the schematic of a subsystem a terminal will be created on the subsystem block. The name of the port block will appear as the terminal label. If you choose to hide the block name by unselecting the show button in the dialog box the terminal label will also disappear.</p> <p>Terminals can be moved around the edges of the subsystem by holding down the Shift key while dragging the terminal with the left mouse button or by using the middle mouse button.</p>
	
Note	Since you cannot make electrical connections in Simulink, PLECS does not permit to place electrical port blocks into top-level schematics.
Parameter	Width The width of the connected wire. The default auto means that the width is inherited from connected components.

Flux Rate Meter

Purpose

Output the measured rate-of-change of magnetic flux

Library

Magnetic

Description



The Flux Rate Meter measures the rate-of-change $\dot{\Phi}$ of the magnetic flux through the component and provides it as a signal at the output. The direction of a positive flux is indicated with a small arrow in the component symbol. The output signal can be made accessible in Simulink with an Output block (see page 387) or by dragging the component into the dialog box of a Probe block.

The magnetic flux Φ cannot be measured directly in the circuit. However, most permeance components provide the magnetic flux as a probe signal.

Note The Flux Rate Meter is ideal, i.e. it has infinite internal permeance. Hence, if multiple Flux Rate Meters are connected in parallel the flux through an individual meter is undefined. This produces a run-time error.

Probe Signals

Flux rate

The rate-of-change $\dot{\Phi}$ of magnetic flux flowing through the component, in Wb/s or V.

Fourier Series

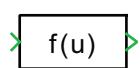
Purpose	Synthesize periodic output signal from Fourier coefficients
Library	Control / Functions & Tables
Description	The Fourier Series block calculates the series  $y = \frac{a_0}{2} + \sum_n a_n \cdot \cos(nx) + b_n \cdot \sin(nx)$ as a function of the input signal x .
Parameter	Fourier coefficients The coefficients a_0 , a_n , and b_n of the fourier series. The vectors a_n and b_n must have the same length.
Probe Signals	Input The input signal. Output The output signal.

Function

Purpose Apply arbitrary arithmetic expression to scalar or vectorized input signal

Library Control / Functions & Tables

Description The Function block applies an arithmetic expression specified in C language syntax to its input. The input may be a scalar or vectorized continuous signal, the output is always a scalar continuous signal. The expression may consist of one or more of the following components:

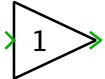


- u — the input of the block. If the input is vectorized, $u(i)$ or $u[i]$ represents the i th element of the vector. To access the first element, enter $u(1)$, $u[1]$, or u alone.
- Brackets
- Numeric constants, including π
- Arithmetic operators (+ - * / ^)
- Relational operators (== != > < >= <=)
- Logical operators (&& || !)
- Mathematical functions — abs , acos , asin , atan , atan2 , cos , cosh , exp , log , log10 , max , min , mod , pow , sgn , sin , sinh , sqrt , tan , and tanh .
- Workspace variables

Parameter **Expression**
The expression applied to the input signal, in C language syntax.

Probe Signals **Input**
The input signal.
Output
The output signal.

Gain

Purpose	Multiply input signal by constant
Library	Control / Math
Description	The Gain block multiplies the input signal with the gain value. The multiplication can either be an element-wise ($K \cdot u$) or a matrix multiplication ($K * u$). 
Parameters	Gain The gain value to multiply with the input signal. For element-wise multiplication the gain value can be a scalar or a vector matching the width of the input signal. For matrix multiplication the the gain value can be a scalar, a vector or a matrix which has as many columns as the width of the input signal. Multiplication Specifies whether element-wise ($K \cdot u$) or matrix multiplication ($K * u$) should be used.
Probe Signals	Input The input signal. Output The output signal.

GTO

Purpose	Ideal GTO with optional forward voltage and on-resistance
Library	Electrical / Power Semiconductors
Description	The Gate Turn Off Thyristor can also be switched off via the gate. Like a normal thyristor it closes if the voltage between anode and cathode is positive and a positive gate signal is applied. It opens if the current passes through zero or if the gate signal becomes negative.
	
Parameters	The following parameters may either be scalars or vectors corresponding to the implicit width of the component:
	Forward voltage Additional dc voltage V_f in volts (V) between anode and cathode when the GTO is conducting. The default is 0.
	On-resistance The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.
	Initial conductivity Initial conduction state of the GTO. The GTO is initially blocking if the parameter evaluates to zero, otherwise it is conducting.
	Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79). If no thermal description is given the losses are calculated based on the voltage drop $v_{on} = V_f + R_{on} \cdot i$.
	Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.
Probe Signals	GTO voltage The voltage measured between anode and cathode.
	GTO current The current through the GTO flowing from anode to cathode.
	GTO gate signal The gate input signal of the GTO.

GTO conductivity

Conduction state of the internal switch. The signal outputs 0 when the GTO is blocking, and 1 when it is conducting.

GTO junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

GTO conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

GTO switching loss

Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

GTO (Reverse Conducting)

Purpose	Ideal GTO with ideal anti-parallel diode
Library	Electrical / Power Semiconductors
Description	This model of a Gate Turn Off Thyristor has an integrated anti-parallel diode. The diode is usually included in power GTO packages.
	
Parameters	The following parameters may either be scalars or vectors corresponding to the implicit width of the component:
	Initial conductivity
	Initial conduction state of the GTO. The GTO is initially blocking if the parameter evaluates to zero, otherwise it is conducting.
	Thermal description
	Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapters “Thermal Modeling” (on page 79) and “Losses of Semiconductor Switch with Diode” (on page 94) for more information.
	Initial temperature
	Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.
Probe Signals	Device voltage
	The voltage measured between anode and cathode.
	Device current
	The current through the device flowing from anode to cathode.
	Device gate signal
	The gate input signal of the device.
	Device conductivity
	Conduction state of the internal switch. The signal outputs 0 when the device is blocking, and 1 when it is conducting.
	Device junction temperature
	Temperature of the first thermal capacitor in the equivalent Cauer network.

Device conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

Device switching loss

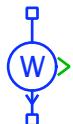
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

Heat Flow Meter

Purpose Output measured heat flow as signal

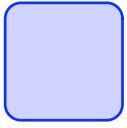
Library Thermal

Description The Heat Flow Meter measures the heat flow through the component and provides it as a signal at the output. The direction of a positive heat flow is indicated by the small arrow at one of the thermal ports. The output signal can be made accessible in Simulink with a Output block (see page 387) or by dragging the component into the dialog box of a Probe block.



Probe Signals **Measured heat flow**
The measured heat flow in watts (W).

Heat Sink

Purpose	Isotherm environment for placing components
Library	Thermal
Description	<p>The Heat Sink absorbs the thermal losses dissipated by the components within its boundaries. At the same time it defines an isotherm environment and propagates its temperature to the components which it encloses. To change the size of a Heat Sink, select it, then drag one of its selection handles.</p> <p></p> <p>With the parameter Number of terminals you can add and remove thermal connectors to the heat sink in order to connect it to an external thermal network. The connectors can be dragged along the edge of the heat sink with the mouse by holding down the Shift key or using the middle mouse button. In order to remove a thermal connector, disconnect it, then reduce the Number of terminals. PLECS will not allow you to remove connected terminals.</p> <p>For additional information see chapter “Thermal Modeling” (on page 79).</p>
Parameters	<p>Number of terminals This parameter allows you to change the number of external thermal connectors of a heat sink. The default is 0.</p> <p>Thermal capacitance The value of the internal thermal capacitance, in J/K. The default is 1. If the capacitance is set to zero the heat sink must be connected to an external thermal capacitance or to a fixed temperature.</p> <p>Initial temperature The initial temperature difference between the heat sink and the thermal reference at simulation start, in kelvin (K). The default is 0.</p>
Probe Signals	<p>Temperature The temperature difference between the heat sink and the thermal reference, in kelvin (K).</p>

Hit Crossing

Purpose

Detect when signal reaches or crosses given value

Library

Control / Discontinuous

Description



The Hit Crossing block detects when the input signal reaches or crosses a value in the specified direction. If a variable-step solver is used, a simulation step is forced at the time when the crossing occurs. The output signal is 1 for one simulation time step when a crossing occurs, 0 otherwise.

Parameters

Hit crossing offset

The offset that the input signal has to reach or cross.

Hit crossing direction

The value **rising** causes hit crossings only when the input signal is rising. If **falling** is chosen, only hit crossings for a falling input signal are detected. The setting **either** causes hit crossing for rising and falling signals to be detected.

Threshold

The threshold value used for the switch criteria.

Show output port

The output terminal of the Hit Crossing block will be hidden if the parameter is set to **off**. This setting only makes sense to force a simulation step while using a variable-step solver.

Probe Signals

Input

The block input signal.

Crossing signal

Outputs 1 for one simulation time step when a crossing occurs, 0 otherwise. This probe signal is identical to the output signal.

Hysteretic Core

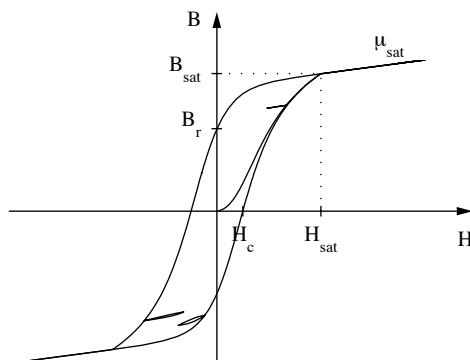
Purpose Magnetic core element with static hysteresis

Library Magnetic

Description This component models a segment of a magnetic core. It establishes a non-linear relationship between the magnetic field strength H and the flux density B . The hysteresis characteristics is based on a Preisach model with a Lorentzian distribution function.



The figure below shows a fully excited major hysteresis curve with some minor reversal loops. The major curve is defined by the saturation point (H_{sat} , B_{sat}), the coercitive field strength H_c , the remanence flux density B_r and the saturated permeability μ_{sat} .



Parameters

Cross-sectional area

Cross-sectional area A of the flux path, in m^2 .

Length of flux path

Length l of the flux path, in m.

Coercitive field strength

Coercitive field strength H_c for $B = 0$, in A/m.

Remanence flux density

Remanence flux density B_r for $H = 0$, in teslas (T).

Saturation field strength

Field strength H_{sat} at the saturation point, in A/m.

Saturation flux density

Flux density B_{sat} at the saturation point, in teslas (T).

Saturated rel. permeability

Relative permeability $\mu_{r,\text{sat}} = \mu_{\text{sat}}/\mu_0$ of the core material for $H > H_{\text{sat}}$.

Probe Signals**MMF**

The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).

Flux

The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.

Field strength

The magnetic field strength H in the core element, in A/m.

Flux density

The magnetic flux density B in the core element, in teslas (T).

Loss energy

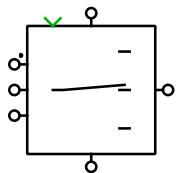
The energy dissipated in the core, in joules (J). An energy pulse is generated each time a minor or major hysteresis loop is closed.

Ideal 3-Level Converter (3ph)

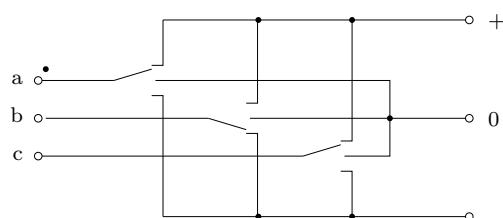
Purpose Switch-based 3-phase 3-level converter

Library Electrical / Converters

Description Implements a three-phase three-level converter with ideal switches. The converter is modeled using the Triple Switch component (see page 455). The gate input is a vector of three signals – one per leg. The phase output is connected to the positive, neutral, and negative dc level according to the sign of the corresponding gate signal.



The electrical circuit for the converter is shown below:



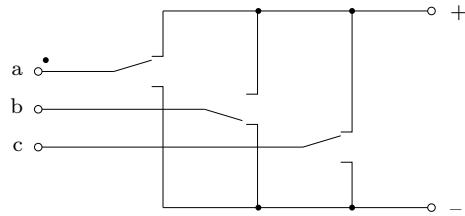
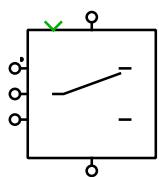
Ideal Converter (3ph)

Purpose Switch-based 3-phase converter

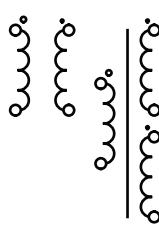
Library Electrical / Converters

Description Implements a three-phase two-level converter with ideal bi-positional switches. The converter is modeled using the Double Switch component (see page 248). The gate input is a vector of three signals – one per leg. The phase output is connected to the positive dc level upon a positive gate signal, and else to the negative dc level.

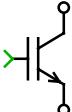
The electrical circuit for the converter is shown below:



Ideal Transformer

Purpose	Ideally coupled windings without inductance
Library	Electrical / Transformers
Description	This component represents a transformer with two or more ideally coupled windings. At all windings w , the voltage v_w across the winding divided by the corresponding number of turns n_w is the same:
	$\frac{v_1}{n_1} = \frac{v_2}{n_2} = \frac{v_3}{n_3} = \dots$
	The currents i_w of all windings multiplied with the corresponding number of turns add up to zero: $0 = i_1 \cdot n_1 + i_2 \cdot n_2 + i_3 \cdot n_3 + \dots$
	In the transformer symbol, the first primary side winding is marked with a little circle. The orientation of the other windings is indicated by a dot. To change the orientation of a specific winding w make the corresponding number of turns n_w negative.
Parameters	<p>Number of windings A two-element vector $[w_1 \ w_2]$ containing the number of windings on the primary side w_1 and on the secondary side w_2. The default is $[1 \ 1]$, which represents a two-winding transformer with opposite windings.</p> <p>Number of turns A row vector specifying the number of turns for each winding. The vector length must match the total number of primary and secondary side windings. First, all primary side windings are specified, followed by the specifications for all secondary side windings.</p>

IGBT

Purpose	Ideal IGBT with optional forward voltage and on-resistance
Library	Electrical / Power Semiconductors
Description	The Insulated Gate Bipolar Transistor is a semiconductor switch that is controlled via the external gate. It conducts a current from collector to emitter only if the gate signal is not zero.
	
Parameters	The following parameters may either be scalars or vectors corresponding to the implicit width of the component:
	Forward voltage
	Additional dc voltage V_f in volts (V) between collector and emitter when the IGBT is conducting. The default is 0.
	On-resistance
	The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.
	Initial conductivity
	Initial conduction state of the IGBT. The IGBT is initially blocking if the parameter evaluates to zero, otherwise it is conducting.
	Thermal description
	Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79). If no thermal description is given the losses are calculated based on the voltage drop $v_{on} = V_f + R_{on} \cdot i$.
	Initial temperature
	Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.
Probe Signals	
	IGBT voltage
	The voltage measured between collector and emitter.
	IGBT current
	The current through the IGBT flowing from collector to emitter.
	IGBT gate signal
	The gate input signal of the IGBT.

IGBT conductivity

Conduction state of the internal switch. The signal outputs 0 when the IGBT is blocking, and 1 when it is conducting.

IGBT junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

IGBT conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

IGBT switching loss

Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

IGBT 3-Level Converter (3ph)

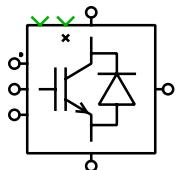
Purpose

3-phase 3-level neutral-point clamped IGBT converter

Library

Electrical / Converters

Description

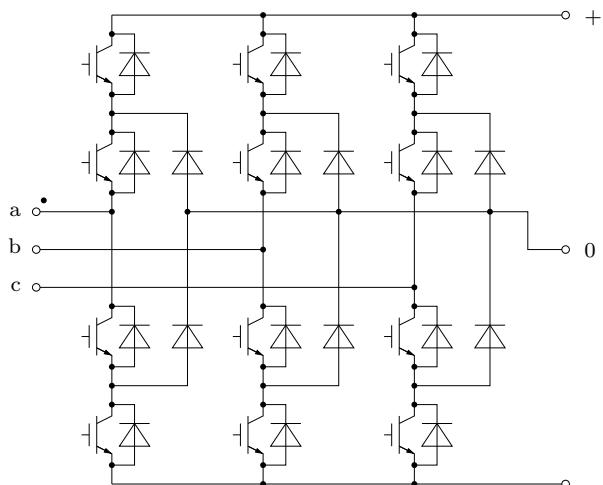


Implements a three-phase three-level IGBT converter with neutral point clamping. The gate input is a vector of three signals – one per leg. The top-most IGBT, connected to the positive dc level, is turned on if the corresponding gate signal is ≥ 1 , and the second IGBT if the signal is ≥ 0 . The third IGBT is turned on for signals ≤ 0 and the lowest one for signals ≤ -1 . Gate signal values of 1, 0 and -1 connect the phase output to the positive, neutral and negative dc level. By applying a non-zero signal at the inhibit input marked with "x" you can turn off all IGBTs.

You can choose between two different converter models:

- The basic **IGBT 3-Level Converter** is modeled using the component IGBT with Diode (see page 275). No parameters can be entered.
- The **IGBT 3-Level Converter with Parasitics** is based on individual IGBT (see page 269) and Diode (see page 232) components. In this model you may specify forward voltages and on-resistances separately for the IGBTs and the diodes.

The electrical circuit for the converter is shown below:



Parameters

For a description of the parameters see the documentation of the IGBT with Diode (on page 275), the IGBT (on page 269) and the Diode (on page 232).

Probe Signals

The three-level IGBT converters provide 36 probe signals grouped by leg. Each signal is a vector containing the appropriate quantities of the individual devices: voltage, current, conductivity, conduction loss and switching loss. The vector elements are ordered top-to-bottom.

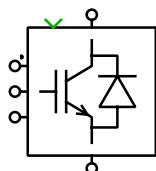
For the **IGBT 3-Level Converter with Parasitics** the diode probe signal vectors are in the order: anti-parallel diodes (top-to-bottom), clamping diodes (top-to-bottom).

IGBT Converter (3ph)

Purpose 3-phase IGBT converter

Library Electrical / Converters

Description

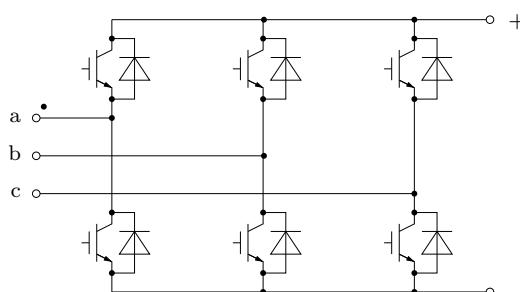


Implements a three-phase two-level IGBT converter with reverse diodes. The gate input is a vector of three signals – one per leg. The upper IGBT, connected to the positive dc level, is on if the corresponding gate signal is positive. The lower IGBT is on if the gate signal is negative. If the gate signal is zero both IGBTs in the leg are switched off.

You can choose between two different converter models:

- The basic **IGBT Converter** is modeled using the component IGBT with Diode (see page 275). PLECS needs only six internal switches to represent this converter, so the simulation is faster compared to the detailed converter. No electrical parameters can be entered, but the thermal losses may be specified.
- The **IGBT Converter with Parasitics** is based on individual IGBT (see page 269) and Diode (see page 232) components. In this model you may specify all electrical and thermal parameters separately for the IGBTs and the diodes.

The electrical circuit for the converter is shown below:



Parameters

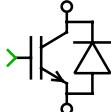
For a description of the parameters see the documentation of the IGBT with Diode (on page 275), the IGBT (on page 269) and the Diode (on page 232).

Probe Signals

The two-level IGBT converters provide six or twelve probe signals, each a vector containing the appropriate quantities of the individual devices: voltage,

current, conductivity, conduction loss and switching loss. The vector elements are ordered top-to-bottom, left-to-right: a+, a-, b+, b-, c+, c-.

IGBT with Diode

Purpose	Ideal IGBT with ideal anti-parallel diode
Library	Electrical / Power Semiconductors
Description	This model of an Insulated Gate Bipolar Transistor has an integrated anti-parallel diode. The diode is usually required in AC applications such as voltage source inverters.
	
Parameters	<p>Initial conductivity Initial conduction state of the device. The device is initially blocking if the parameter evaluates to zero, otherwise it is conducting. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.</p> <p>Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapters “Thermal Modeling” (on page 79) and “Losses of Semiconductor Switch with Diode” (on page 94).</p> <p>Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start. This parameter may either be a scalar or a vector corresponding to the implicit width of the component.</p>
Probe Signals	<p>Device voltage The voltage measured between collector/cathode and emitter/anode. The device voltage can never be negative.</p> <p>Device current The current through the device. The current is positive if it flows through the IGBT from collector to emitter and negative if it flows through the diode from anode to cathode.</p> <p>Device gate signal The gate input signal of the device.</p> <p>Device conductivity Conduction state of the internal switch. The signal outputs 0 when the device is blocking, and 1 when it is conducting.</p>

Device junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

Device conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

Device switching loss

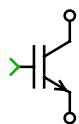
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

IGBT with Limited di/dt

Purpose Dynamic IGBT model with finite current slopes during turn-on and turn-off

Library Electrical / Power Semiconductors

Description



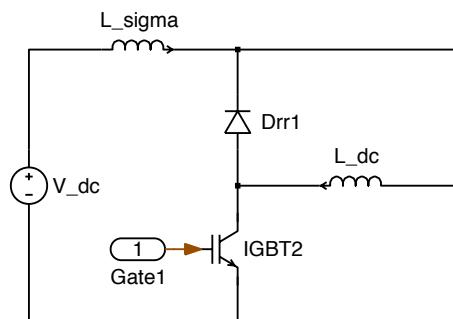
In contrast to the ideal IGBT model (see page 269) that switches instantaneously, this model includes collector current transients during switching. Thanks to the continuous current decay during turn-off, stray inductances may be connected in series with the device. In converter applications, the di/dt limitation during turn-on determines the magnitude of the reverse recovery effect in the free-wheeling diodes.

This IGBT model is used to simulate overvoltages produced by parasitic inductances in the circuit. Since the voltage and current transient waveforms are simplified, the model is not suited for the simulation of switching losses.

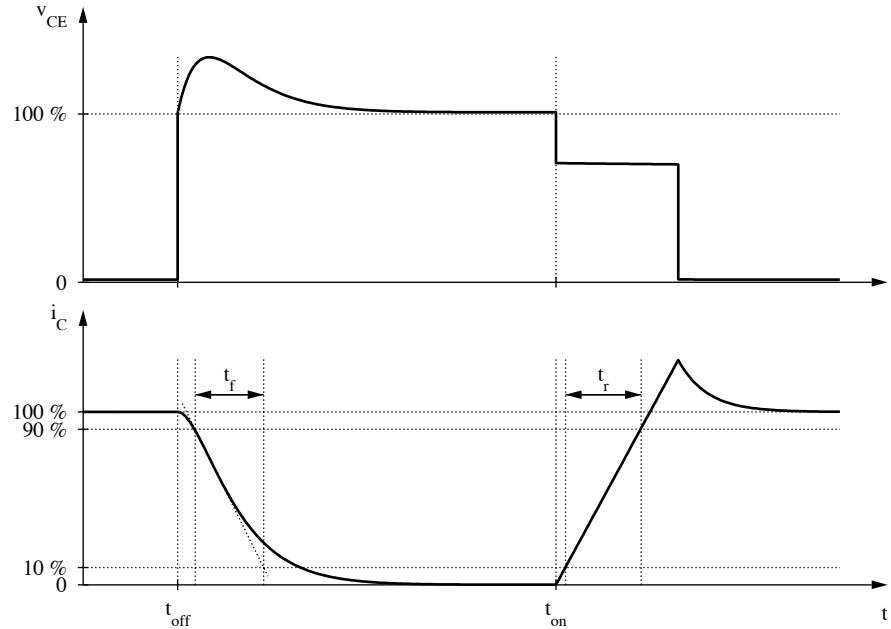
Note

- Due to the small time-constants introduced by the turn-on and turn-off transients a stiff solver is recommended for this device model.
- If multiple IGBTs are connected in series, the off-resistance may not be infinite.

The behavior of this IGBT model is demonstrated with the following test circuit. The free-wheeling diode for the inductive load is modeled with reverse recovery (see page 234).



The diagram below shows the collector current $i_C(t)$ of the IGBT and the resulting collector-emitter voltage $v_{CE}(t)$ during switching:



Collector current and collector-emitter voltage

At $t = t_{\text{off}}$ the gate signal becomes zero, and the device current i_C begins to fall. The current slope follows an aperiodic oscillation

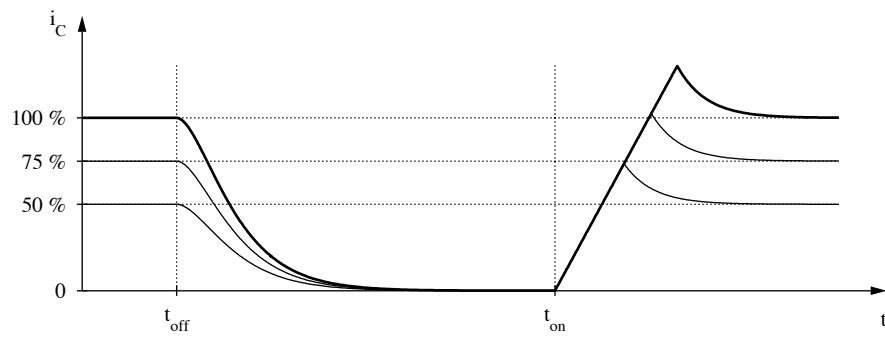
$$i_C(t) = i_C(t_{\text{off}}) \left[e^{-\frac{2.4(t-t_{\text{off}})}{t_f}} \left(1 + \frac{2.4(t-t_{\text{off}})}{t_f} \right) \right]$$

where t_f is the fall time specified in the component parameters. As illustrated in the diagram, the maximum rate-of-change during turn-off is determined by t_f .

At $t = t_{\text{on}}$ a positive gate signal is applied. Unless the rate-of-change is limited by other circuit components, the current rises linearly with constant di/dt . The maximum di/dt depends on the rated continuous collector current I_C and the rise time t_r specified in the component parameters:

$$\frac{di_{\max}}{dt} = 0.8 \cdot \frac{I_C}{t_r}$$

The second diagram shows the collector current transients for different on-state currents. It can be seen that the fall time is independent of the on-state current. Since di/dt during turn-on is constant, the actual rise time is proportional to the on-state current. In a real IGBT, the rise time would only vary slightly with different on-state currents. Hence, assuming constant di/dt is a worst-case estimate in respect of the reverse-recovery current in the free-wheeling diode.



Parameters

Blocking voltage

Maximum voltage V_{CES} in volts (V) that under any conditions should be applied between collector and emitter.

Continuous collector current

Maximum dc current I_C in amperes (A) that the IGBT can conduct.

Forward voltage

Additional dc voltage V_f in volts (V) between collector and emitter when the IGBT is conducting. The default is 0.

On-resistance

The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.

Off-resistance

The resistance R_{off} of the blocking device, in ohms (Ω). The default is inf. If multiple IGBTs are connected in series, the off-resistance must have a large finite value.

Rise time

Time t_r in seconds between instants when the collector current has risen from 10 % to 90 % of the continuous collector current I_C (see figure above).

Fall time

Time t_f in seconds between instants when the collector current has dropped from 90 % to 10 % of its initial value along an extrapolated straight line tangent to the maximum rate-of-change of the current (see figure above).

Stray inductance

Internal inductance L_σ in henries (H) measured between the collector and emitter terminals.

Initial current

The initial current through the component at simulation start, in amperes (A). The default is 0.

Probe Signals**IGBT voltage**

The voltage measured between collector and emitter.

IGBT current

The current through the IGBT flowing from collector to emitter.

IGBT conductivity

Conduction state of the internal switch. The signal outputs 0 when the IGBT is blocking, and 1 when it is conducting.

IGCT (Reverse Blocking)

Purpose	Ideal IGCT with optional forward voltage and on-resistance
Library	Electrical / Power Semiconductors
Description	The Integrated Gate Commutated Thyristor is a semiconductor switch that is controlled via the external gate. It conducts a current from anode to cathode only if the gate signal is not zero.
	
Parameters	<p>The following parameters may either be scalars or vectors corresponding to the implicit width of the component:</p> <p>Forward voltage Additional dc voltage V_f in volts (V) between anode and cathode when the IGCT is conducting. The default is 0.</p> <p>On-resistance The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.</p> <p>Initial conductivity Initial conduction state of the IGCT. The IGCT is initially blocking if the parameter evaluates to zero, otherwise it is conducting.</p> <p>Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79). If no thermal description is given the losses are calculated based on the voltage drop $v_{on} = V_f + R_{on} \cdot i$.</p> <p>Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.</p>
Probe Signals	<p>IGCT voltage The voltage measured between anode and cathode.</p> <p>IGCT current The current through the IGCT flowing from anode to cathode.</p> <p>IGCT gate signal The gate input signal of the IGCT.</p>

IGCT conductivity

Conduction state of the internal switch. The signal outputs 0 when the IGCT is blocking, and 1 when it is conducting.

IGCT junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

IGCT conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

IGCT switching loss

Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

IGCT (Reverse Conducting)

Purpose	Ideal IGCT with ideal anti-parallel diode
Library	Electrical / Power Semiconductors
Description	This model of an Integrated Gate Commutated Thyristor has an integrated anti-parallel diode. The diode is usually included in power IGCT packages.
	
Parameters	<p>Initial conductivity Initial conduction state of the device. The device is initially blocking if the parameter evaluates to zero, otherwise it is conducting. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.</p> <p>Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapters “Thermal Modeling” (on page 79) and “Losses of Semiconductor Switch with Diode” (on page 94).</p> <p>Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start. This parameter may either be a scalar or a vector corresponding to the implicit width of the component.</p>
Probe Signals	<p>Device voltage The voltage measured between anode and cathode. The device voltage can never be negative.</p> <p>Device current The current through the device. The current is positive if it flows through the IGCT from anode to cathode and negative if it flows through the diode from cathode to anode.</p> <p>Device gate signal The gate input signal of the device.</p> <p>Device conductivity Conduction state of the internal switch. The signal outputs 0 when the device is blocking, and 1 when it is conducting.</p>

Device junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

Device conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

Device switching loss

Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

Induction Machine

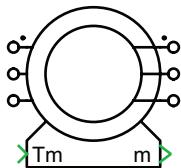
Purpose

Non-saturable induction machine with slip-ring rotor

Library

Electrical / Machines

Description



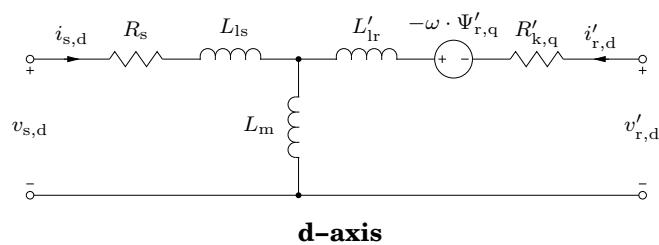
This model of a slip-ring induction machine can only be used with the continuous state-space method. If you want to use the discrete state-space method or if you need to take saturation into account, please use the Induction Machine with Saturation (see page 296).

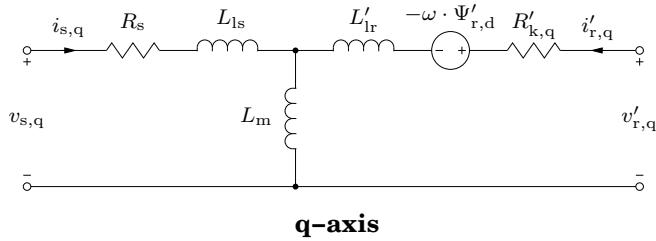
The machine model is based on a stationary reference frame (Clarke transformation). A sophisticated implementation of the Clarke transformation facilitates the connection of external inductances in series with the stator windings. However, external inductors cannot be connected to the rotor windings due to the current sources in the model. In this case, external inductors must be included in the leakage inductance of the rotor.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator and rotor windings is marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit** menu. If you want to make changes, you must first choose **Break library link** and then **Unprotect**, both from the **Edit** menu.

Electrical System





The rotor flux is computed as

$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q})$$

The three-phase voltages $v_{s,ab}$ and $v_{s,bc}$ at the stator terminals are transformed into dq quantities:

$$\begin{bmatrix} v_{s,d} \\ v_{s,q} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} v_{s,ab} \\ v_{s,bc} \end{bmatrix}$$

Likewise, the stator currents in the stationary reference frame are transformed back into three-phase currents:

$$\begin{bmatrix} i_{s,a} \\ i_{s,b} \\ i_{s,c} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} i_{s,d} \\ i_{s,q} \end{bmatrix}$$

Similar equations apply to the voltages and currents at the rotor terminals with θ being the electrical rotor position:

$$\begin{bmatrix} v'_{r,d} \\ v'_{r,q} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos \theta & -\cos(\theta - \frac{2\pi}{3}) \\ \sin \theta & -\sin(\theta - \frac{2\pi}{3}) \end{bmatrix} \cdot \begin{bmatrix} v'_{r,ab} \\ v'_{r,bc} \end{bmatrix}$$

$$\begin{bmatrix} i'_{r,a} \\ i'_{r,b} \\ i'_{r,c} \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ \cos(\theta + \frac{2\pi}{3}) & \sin(\theta + \frac{2\pi}{3}) \\ \cos(\theta - \frac{2\pi}{3}) & \sin(\theta - \frac{2\pi}{3}) \end{bmatrix} \cdot \begin{bmatrix} i'_{r,d} \\ i'_{r,q} \end{bmatrix}$$

Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p L_m (i_{s,q} i'_{r,d} - i_{s,d} i'_{r,q})$$

Mechanical System

Mechanical rotor speed ω_m :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p\omega_m$$

Mechanical rotor angle θ_m :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p\theta_m$$

Parameters

Stator resistance

Stator winding resistance R_s in ohms (Ω).

Stator leakage inductance

Stator leakage inductance L_{ls} in henries (H).

Rotor resistance

Rotor winding resistance R'_r in ohms (Ω), referred to the stator side.

Rotor leakage inductance

Rotor leakage inductance L'_{lr} in henries (H), referred to the stator side.

Magnetizing inductance

Magnetizing inductance L_m in henries (H), referred to the stator side.

Inertia

Combined rotor and load inertia J in Nms².

Friction coefficient

Viscous friction F in Nms.

Number of pole pairs

Number of pole pairs p .

Initial rotor speed

Initial mechanical rotor speed $\omega_{m,0}$ in s⁻¹.

Initial rotor position

Initial mechanical rotor angle $\theta_{m,0}$ in radians. If $\theta_{m,0}$ is an integer multiple of $2\pi/p$ the stator windings are aligned with the rotor windings at simulation start.

Initial stator currents

A two-element vector containing the initial stator currents $i_{s,a,0}$ and $i_{s,b,0}$ of phases a and b in amperes (A).

Initial stator flux

A two-element vector containing the initial stator flux $\Psi'_{s,d,0}$ and $\Psi'_{s,q,0}$ in the stationary reference frame in Vs.

Inputs and Outputs

Mechanical torque

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector “m” contains the following 3 signals:

(1) Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

(2) Rotor position

The mechanical rotor angle θ_m in radians.

(3) Electrical torque

The electrical torque T_e of the machine in Nm.

Probe Signals

Stator phase currents

The three-phase stator winding currents $i_{s,a}$, $i_{s,b}$ and $i_{s,c}$, in A. Currents flowing into the machine are considered positive.

Rotor phase currents

The three-phase rotor winding currents $i'_{r,a}$, $i'_{r,b}$ and $i'_{r,c}$ in A, referred to the stator side. Currents flowing into the machine are considered positive.

Stator flux (dq)

The stator flux linkages $\Psi_{s,d}$ and $\Psi_{s,q}$ in the stationary reference frame in Vs:

$$\Psi_{s,d} = L_{ls} i_{s,d} + L_m (i_{s,d} + i'_{r,d})$$

$$\Psi_{s,q} = L_{ls} i_{s,q} + L_m (i_{s,q} + i'_{r,q})$$

Magnetizing flux (dq)

The magnetizing flux linkages $\Psi_{m,d}$ and $\Psi_{m,q}$ in the stationary reference frame in Vs:

$$\Psi_{m,d} = L_m (i_{s,d} + i'_{r,d})$$

$$\Psi_{m,q} = L_m (i_{s,q} + i'_{r,q})$$

Rotor flux (dq)

The rotor flux linkages $\Psi'_{r,d}$ and $\Psi'_{r,q}$ in the stationary reference frame in Vs.

Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

Rotor position

The mechanical rotor angle θ_m in radians.

Electrical torque

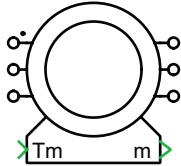
The electrical torque T_e of the machine in Nm.

Induction Machine (Open Stator Windings)

Purpose Non-saturable induction machine with squirrel-cage rotor and open stator windings

Library Electrical / Machines

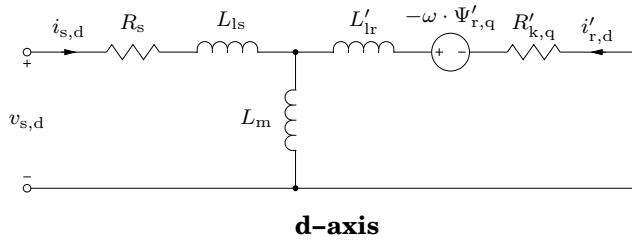
Description This model of a squirrel-cage induction machine can only be used with the continuous state-space method. The machine model is based on a stationary reference frame (Clarke transformation). A sophisticated implementation of the Clarke transformation facilitates the connection of external inductances in series with the stator windings.



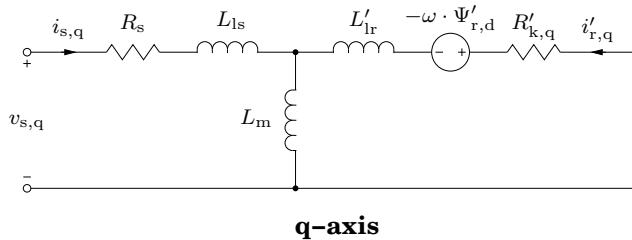
The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, the positive terminal of phase a of the stator windings is marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit** menu.

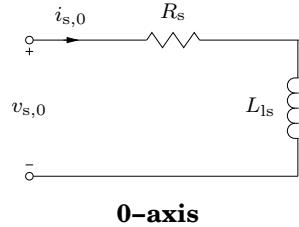
Electrical System:



d-axis



q-axis



The rotor flux is computed as

$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q})$$

The three-phase voltages $v_{s,a}$, $v_{s,b}$ and $v_{s,c}$ across the individual stator windings are transformed into d₀q₀ quantities:

$$\begin{bmatrix} v_{s,d} \\ v_{s,q} \\ v_{s,0} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} v_{s,a} \\ v_{s,b} \\ v_{s,c} \end{bmatrix}$$

Likewise, the stator currents in the stationary reference frame are transformed back into three-phase currents:

$$\begin{bmatrix} i_{s,a} \\ i_{s,b} \\ i_{s,c} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} i_{s,d} \\ i_{s,q} \\ i_{s,0} \end{bmatrix}$$

Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p L_m (i_{s,q} i'_{r,d} - i_{s,d} i'_{r,q})$$

Mechanical System

Mechanical rotor speed ω_m :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p \omega_m$$

Mechanical rotor angle θ_m :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p \theta_m$$

Parameters

Most parameters for the Induction Machine with slip-ring rotor (see page 285) are also applicable for this machine. Only the following parameter differs:

Initial stator currents

A three-element vector containing the initial stator currents $i_{s,a,0}$, $i_{s,b,0}$ and $i_{s,c,0}$ of phase a, b and c in amperes (A).

Inputs and Outputs

Same as for the Induction Machine with slip-ring rotor (see page 285).

Probe Signals

Most probe signals for the Induction Machine with slip-ring rotor (see page 285) are also available with this machine. Only the following probe signal is different:

Rotor currents

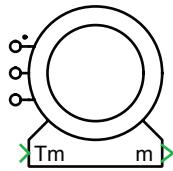
The rotor currents $i'_{r,d}$ and $i'_{r,q}$ in the stationary reference frame in A, referred to the stator side.

Induction Machine (Squirrel-Cage)

Purpose Non-saturable induction machine with squirrel-cage rotor

Library Electrical / Machines

Description This model of a squirrel-cage induction machine can only be used with the continuous state-space method. If you want to use the discrete state-space method or if you need to take saturation into account, please use the Induction Machine with Saturation (see page 296) and short-circuit the rotor terminals.

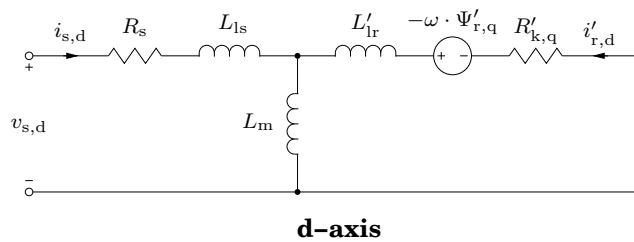


The machine model is based on a stationary reference frame (Clarke transformation). A sophisticated implementation of the Clarke transformation facilitates the connection of external inductances in series with the stator windings.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator winding is marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit** menu.

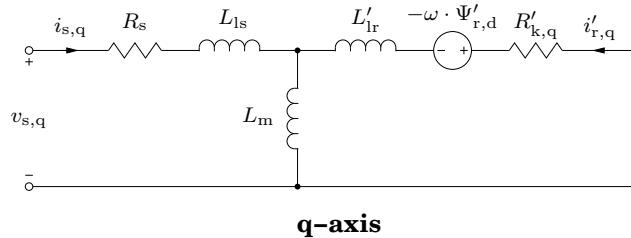
Electrical System



The rotor flux is computed as

$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q})$$



The three-phase voltages $v_{s,ab}$ and $v_{s,bc}$ at the stator terminals are transformed into dq quantities:

$$\begin{bmatrix} v_{s,d} \\ v_{s,q} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} v_{s,ab} \\ v_{s,bc} \end{bmatrix}$$

Likewise, the stator currents in the stationary reference frame are transformed back into three-phase currents:

$$\begin{bmatrix} i_{s,a} \\ i_{s,b} \\ i_{s,c} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} i_{s,d} \\ i_{s,q} \end{bmatrix}$$

Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p L_m (i_{s,q} i'_{r,d} - i_{s,d} i'_{r,q})$$

Mechanical System

Mechanical rotor speed ω_m :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p\omega_m$$

Mechanical rotor angle θ_m :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p\theta_m$$

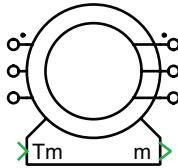
Parameters	Same as for the Induction Machine with slip-ring rotor (see page 285).
Inputs and Outputs	Same as for the Induction Machine with slip-ring rotor (see page 285).
Probe Signals	Most probe signals for the Induction Machine with slip-ring rotor (see page 285) are also available with this squirrel-cage machine. Only the following probe signal is different:
Rotor currents	The rotor currents $i'_{r,d}$ and $i'_{r,q}$ in the stationary reference frame in A, referred to the stator side.

Induction Machine with Saturation

Purpose

Induction machine with slip-ring rotor and main-flux saturation

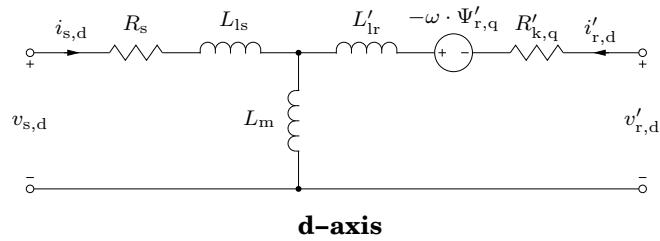
Description



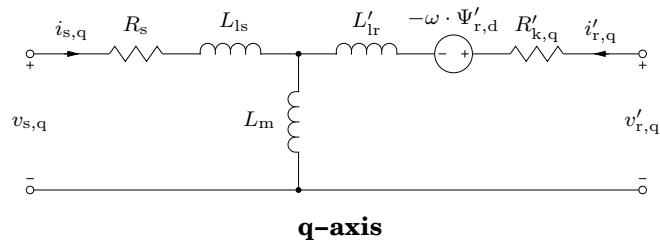
The Induction Machine with Saturation models main flux saturation by means of a continuous function.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator and rotor winding is marked with a dot.

Electrical System:



d-axis



q-axis

The rotor flux is defined as

$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q}) .$$

The machine model offers two different implementations of the electrical system: a traditional stationary reference frame and a voltage-behind-reactance formulation.

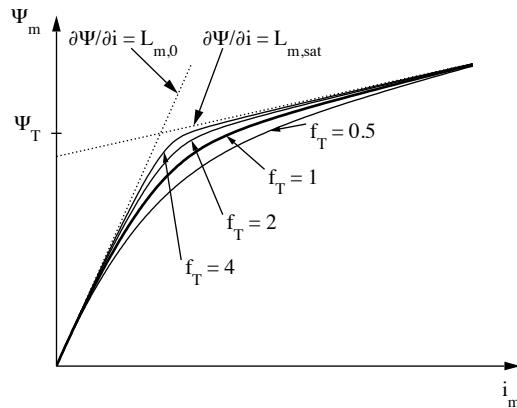
Stationary Reference Frame This implementation is based on machine equations in the stationary reference frame (Clarke transformation). Constant coefficients in the stator and rotor equations make the model numerically efficient. However, interfacing the reference frame with the external 3-phase network may be difficult. Since the coordinate transformations are based on voltage-controlled current sources inductors and naturally commutated devices such as diode rectifiers may not be directly connected to the stator terminals. In these cases, fictitious RC snubbers are required to create the necessary voltages across the terminals. The implementation can be used with both the continuous and the discrete state-space method.

Voltage behind Reactance This formulation allows for direct interfacing of arbitrary external networks with the 3-phase stator terminals. The rotor dynamics are expressed using explicit state-variable equations while the stator branch equations are described in circuit form. However, due to the resulting time-varying inductance matrices, this implementation is numerically less efficient than the traditional reference frame.

In both implementations, the value of the main flux inductances $L_{m,d}$ and $L_{m,q}$ are not constant but depend on the main flux linkage Ψ_m as illustrated in the Ψ_m/i_m diagram. For flux linkages far below the transition flux Ψ_T , the relationship between flux and current is almost linear and is determined by the unsaturated magnetizing inductance $L_{m,0}$. For large flux linkages the relationship is governed by the saturated magnetizing inductance $L_{m,sat}$. Ψ_T defines the knee of the transition between unsaturated and saturated main flux inductance. The tightness of the transition is defined with the form factor f_T . If you do not have detailed information about the saturation characteristic of your machine, $f_T = 1$ is a good starting value. The function

```
plsaturation(Lm0,Lmsat,PsiT,fT)
```

plots the main flux vs. current curve and the magnetizing inductance vs. current curve for the parameters specified.



The model accounts for steady-state cross-saturation, i.e. the steady-state magnetizing inductances along the d-axis and q-axis are functions of the currents in both axes. In the implementation, the stator currents and the main flux linkage are chosen as state variables. With this type of model, the representation of dynamic cross-saturation can be neglected without affecting the machine's performance. The computation of the time derivative of the main flux inductance is not required.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit** menu. If you want to make changes, you must first choose **Break library link** and then **Unprotect**, both from the **Edit** menu.

Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p (i_{s,q} \Psi_{s,d} - i_{s,d} \Psi_{s,q})$$

Mechanical System

Mechanical rotor speed ω_m :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p\omega_m$$

Mechanical rotor angle θ_m :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p\theta_m$$

Parameters

Model

Implementation in the stationary reference frame or as a voltage behind reactance.

Stator resistance

Stator winding resistance R_s in ohms (Ω).

Stator leakage inductance

Stator leakage inductance L_{ls} in henries (H).

Rotor resistance

Rotor winding resistance R'_r in ohms (Ω), referred to the stator side.

Rotor leakage inductance

Rotor leakage inductance L'_{lr} in henries (H), referred to the stator side.

Unsaturated magnetizing inductance

Unsaturated main flux inductance $L_{m,0}$, in henries (H), referred to the stator side. If you do not want to model saturation, set $L_{m,sat} = L_{m,0}$.

Magnetizing flux at saturation transition

Transition flux linkage Ψ_T , in Vs, defining the knee between unsaturated and saturated main flux inductance.

Tightness of saturation transition

Form factor f_T defining the tightness of the transition between unsaturated and saturated main flux inductance. The default is 1.

Inertia

Combined rotor and load inertia J in Nms².

Friction coefficient

Viscous friction F in Nms.

Number of pole pairs

Number of pole pairs p .

Initial rotor speed

Initial mechanical rotor speed $\omega_{m,0}$ in s⁻¹.

Initial rotor position

Initial mechanical rotor angle $\theta_{m,0}$ in radians. If $\theta_{m,0}$ is an integer multiple of $2\pi/p$ the stator windings are aligned with the rotor windings at simulation start.

Initial stator currents

A two-element vector containing the initial stator currents $i_{s,a,0}$ and $i_{s,b,0}$ of phases a and b in amperes (A).

Initial stator flux

A two-element vector containing the initial stator flux $\Psi_{s,d,0}$ and $\Psi_{s,q,0}$ in the stationary reference frame in Vs.

Inputs and Outputs

Mechanical torque

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector “m” contains the following 3 signals:

(1) Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

(2) Rotor position

The mechanical rotor angle θ_m in radians.

(3) Electrical torque

The electrical torque T_e of the machine in Nm.

Probe Signals

Stator phase currents

The three-phase stator winding currents $i_{s,a}$, $i_{s,b}$ and $i_{s,c}$, in A. Currents flowing into the machine are considered positive.

Rotor phase currents

The three-phase rotor winding currents $i'_{r,a}$, $i'_{r,b}$ and $i'_{r,c}$ in A, referred to the stator side. Currents flowing into the machine are considered positive.

Stator flux (dq)

The stator flux linkages $\Psi_{s,d}$ and $\Psi_{s,q}$ in the stationary reference frame in Vs.

Magnetizing flux (dq)

The magnetizing flux linkages $\Psi_{m,d}$ and $\Psi_{m,q}$ in the stationary reference frame in Vs.

Rotor flux (dq)

The rotor flux linkages $\Psi'_{r,d}$ and $\Psi'_{r,q}$ in the stationary reference frame in Vs, referred to the stator side.

Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

Rotor position

The mechanical rotor angle θ_m in radians.

Electrical torque

The electrical torque T_e of the machine in Nm.

References

- D. C. Aliprantis, O. Wasyczuk, C. D. Rodriguez Valdez, "A voltage-behind-reactance synchronous machine model with saturation and arbitrary rotor network representation", IEEE Transactions on Energy Conversion, Vol. 23, No. 2, June 2008.
- K. A. Corzine, B. T. Kuhn, S. D. Sudhoff, H. J. Hegner, "An improved method for incorporating magnetic saturation in the Q-D synchronous machine model", IEEE Transactions on Energy Conversion, Vol. 13, No. 3, Sept. 1998.
- E. Levi, "A unified approach to main flux saturation modelling in D-Q axis models of induction machines", IEEE Transactions on Energy Conversion, Vol. 10, No. 3, Sept. 1995.
- E. Levi, "Impact of cross-saturation on accuracy of saturated induction machine models", IEEE Transactions on Energy Conversion, Vol. 12, No. 3, Sept. 1997.

Inductor

Purpose Ideal inductor

Library Electrical / Passive Components

Description This component provides one or multiple ideal inductors between its two electrical terminals. If the component is vectorized, a magnetic coupling can be specified between the internal inductors. Inductors may be switched in series only if their momentary currents are equal.

Note An inductor may not be connected in series with a current source. Doing so would create a dependency between an input variable (the source current) and a state variable (the inductor current) in the underlying state-space equations.

Parameters

Inductance

The inductance in henries (H). All finite positive and negative values are accepted, including 0. The default is 0.001.

In a vectorized component, all internal inductors have the same inductance if the parameter is a scalar. To specify the inductances individually use a vector $[L_1 \ L_2 \dots \ L_n]$. The length n of the vector determines the component's width:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} L_1 & 0 & \cdots & 0 \\ 0 & L_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt}i_1 \\ \frac{d}{dt}i_2 \\ \vdots \\ \frac{d}{dt}i_n \end{bmatrix}$$

In order to model a magnetic coupling between the internal inductors enter a square matrix. The size n of the matrix corresponds to the width of the component. L_i is the self inductance of the internal inductor and $M_{i,j}$ the mutual inductance:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} L_1 & M_{1,2} & \cdots & M_{1,n} \\ M_{2,1} & L_2 & \cdots & M_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & L_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt} i_1 \\ \frac{d}{dt} i_2 \\ \vdots \\ \frac{d}{dt} i_n \end{bmatrix}$$

The inductance matrix must be invertible, i.e. it may not be singular. A singular inductance matrix results for example when two or more inductors are ideally coupled. To model this, use an inductor in parallel with an Ideal Transformer (see page 268).

The relationship between the coupling factor $k_{i,j}$ and the mutual inductance $M_{i,j}$ is

$$M_{i,j} = M_{j,i} = k_{i,j} \cdot \sqrt{L_i \cdot L_j}$$

Initial current

The initial current through the inductor at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the width of the component. The direction of a positive initial current is indicated by a small arrow in the component symbol. The default of the initial current is 0.

Probe Signals

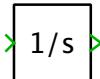
Inductor current

The current flowing through the inductor, in amperes (A). The direction of a positive current is indicated with a small arrow in the component symbol.

Inductor voltage

The voltage measured across the inductor, in volts (V).

Integrator

Purpose	Integrate input signal with respect to time
Library	Control / Continuous
Description	The Integrator block outputs the integral of its input signal at the current time step. The output signal may have an upper and lower limit. It can be reset to its initial value by an external trigger signal.
	

Simulation with the Continuous State-Space Method

When simulated with the continuous method, the input signal is simply passed on to the solver for integration.

Simulation with the Discrete State-Space Method

When simulated with the discrete method, the input signal is integrated within PLECS using the Forward Euler method.

Parameter	Initial condition The initial condition of the integrator. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. External reset The behaviour of the external reset input. The values <code>rising</code> , <code>falling</code> and <code>either</code> cause a reset of the integrator on the rising, falling or both edges of the reset signal. A rising edge is detected when the signal changes from 0 to a positive value, a falling edge is detected when the signal changes from a positive value to 0. If the value <code>level</code> is chosen, the output signal keeps the initial value while the reset input is not 0. Upper saturation limit An upper limit for the output signal. If the value is <code>inf</code> the output signal is unlimited. Lower saturation limit A lower limit for the output signal. If the value is <code>-inf</code> the output signal is unlimited.
------------------	---

Probe Signals**State**

The internal state of the integrator.

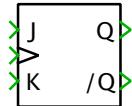
Note If the external reset is used, there is a direct dependency of the output signal on the reset trigger signal. Therefore the generation of the trigger signal must not depend on the output signal directly since this would cause an algebraic loop.

JK Flip-flop

Purpose Implement edge-triggered JK flip-flop

Library Control / Logical

Description The JK flip-flop changes its output when an edge in the clock signal is detected according to the following truth table:



J	K	Q	/Q
0	0	No change	No change
0	1	0	1
1	0	1	0
1	1	$/Q_{prev}$	Q_{prev}

As long as no edge is detected in the clock signal the outputs remain stable. When a trigger occurs and $J = K = 1$ the outputs are toggled, i.e. change from 1 to 0 or vice versa.

The inputs J and K are latched, i.e. when a triggering edge in the clock signal is detected the values of J and K from the previous simulation step are used to set the output. In other words, J and K must be stable for at least one simulation step before the flip-flop is triggered by the clock signal.

Parameter **Trigger edge**
The direction of the edge on which the inputs are read.

Initial state
The state of the flip-flop at simulation start.

Probe Signals **J**
The input signal J .
K
The input signal K .
Clk
The clock input signal.
Q
The output signals Q .

/Q

The output signals /Q.

Leakage Flux Path

Purpose	Permeance of linear leakage flux path
Library	Magnetic
Description	This component models a magnetic leakage flux path. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force \mathcal{F} . Magnetic permeance \mathcal{P} is the reciprocal of magnetic reluctance \mathcal{R} :
	$\mathcal{P} = \frac{1}{\mathcal{R}} = \frac{\Phi}{\mathcal{F}}$
	This component is equivalent to the Magnetic Permeance (see page 315). The only difference is the symbol.
Parameters	<p>Effective Permeance Magnetic permeance of the leakage flux path, in webers per ampere-turn (Wb/A).</p> <p>Initial MMF Magneto-motive force at simulation start, in ampere-turns (A).</p>
Probe Signals	<p>MMF The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).</p> <p>Flux The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.</p>

Linear Core

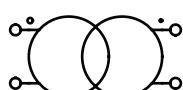
Purpose	Linear magnetic core element
Library	Magnetic
Description	This component models a segment of a magnetic core. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force \mathcal{F}
	$\frac{\Phi}{\mathcal{F}} = \frac{\mu_0 \mu_r A}{l}$ where $\mu_0 = 4\pi \times 10^{-7} \text{ N/A}^2$ is the magnetic constant, μ_r is the relative permeability of the material, A is the cross-sectional area and l the length of the flux path.
Parameters	<p>Cross-sectional area Cross-sectional area A of the flux path, in m^2.</p> <p>Length of flux path Length l of the flux path, in m.</p> <p>Rel. permeability Relative permeability μ_r of the core material.</p> <p>Initial MMF Magneto-motive force at simulation start, in ampere-turns (A).</p>
Probe Signals	<p>MMF The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).</p> <p>Flux The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.</p> <p>Field strength The magnetic field strength H in the core element, in A/m.</p> <p>Flux density The magnetic flux density B in the core element, in teslas (T).</p>

Linear Transformer (2 Windings)

Purpose Single-phase transformer with winding resistance and optional core loss

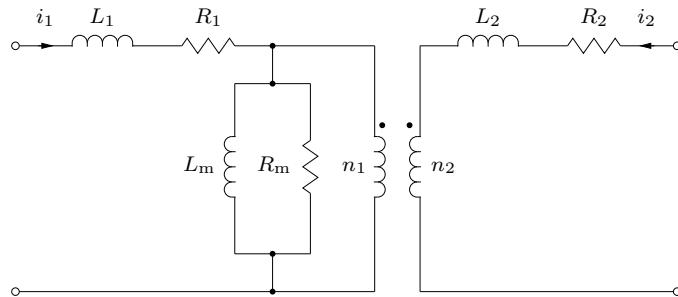
Library Electrical / Transformers

Description



This transformer models two coupled windings on the same core. The magnetization inductance L_m and the core loss resistance R_m are modeled as linear elements. Their values are referred to the primary side. A stiff solver is recommended if R_m is not infinite.

The electrical circuit for this component is given below:



In the transformer symbol, the primary side winding is marked with a little circle. The secondary side winding is marked with a dot.

Parameters

Leakage inductance

A two-element vector containing the leakage inductance of the primary side L_1 and the secondary side L_2 . The inductivity is given in henries (H).

Winding resistance

A two-element vector containing the resistance of the primary winding R_1 and the secondary winding R_2 , in ohms (Ω).

Winding ratio

The ratio n_1/n_2 between the number of turns of the primary and secondary winding.

Magnetization inductance

The magnetization inductance L_m , in henries (H). The value is referred to the primary side.

Core loss resistance

An equivalent resistance R_m representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

Initial current

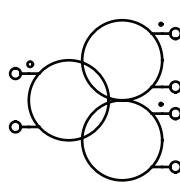
A two-element vector containing the initial currents on the primary side i_1 and the secondary side i_2 , in amperes (A). The currents are considered positive if flowing into the transformer at the marked terminals. The default is [0 0].

Linear Transformer (3 Windings)

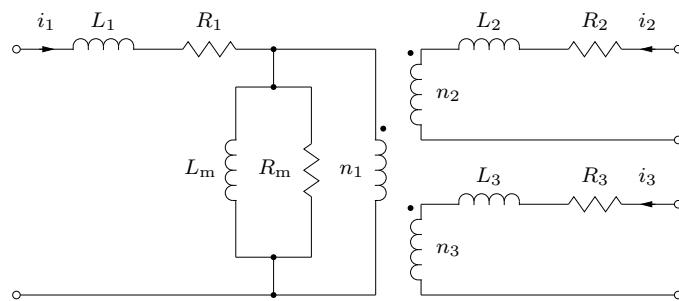
Purpose Single-phase transformer with winding resistance and optional core loss

Library Electrical / Transformers

Description This transformer models three coupled windings on the same core. The magnetization inductance L_m and the core loss resistance R_m are modeled as linear elements. Their values are referred to the primary side. A stiff solver is recommended if R_m is not infinite.



The electrical circuit for this component is given below:



In the transformer symbol, the primary side winding is marked with a little circle. The secondary winding is marked with a dot at the outside terminal, the tertiary winding with a dot at the inside terminal.

Parameters

Leakage inductance

A three-element vector containing the leakage inductance of the primary side L_1 , the secondary side L_2 and the tertiary side L_3 . The inductivity is given in henries (H).

Winding resistance

A three-element vector containing the resistance of the primary winding R_1 , the secondary winding R_2 and the tertiary winding R_3 , in ohms (Ω).

No. of turns

A three-element vector containing the number of turns of the primary winding n_1 , the secondary winding n_2 and the tertiary winding n_3 .

Magnetization inductance

The magnetization inductance L_m , in henries (H). The value is referred to the primary side.

Core loss resistance

An equivalent resistance R_m representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

Initial current

A three-element vector containing the initial currents on the primary side i_1 , the secondary side i_2 and the tertiary side i_3 , in amperes (A). The currents are considered positive if flowing into the transformer at the marked terminals. The default is [0 0 0].

Logical Operator

Purpose Combine input signals logically

Library Control / Logical

Description The selected Logical Operator is applied to the input signals. The output of the Logical Operator is 1 if the logical operation returns true, otherwise 0. In case of a single input, the operator is applied to all elements of the input vector.



Parameters

Operator

Chooses which logical operator is applied to the input signals. Available operators are

- AND $y = u_n \& u_{n-1} \& \dots \& u_1 \& u_0$
- OR $y = u_n | u_{n-1} | \dots | u_1 | u_0$
- NAND $y = \sim(u_n \& u_{n-1} \& \dots \& u_1 \& u_0)$
- NOR $y = \sim(u_n | u_{n-1} | \dots | u_1 | u_0)$
- XOR $y = u_n \text{ xor } u_{n-1} \text{ xor } \dots \text{ xor } u_1 \text{ xor } u_0$
- NOT $y = \sim u$

Number of inputs

The number of input terminals. If the NOT operator is selected, the number of inputs is automatically set to 1.

Probe Signals

Input

The input signals.

Output

The output signal.

Magnetic Permeance

Purpose	Linear magnetic permeance
Library	Magnetic
Description	This component provides a magnetic flux path. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force \mathcal{F} . Magnetic permeance \mathcal{P} is the reciprocal of magnetic reluctance \mathcal{R} :
	$\mathcal{P} = \frac{1}{\mathcal{R}} = \frac{\Phi}{\mathcal{F}}$
Parameters	<p>Permeance Magnetic permeance of the flux path, in webers per ampere-turn (Wb/A).</p> <p>Initial MMF Magneto-motive force at simulation start, in ampere-turns (A).</p>
Probe Signals	<p>MMF The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).</p> <p>Flux The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.</p>

Magnetic Port

Purpose	Add magnetic connector to subsystem
Library	Magnetic
Description	<p>Magnetic ports are used to establish magnetic connections between a schematic and the subschematic of a subsystem (see page 402). If you copy a Magnetic Port block into the schematic of a subsystem a terminal will be created on the subsystem block. The name of the port block will appear as the terminal label. If you choose to hide the block name by unselecting the show name option in the block menu the terminal label will also disappear.</p> <p>Terminals can be moved around the edges of the subsystem by holding down the Shift key while dragging the terminal with the left mouse button or by using the middle mouse button.</p>
	

Note Since you cannot make magnetic connections in Simulink, PLECS does not permit to place magnetic port blocks into top-level schematics.

Magnetic Resistance

Purpose Effective magnetic resistance for modeling losses

Library Magnetic

Description Magnetic resistances (analogous to electrical resistors) are used to model frequency-depending losses in the magnetic circuit. They can be connected in series or in parallel to a permeance, depending on the nature of the specific loss. The energy relationship is maintained as the power



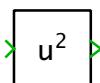
$$P_{\text{loss}} = F \dot{\Phi} = F^2 / R_m$$

converted into heat in a magnetic resistance corresponds to the power lost in the electrical circuit.

Parameters **Resistance**
Effective magnetic resistance R_m , in $\text{A} \cdot (\text{Wb}/\text{s})^{-1}$.

Probe Signals **MMF**
The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).

Math Function

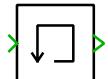
Purpose	Apply specified mathematical function
Library	Control / Math
Description	The Math Function block calculates the output by applying the specified function to the input. For functions that require two inputs, the first input is marked with a black dot.
	
Parameter	<p>Function</p> <p>Chooses which function is applied to the input signals. Available functions are</p> <ul style="list-style-type: none"> • square $y = u^2$ • square root $y = \sqrt{u}$ • exponential $y = e^u$ • logarithm $y = \ln(u)$ • power $y = u^v$ • mod $y = \text{mod}(u, v)$ • rem $y = \text{rem}(u, v)$ <p><code>mod</code> and <code>rem</code> both return the floating-point remainder of u/v. If u and v have different signs, the result of <code>rem</code> has the same sign as u while the result of <code>mod</code> has the same sign as v.</p>
Probe Signals	<p>Input The input signals.</p> <p>Output The output signal.</p>

Memory

Purpose Provide input signal from previous major time step

Library Control / Delays

Description The Memory block delays the input signal by a single major time step taken by the solver.



Note If a variable-step solver is used the delay time of the Memory block also varies. This should be kept in mind when using a memory block to decouple algebraic loops.

Parameter **Initial condition**

The initial output during the first major time step.

Probe Signals

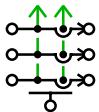
Input

The input signal.

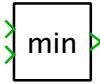
Output

The output signal.

Meter (3-Phase)

Purpose	Measure voltages and currents of 3-phase system
Library	Electrical / Meters
Description	The meter block acts as a set of volt- and ammeters. Voltages can be measured from line to ground (V_a , V_b and V_c) or from line to line (V_{ab} , V_{bc} , V_{ca}) depending on the Voltage measurement parameter. The output for voltage and current is a vectorized signal with three elements.
	
Parameter	<p>Voltage measurement Determine whether the voltages are measured from line to ground or from line to line.</p>
Probe Signals	<p>Measured voltage The measured voltages as a vector with three elements.</p> <p>Measured current The measured currents as a vector with three elements.</p>

Minimum / Maximum

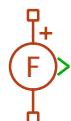
Purpose	Output input signal with highest resp. lowest value
Library	Control / Math
Description	<p>The Minimum / Maximum block compares its input signals against each other. If the Operation parameter is set to Minimum, the output will be set to the value of the input signal with the lowest value. If the Operation parameter is set to Maximum, the output will be set to the value of the input signal with the highest value.</p>  <p>In case of a single input, all elements of the input vector are compared. Vectorized input signals of the same width are compared element wise and result in a vectorized output signal. If vectorized and scalar input signals are mixed, the scalar input signals are expanded to the width of the vectorized input signals.</p>
Parameter	<p>Operation Selects between Minimum and Maximum as described above.</p> <p>Number of inputs The number of inputs.</p>
Probe Signals	<p>Input i The ith input signal.</p> <p>Output The block output signal.</p>

MMF Meter

Purpose Output the measured magneto-motive force

Library Magnetic

Description



The MMF Meter measures the magneto-motive force between its two magnetic terminals and provides it as a signal at the output of the component. A positive MMF is measured when the magnetic potential at the terminal marked with a "+" is greater than at the unmarked one. The output signal can be made accessible in Simulink with an Output block (see page 387) or by dragging the component into the dialog box of a Probe block.

Note The MMF Meter is ideal, i.e. it has an zero internal permeance. Hence, if multiple meters are connected in series the MMF across an individual meter is undefined. This produces a run-time error.

Probe Signals

MMF

The measured magneto-motive force in ampere-turns (A).

MMF Source (Constant)

Purpose	Generate a constant magneto-motive force
Library	Magnetic
Description	The Constant MMF Source generates a constant magneto-motive force (MMF) between its two magnetic terminals. The MMF is considered positive at the terminal marked with a “+”.
	
	Note An MMF source may not be short-circuited or connected in parallel to a permeance or any other MMF source.
Parameter	Voltage The magnitude of the MMF, in ampere-turns (A). The default value is 1.
Probe Signals	MMF The magneto-motive force of the source, in ampere-turns (A).

MMF Source (Controlled)

Purpose	Generate a variable magneto-motive force
Library	Magnetic
Description	The Controlled MMF Source generates a variable magneto-motive force (MMF) between its two terminals. The MMF is considered positive at the terminal marked with a “+”. The momentary MMF is determined by the signal fed into the input of the component.
Probe Signals	MMF The magneto-motive force of the source, in ampere-turns (A).

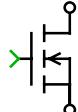


Note An MMF source may not be short-circuited or connected in parallel to a permeance or any other MMF source.

Monoflop

Purpose	Generate pulse of specified width when triggered
Library	Control / Logical
Description	<p>The output of the monoflop changes to 1 when the trigger condition is fulfilled. When the trigger condition is no longer fulfilled the output stays 1 for the given duration and changes to 0 afterwards.</p>  <p>Depending on the trigger type the behavior is as follows:</p> <ul style="list-style-type: none"> rising The output is set to 1 for the given duration when the input changes from 0 to a positive value. falling The output is set to 1 for the given duration when the input changes from a positive value to 0. level The output is set to 1 when the input is a positive value. It stays 1 for the given duration after the input returns to 0. <p>The monoflop is resettable, i.e. if the trigger condition is fulfilled again while the output is 1 a new pulse is started.</p>
Parameter	<p>Trigger type The trigger type as described above.</p> <p>Pulse duration The duration for which the output is set to 1.</p>
Probe Signals	<p>Input The input signal.</p> <p>Output The output signal.</p>

MOSFET

Purpose	Ideal MOSFET with optional on-resistance
Library	Electrical / Power Semiconductors
Description	The Metal Oxide Semiconductor Field Effect Transistor is a semiconductor switch that is controlled via the external gate. It conducts a current from drain to source (or vice-versa) only if the gate signal is not zero.
	
Parameters	The following parameters may either be scalars or vectors corresponding to the implicit width of the component:
	On-resistance The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.
	Initial conductivity Initial conduction state of the MOSFET. The MOSFET is initially blocking if the parameter evaluates to zero, otherwise it is conducting.
	Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79). If no thermal description is given the losses are calculated based on the voltage drop $v_{on} = R_{on} \cdot i$.
	Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.
Probe Signals	MOSFET voltage The voltage measured between drain and source.
	MOSFET current The current through the MOSFET flowing from drain to source.
	MOSFET gate signal The gate input signal of the MOSFET.
	MOSFET conductivity Conduction state of the internal switch. The signal outputs 0 when the MOSFET is blocking, and 1 when it is conducting.

MOSFET junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

MOSFET conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

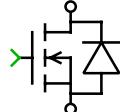
MOSET switching loss

Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

MOSFET Converter (3ph)

Purpose	3-phase MOSFET converter
Library	Electrical / Converters
Description	<p>Implements a three-phase two-level MOSFET converter with reverse diodes. The gate input is a vector of three signals – one per leg. The upper MOSFET, connected to the positive dc level, is on if the corresponding gate signal is positive. The lower MOSFET is on if the gate signal is negative. If the gate signal is zero both MOSFETs in the leg are switched off.</p> <p>You can choose between two different converter models:</p> <ul style="list-style-type: none"> • The basic MOSFET Converter is modeled using the component MOSFET with Diode (see page 329). PLECS needs only six internal switches to simulate this converter. Only the on-resistances of the MOSFETs can be entered. • The MOSFET Converter with Parasitics is based on individual MOSFET (see page 326) and Diode (see page 232) components. In this model you may specify forward voltages and on-resistances separately for the MOSFETs and diodes.
Parameters	For a description of the parameters see the documentation of the MOSFET with Diode (on page 329), the MOSFET (on page 326) and the Diode (on page 232).
Probe Signals	The two-level MOSFET converters provide six or twelve probe signals, each a vector containing the appropriate quantities of the individual devices: voltage, current, conductivity, conduction loss and switching loss. The vector elements are ordered top-to-bottom, left-to-right: a+, a-, b+, b-, c+, c-.

MOSFET with Diode

Purpose	Ideal MOSFET with ideal anti-parallel diode
Library	Electrical / Power Semiconductors
Description	This model of a Metal Oxide Semiconductor Field Effect Transistor has an integrated anti-parallel diode. The diode is usually included in power MOSFET packages.
	
Parameters	<p>Initial conductivity Initial conduction state of the device. The device is initially blocking if the parameter evaluates to zero, otherwise it is conducting. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.</p> <p>Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapters “Thermal Modeling” (on page 79) and “Losses of Semiconductor Switch with Diode” (on page 94).</p> <p>Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start. This parameter may either be a scalar or a vector corresponding to the implicit width of the component.</p>
Probe Signals	<p>Device voltage The voltage measured between drain and source. The device voltage can never be negative.</p> <p>Device current The current through the device. The current is positive if it flows through the MOSFET from drain to source and negative if it flows through the diode from source to drain.</p> <p>Device gate signal The gate input signal of the device.</p> <p>Device conductivity Conduction state of the internal switch. The signal outputs 0 when the device is blocking, and 1 when it is conducting.</p>

Device junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

Device conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

Device switching loss

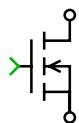
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

MOSFET with Limited di/dt

Purpose Dynamic MOSFET model with finite current slopes during turn-on and turn-off

Library Electrical / Power Semiconductors

Description In contrast to the ideal MOSFET model (see page 326) that switches instantaneously, this model includes drain current transients during switching.



Thanks to the continuous current decay during turn-off, stray inductances may be connected in series with the device.

This MOSFET model is used to simulate overvoltages produced by parasitic inductances and the reverse recovery effect of diodes. Due to simplified voltage and current transient waveforms, the model is not suited for the simulation of switching losses. The dynamic behavior of this MOSFET model is identical with the one of the IGBT with limited di/dt (see page 277).

Note

- Due to the small time-constants introduced by the turn-on and turn-off transients a stiff solver is recommended for this device model.
- If multiple MOSFETs are connected in series, the off-resistance may not be infinite.

Parameters

Blocking voltage

Maximum voltage V_{DSS} in volts (V) that under any conditions should be applied between drain and source.

Continuous drain current

Maximum dc current I_D in amperes (A) that the MOSFET can conduct.

On-resistance

The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.

Off-resistance

The resistance R_{off} of the blocking device, in ohms (Ω). The default is inf. If multiple IGBTs are connected in series, the off-resistance must have a large finite value.

Rise time

Time t_r in seconds between instants when the drain current has risen from 10 % to 90 % of the continuous drain current I_D .

Fall time

Time t_f in seconds between instants when the drain current has dropped from 90 % to 10 % of its initial value along an extrapolated straight line tangent the maximum rate-of-change of the current.

Fall time

Time t_f in seconds between instants when the drain current has dropped from 90 % to 10 % of its initial value along an extrapolated straight line tangent the maximum rate-of-change of the current decay.

Stray inductance

Internal inductance L_σ in henries (H) measured between the drain and source terminals.

Initial current

The initial current through the component at simulation start, in amperes (A). The default is 0.

Probe Signals**MOSFET voltage**

The voltage measured between drain and source.

MOSFET current

The current through the MOSFET flowing from drain to source.

MOSFET conductivity

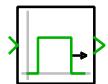
Conduction state of the internal switch. The signal outputs 0 when the MOSFET is blocking, and 1 when it is conducting.

Moving Average

Purpose Continuously average input signal over specified time period

Library Control / Filters

Description The Moving Average filter averages a continuous input signal u over the specified averaging time T . The output y is continuously updated in every simulation step:



$$y(t) = \frac{1}{T} \int_{t-T}^t u(\tau) d\tau$$

The implementation of this block avoids accumulating numerical integration errors typically associated with continuous-time implementations of FIR filters. However, the Moving Average filter is computationally more expensive and less accurate than the similar Periodic Average (see page 343).

Parameters **Averaging time**
The length of the averaging period in sec.

Initial buffer size
Size of the internal ring buffer at simulation start. The buffer size will be increased during the simulation if required.

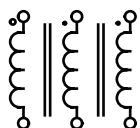
Probe Signals **Input**
The input signal.
Output
The filtered output signal.

Mutual Inductor

Purpose Ideal mutual inductor

Library Electrical / Passive Components

Description This component provides two or more coupled inductors. Electrically, it is equivalent with a vectorized Inductor (see page 302). In contrast to the vectorized Inductor, this component displays the individual inductors in the schematic as separate windings.



In the symbol of the mutual inductor, the positive terminal of winding 1 is marked with a little circle. The positive terminals of all other windings are marked with dots.

Note An inductor may not be connected in series with a current source. Doing so would create a dependency between an input variable (the source current) and a state variable (the inductor current) in the underlying state-space equations.

Parameters

Number of windings

The number of ideal inductors represented by the component.

Inductance

The inductance in henries (H). All finite positive and negative values are accepted, including 0.

If the parameter is a scalar or a vector no coupling exists between the windings. In order to model a magnetic coupling between the windings a square matrix must be entered. The size n of the matrix corresponds to the number of windings. L_i is the self inductance of the internal inductor and $M_{i,j}$ the mutual inductance:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} L_1 & M_{1,2} & \cdots & M_{1,n} \\ M_{2,1} & L_2 & \cdots & M_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & L_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt} i_1 \\ \frac{d}{dt} i_2 \\ \vdots \\ \frac{d}{dt} i_n \end{bmatrix}$$

The inductance matrix must be invertible, i.e. it may not be singular. A singular inductance matrix results for example when two or more inductors are ideally coupled. To model this, use an inductor in parallel with an Ideal Transformer (see page 268).

The relationship between the coupling factor $k_{i,j}$ and the mutual inductance $M_{i,j}$ is

$$M_{i,j} = M_{j,i} = k_{i,j} \cdot \sqrt{L_i \cdot L_j}$$

Initial current

The initial current in the windings at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the number of windings. The direction of the initial current inside the component is from the positive to the negative terminal. The default of the initial current is 0.

Probe Signals

Winding i current

The current flowing through winding i , in amperes (A).

Winding i voltage

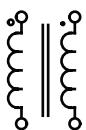
The voltage measured across winding i , in volts (V).

Mutual Inductance (2 Windings)

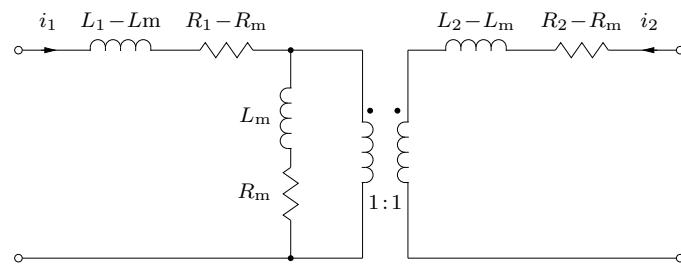
Purpose Magnetic coupling between two lossy windings

Library Electrical / Transformers

Description This component implements a magnetic coupling between two separate windings. For both windings the self inductance and resistance are specified individually. The mutual inductance and resistance are modeled as linear elements.



The electrical circuit for this component is given below:



In the symbol of the mutual inductance, the positive terminal of the primary winding is marked with a little circle. The positive terminal of the secondary winding is marked with a dot.

Parameters

Self inductance

A two-element vector containing the self inductance for the primary winding L_1 and the secondary winding L_2 . The inductivity is given in henries (H).

Winding resistance

A two-element vector containing the self resistance of the primary winding R_1 and the secondary winding R_2 , in ohms (Ω).

Mutual inductance

The mutual inductance L_m , in henries (H).

Mutual resistance

The mutual resistance R_m , in ohms (Ω).

Initial current

A two-element vector containing the initial currents on the primary side i_1 and the secondary side i_2 , in amperes (A). The direction of the initial

current inside the component is from the positive to the negative terminal.
The default value is [0 0].

Probe Signals

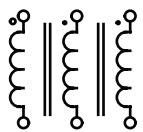
Winding i current

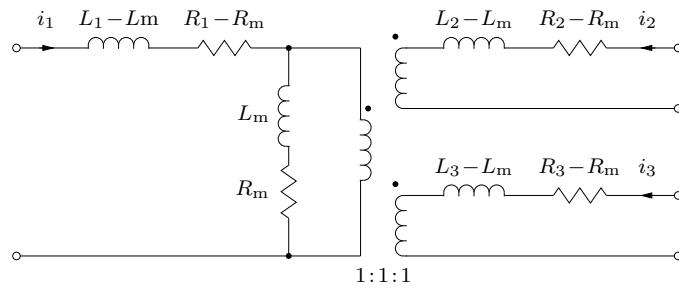
The current flowing through winding i , in amperes (A).

Winding i voltage

The voltage measured across winding i , in volts (V).

Mutual Inductance (3 Windings)

Purpose	Magnetic coupling between three lossy windings
Library	Electrical / Transformers
Description	<p>This component implements a magnetic coupling between three separate windings. For all windings the self inductance and resistance are specified individually. The mutual inductance and resistance are modeled as linear elements.</p>  <p>The electrical circuit for this component is given below:</p>



In the symbol of the mutual inductance, the positive terminal of the primary winding is marked with a little circle. The positive terminals of the secondary and tertiary windings are marked with dots.

Parameters	<p>Self inductance A three-element vector containing the self inductance for the primary winding L_1, the secondary winding L_2 and the tertiary winding L_3. The inductivity is given in henries (H).</p> <p>Winding resistance A three-element vector containing the self resistance of the primary winding R_1, the secondary winding R_2 and the tertiary winding R_3, in ohms (Ω).</p> <p>Mutual inductance The mutual inductance L_m, in henries (H).</p> <p>Mutual resistance The mutual resistance R_m, in ohms (Ω).</p>
-------------------	--

Initial current

A three-element vector containing the initial currents on the primary side i_1 , the secondary side i_2 and the tertiary side i_3 , in amperes (A). The direction of the initial current inside the component is from the positive to the negative terminal. The default value is [0 0 0].

Probe Signals**Winding i current**

The current flowing through winding i , in amperes (A).

Winding i voltage

The voltage measured across winding i , in volts (V).

Op-Amp

Purpose Ideal operational amplifier with finite gain

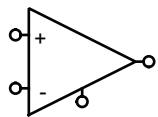
Library Electrical / Electronics

Description This Op-Amp amplifies a voltage between the non-inverting "+" and inverting "-" input with a specified gain. The resulting voltage is applied between the output and ground terminal. Output and ground are electrically isolated from the inputs. If you want to build a linear amplifier the output voltage must somehow be fed back to the inverting input. The demo models p10pAmps and p1ActiveLowPass demonstrate different applications with op-amps.

Parameter

Open-loop gain

The voltage gain of the Op-Amp. The default is 1e6.



Op-Amp with Limited Output

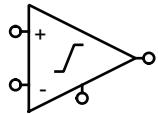
Purpose

Ideal operational amplifier with limited output voltage

Library

Electrical / Electronics

Description



This component amplifies a voltage between the non-inverting “+” and inverting “-” input with a specified gain, taking into account the specified output voltage limits. The resulting voltage is applied between the output and ground terminal. Output and ground are electrically isolated from the inputs. If you want to build a linear amplifier the output voltage must somehow be fed back to the inverting input. The demo model p10pAmps shows a possible application of the Limited Op-Amp.

Parameters

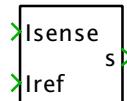
Open-loop gain

The voltage gain of the amplifier if operating in linear mode. The default is 1e6.

Output voltage limits

A two-element vector containing the minimum and maximum output voltage V_{\min} and V_{\max} in volts (V). The default is [-10 10].

Peak Current Controller

Purpose	Implement peak current mode control
Library	Control / Modulators
Description	This block implements current mode control in a switching converter. At the beginning of each switching cycle, the output is set. When the I_{sense} input exceeds the I_{ref} input, the output is reset.
	
Parameters	<p>Switching frequency The switching frequency of the output signal.</p> <p>Minimum duty cycle This sets the minimum time the output remains on for at the beginning of each switching period. This value must be non-negative and less than the maximum duty cycle.</p> <p>Maximum duty cycle This defines the maximum permissible duty cycle of the switch output. If $I_{sense} < I_{ref}$, the output will turn off if the duty cycle exceeds this maximum value. The maximum duty cycle must be less than 100 %.</p> <p>Slope compensation Slope compensation can be applied to ensure stability when the output duty cycle exceeds 50 %. Entering a parameter, I_{slope}, reduces I_{ref} during each switching cycle as follows: $I'_{ref} = I_{ref} - I_{slope} \cdot t/T_s$, where t is the time elapsed from the start of the switching cycle and T_s is the switching period. Slope compensation can be omitted by setting I_{slope} to 0.</p>

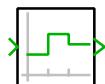
Periodic Average

Purpose

Periodically average input signal over specified time

Library

Control / Filters

Description


This block periodically averages a continuous input signal u over the specified averaging time T . The output y is updated at the end of each averaging period. Mathematically, this block corresponds to a moving average filter where the output is processed by a zero-order hold:

$$y(t) = \frac{1}{T} \int_{t-T}^t u(\tau) d\tau \cdot \text{rect}\left(t - \frac{n + 1/2}{T}\right)$$

However, the implementation of Periodic Average filter is computationally less expensive and more accurate than the continuous Moving Average (see page 333) filter.

The block is suited to determine average conduction losses of power semiconductors. To determine average switching losses, use the Periodic Impulse Average (see page 344).

Parameters
Averaging time

The length of the averaging period (in sec.) and the sample time of the output signal. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Probe Signals
Input

The input signal.

Output

The filtered output signal.

Periodic Impulse Average

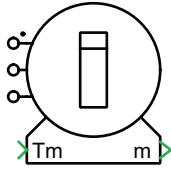
Purpose	Periodically average Dirac impulses over specified time
Library	Control / Filters
Description	<p>This block periodically averages an input signal u consisting of a series of Dirac impulses δ. The output y is updated at the end of each averaging period T. Mathematically, this block corresponds to a moving average filter where the output is processed by a zero-order hold:</p>  $y(t) = \frac{1}{T} \int_{t-T}^t u(\tau) d\tau \cdot \text{rect}\left(t - \frac{n + 1/2}{T}\right)$ <p>The block is suited to determine average switching losses of power semiconductors. To determine average conduction losses, use the Periodic Average (see page 343).</p>
Parameters	<p>Averaging time The length of the averaging period (in sec.) and the sample time of the output signal. See also the Discrete-Periodic sample time type in section “Sample Times” (on page 32).</p>
Probe Signals	<p>Input The input signal.</p> <p>Output The filtered output signal.</p>

Permanent Magnet Synchronous Machine

Purpose Synchronous machine excited by permanent magnets

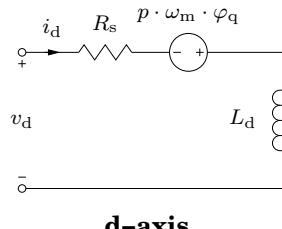
Library Electrical / Machines

Description This three-phase permanent magnet synchronous machine has a sinusoidal back EMF.

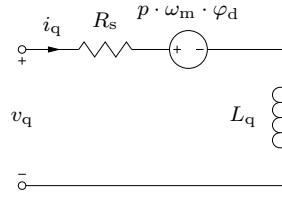


The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a is marked with a dot.

Electrical System



d-axis



q-axis

Stator flux linkages:

$$\varphi_q = L_q i_q$$

$$\varphi_d = L_d i_d + \varphi'_m$$

The machine model offers two different implementations of the electrical system: a traditional rotor reference frame and a voltage-behind-reactance formulation.

Rotor Reference Frame Using Park's transformation, the 3-phase circuit equations in physical variables are transformed to the dq rotor reference frame. This results in constant coefficients in the differential equations making the model numerically efficient. However, interfacing the dq model with the external 3-phase network may be difficult. Since the coordinate transformations are based on voltage-controlled current sources, inductors and naturally commutated devices such as diode rectifiers may not be directly connected to the stator terminals.

Voltage behind Reactance This formulation allows for direct interfacing of arbitrary external networks with the 3-phase stator terminals. The electrical system is described in circuit form. Due to the resulting time-varying inductance matrices, this implementation is numerically less efficient than the traditional rotor reference frame.

Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p (\varphi_d i_q - \varphi_q i_d)$$

Mechanical System

Mechanical rotor speed ω_m :

$$\begin{aligned}\dot{\omega}_m &= \frac{1}{J} (T_e - F\omega_m - T_m) \\ \dot{\theta}_m &= \omega_m\end{aligned}$$

Parameters

Model

Implementation in the rotor reference frame or as a voltage behind reactance.

Stator resistance

Armature or stator resistance R_s in Ω .

Stator inductance

A two-element vector containing the combined stator leakage and magnetizing inductance. L_d is referred to the d-axis and L_q to the q-axis of the rotor. The values are in henries (H).

Flux induced by magnets

Constant flux linkage φ'_m in Vs induced by the magnets in the stator windings.

Inertia

Combined rotor and load inertia J in Nms².

Friction coefficient

Viscous friction F in Nms.

Number of pole pairs

Number of pole pairs p .

Initial rotor speed

Initial mechanical rotor speed $\omega_{m,0}$ in radians per second (s⁻¹).

Initial rotor position

Initial mechanical rotor angle $\theta_{m,0}$ in radians.

Initial stator currents

A two-element vector containing the initial stator currents $i_{a,0}$ and $i_{b,0}$ of phase a and b in amperes (A).

Inputs and Outputs**Mechanical torque**

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector "m" contains the following 3 signals:**(1) Rotor speed**

The rotational speed ω_m of the rotor in radians per second (s⁻¹).

(2) Rotor position

The mechanical rotor angle θ_m in radians.

(3) Electrical torque

The electrical torque T_e of the machine in Nm.

Probe Signals**Stator phase currents**

The three-phase stator winding currents i_a , i_b and i_c , in A. Currents flowing into the machine are considered positive.

Stator flux (dq)

The stator flux linkages φ_d and φ_q in the stationary reference frame in Vs:

$$\begin{aligned}\varphi_q &= L_q i_q \\ \varphi_d &= L_d i_d + \varphi'_m\end{aligned}$$

Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

Rotor position

The mechanical rotor angle θ_m in radians.

Electrical torque

The electrical torque T_e of the machine in Nm.

See also

If the stator inductance is independent of the rotor angle, i.e. $L_d = L_q$, it is computational more efficient to use the simplified Brushless DC Machine (see page 207) with a sinusoidal back EMF. The parameters need to be converted as follows:

$$L - M = L_d = L_q$$

$$K_E = -\varphi'_m \cdot p$$

For back EMF shapes other than sinusoidal, and/or if the stator inductance has a complex angle dependency please use the sophisticated model of the Brushless DC Machine (see page 203). The sophisticated BLDC machine can be configured as a PMSM with sinusoidal back EMF if the parameters are converted as follows:

$$K_{c,n} = [0]$$

$$K_{s,n} = [-\varphi'_m \cdot p]$$

$$L_0 - M = \frac{L_d + L_q}{2}$$

$$L_{c,n} = [0 \ L_d - L_q]$$

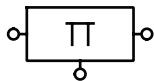
$$L_{s,n} = [0 \ 0]$$

Pi-Section Line

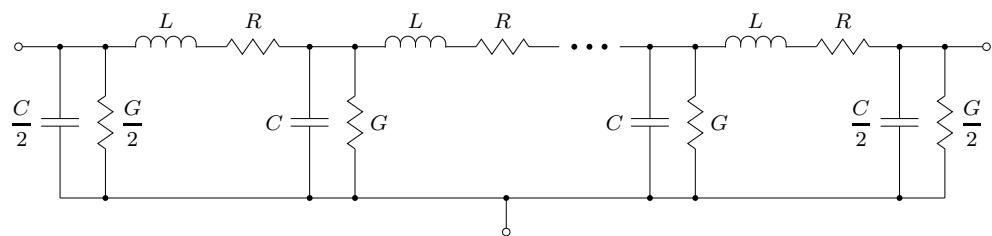
Purpose Single-phase pi-section transmission line

Library Electrical / Passive Components

Description The Pi-Section Line implements a single-phase transmission line with parameters lumped in pi sections.



A transmission line is characterized by a uniform distribution of inductance, resistance, capacitance and conductance along the line. However, in many cases these distributed parameters can be approximated by cascading multiple pi sections with discrete components. The figure below illustrates the electrical circuit used for the line model.



Let l be the length of the line and n the number of pi sections representing the line. The inductance L , the resistance R , the capacitance C and the conductance G of the discrete elements can then be calculated from their per-unit-length counterparts L' , R' , C' and G' using the following equations:

$$L = \frac{l}{n} L', \quad R = \frac{l}{n} R', \quad C = \frac{l}{n} C', \quad G = \frac{l}{n} G'$$

Parameters

Inductance per unit length

The series line inductance L' per unit length. If the length l is specified in meters (m) the unit of L' is henries per meter (H/m).

Resistance per unit length

The series line resistance R' per unit length. If the length l is specified in meters (m) the unit of R' is ohms per meter (Ω/m).

Capacitance per unit length

The capacitance C' between the line conductors per unit length. If the length l is specified in meters (m) the unit of C' is farads per meter (F/m).

Conductance per unit length

The conductance G' between the line conductors per unit length. If the length l is specified in meters (m) the unit of G' is siemens per meter (S/m).

Length

The length l of the line. The unit of l must match the units L' , R' , C' and G' are based on.

Number of pi sections

Number of sections used to model the transmission line. The default is 3.

Initial voltage

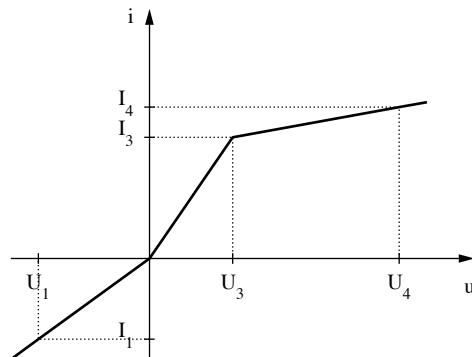
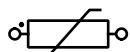
A scalar value specifying the initial voltage of all capacitors at simulation start, in volts (V).

Piece-wise Linear Resistor

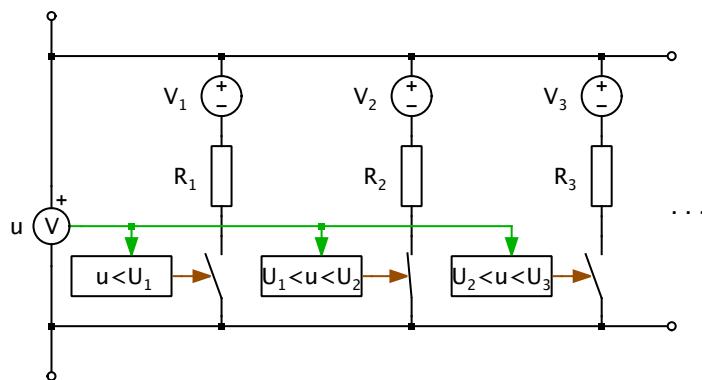
Purpose Resistance defined by voltage-current pairs

Library Electrical / Passive Components

Description This component models a piece-wise linear resistor. The resistance characteristic is defined by a set of voltage-current values.



The operating mode of the piece-wise linear resistor is illustrated in the diagram below. The voltage across the device dictates which internal switch is closed. The values 0 V / 0 A must always be defined in the set of voltage / current values to ensure the current is zero at zero voltage.



Note In order to model a saturation characteristic with n segments, this component requires n ideal switches. It is therefore advisable to keep the number of segments low in order to maintain a high simulation speed.

Parameters**Voltage values**

A vector of voltage values U in volts (V) that defines the piece-wise linear characteristic. The voltage values must be strictly monotonic increasing. At least two values are required. The value 0 must be present, the corresponding current value must also be 0.

Current values

A vector of current values I in amperes (A) that defines the piece-wise linear characteristic. The current values must be strictly monotonic increasing. The number of current values must match the number of voltage values. The value 0 must be present, the corresponding voltage value must also be 0.

Probe Signals**Resistor voltage drop**

The voltage measured across the component, in volts (V). The positive terminal of the resistor is marked with a small black dot.

Resistor current

The current flowing through the component, in amperes (A).

Resistor power

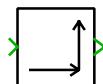
The power consumed by the resistor, in watts (W).

Polar to Rectangular

Purpose Convert polar coordinates to Cartesian coordinates

Library Control / Transformations

Description This block transforms a signal representing polar coordinates $[r, \theta]$ into rectangular coordinates $[x, y]$:



$$\begin{aligned} x &= r * \cos(\theta) && \text{where } \theta \text{ is in radians.} \\ y &= r * \sin(\theta) \end{aligned}$$

Product

Purpose	Multiply and divide scalar or vectorized input signals
Library	Control / Math
Description	<p>The Product block multiplies or divides input signals. If the division operator / is used, the reciprocal of the input signal is used for multiplication.</p> <p>In case of a single input, all elements of the input vector are multiplied. Vectorized input signals of the same width are multiplied element wise and result in a vectorized output signal. If vectorized and scalar input signals are mixed, the scalar input signals are expanded to the width of the vectorized input signals.</p>
Parameter	<p>List of operators or number of inputs</p> <p>The inputs can be specified either with</p> <ul style="list-style-type: none">• a string containing * or / for each input and for spacers, or• a positive integer declaring the number of inputs to be multiplied.
Probe Signals	<p>Input i The ith input signal.</p> <p>Output The block output signal.</p>

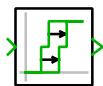
Pulse Delay

Purpose

Delay discrete-value input signal by fixed time

Library

Control / Delays

Description

The Pulse Delay applies a fixed-time delay to an input signal that changes at discrete instants and is otherwise constant. The output values are held constant between simulation time steps. The signal can be a scalar or vector. For delaying continuously changing signals please use the continuous Transport Delay (see page 450).

Parameters**Time delay**

Time by which the input signal is delayed.

Initial output

Output value after simulation start before the input values appear at the output.

Initial buffer size

Size of the internal ring buffer at simulation start. The buffer size will be increased during the simulation if required.

Pulse Generator

Purpose	Generate periodic rectangular pulses
Library	Control / Sources
Description	The Pulse Generator outputs a signal that periodically switches between a high- and low-state.
	
Parameter	High-state output The value of the output signal in the high-state. Low-state output The value of the output signal in the low-state. Frequency The frequency of the output signal in hertz (Hz). Duty cycle The fraction of the period length during which the output signal is in the high-state. The duty cycle value must be in the range [0 1]. For example, a value of 0.1 means that the signal is in the high-state for the first 10% of the period time. Phase delay The phase delay in seconds (s). If the phase delay is 0 the period begins at the start of the high state.
Probe Signals	Output The output signal of the pulse generator.

Quantizer

Purpose	Apply uniform quantization to input signal
Library	Control / Discontinuous
Description	The Quantizer maps the input signal to an integer multiple of the quantization interval:
	$y = q * \text{round} \left(\frac{u}{q} \right)$
Parameters	<p>Quantization interval The quantum q used in the mapping function.</p> <p>Step detection When set to on, the Quantizer produces a zero-crossing signal that enables the solver to detect the precise instants, at which the output needs to change. This may be necessary when quantizing a continuous signal. When set to off, the Quantizer will not influence the step size of the solver.</p>
Probe Signals	<p>Input The input signal.</p> <p>Output The output signal.</p>

Ramp

Purpose	Generate constantly rising or falling signal
Library	Control / Sources
Description	The Ramp block generates a signal that increases or decreases linearly over time once the start time is reached. The output can be limited to a final value.
	
Parameter	Slope The slope of the signal (per second). Start time The time at which the ramp starts. Initial output The output value before the start time is reached. Final output The final value for the output signal. If the parameter is set to <code>inf</code> the output signal is unlimited.
Probe Signals	Output The output signal of the pulse generator.

Rate Limiter

Purpose Limit rising and falling rate of change

Library Control / Discontinuous

Description The Rate Limiter restricts the first derivative of the signal passing through it. While the rate of change is within the specified limits, the output follows the input. When the rate of change exceeds the rising or falling limit, the output falls behind the input with a fixed slope until output and input become equal again.

Parameters

Rising rate limit

The maximum rate of change of the output signal (typically positive).

Falling rate limit

The minimum rate of change of the output signal (typically negative).

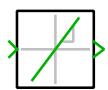
Probe Signals

Input

The input signal.

Output

The output signal.

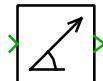


Rectangular to Polar

Purpose Convert Cartesian coordinates to polar coordinates

Library Control / Transformations

Description This block transforms a signal representing rectangular coordinates $[x, y]$ into polar coordinates $[r, \theta]$:



$$\begin{aligned} r &= \sqrt{x^2 + y^2} \\ \theta &= \text{atan2}(x, y) \end{aligned}$$

θ is calculated in the range $-\pi \leq \theta \leq \pi$.

Relational Operator

Purpose	Compare two input signals
Library	Control / Logical
Description	The Relational Operator compares two input signals. If the comparison is true it outputs 1, otherwise 0. The first input is marked with a dot.
	
Parameter	Relational operator Chooses which comparison operation is applied to the input signals. Available operators are <ul style="list-style-type: none"> • equal (<code>==</code>), • unequal (<code>~=</code>), • less (<code><</code>), • less or equal (<code><=</code>), • greater or equal (<code>>=</code>), • greater (<code>></code>).
Probe Signals	Input The input signals. Output The output signal.

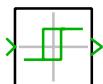
Relay

Purpose

Toggle between on- and off-state with configurable threshold

Library

Control / Discontinuous

Description

The output of the Relay block depends on its internal state. If the input signal exceeds the upper threshold, the relay will be in the on-state. It will be in the off-state if the inputs is less than the lower threshold. The relay does not change for input values between the thresholds.

Parameters**Upper threshold**

The highest value that the input signal may reach before the state changes to the on-state.

Lower threshold

The lowest value that the input signal may reach before the state changes to the off-state.

On-state output

The value of the output signal while the relay is in the on-state.

Off-state output

The value of the output signal while the relay is in the off-state.

Initial state

The state of the relay at simulation start. Possible values are on and off.

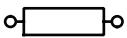
Probe Signals**Input**

The block input signal.

Output

The block output signal.

Resistor

Purpose	Ideal resistor
Library	Electrical / Passive Components
Description	<p>This component provides an ideal resistor between its two electrical terminals. See section “Configuring PLECS” (on page 35) for information on how to change the graphical representation of resistors.</p> <hr/>
	 
	<p>Note Like all other parameters of PLECS components, the resistance cannot be changed during the simulation.</p> <hr/>
Parameter	<p>Resistance The resistance in ohms (Ω). All positive and negative values are accepted, including 0 and inf (∞). The default is 1.</p> <p>In a vectorized component, all internal resistors have the same resistance if the parameter is a scalar. To specify the resistances individually use a vector $[R_1 \ R_2 \dots R_n]$. The length n of the vector determines the width of the component.</p>
Probe Signals	<p>When the resistor is probed, a small dot in the component icon marks the positive terminal.</p> <p>Resistor voltage The voltage measured across the resistor from the positive to the negative terminal, in volts (V).</p> <p>Resistor current The current flowing through the resistor, in amperes (A). A current entering the resistor at the positive terminal is counted positive.</p> <p>Resistor power The power consumed by the resistor, in watts (W).</p>

Rounding

Purpose	Round floating point signal to integer values
Library	Control / Math
Description	This component rounds the value of a floating point signal on its input to an integer value. The rounding algorithm can be selected in the component parameter: 
	floor The output is the largest integer not greater than the input, for example $\text{floor}(1.7) = 1$ and $\text{floor}(-1.3) = -2$.
	ceil The output is the smallest integer not less than the input, for example $\text{ceil}(1.3) = 2$ and $\text{ceil}(-1.7) = -1$.
	round The output is the integer nearest to the input, for example $\text{round}(1.4) = 1$, $\text{round}(-1.3) = -1$ and $\text{round}(-1.5) = -2$.
	fixed The output is the integer value of the input with all decimal places truncated, for example $\text{fixed}(1.7) = 1$ and $\text{fixed}(-1.7) = -1$.
Parameter	Operation The rounding algorithm as described above.
Probe Signals	Input The block input signal. Output The block output signal.

Saturable Capacitor

Purpose Capacitor with piece-wise linear saturation

Library Electrical / Passive Components

Description This component provides a saturable capacitor between its two electrical terminals. The capacitor has a symmetrical piece-wise linear saturation characteristic defined by positive voltage/charge pairs.



Note In order to model a saturation characteristic with n segments, this component requires n ideal capacitors and $2(n - 1)$ ideal switches. It is therefore advisable use as few segments as possible.

Parameters

Voltage values

A vector of positive voltage values in volts (V) defining the piece-wise linear saturation characteristic. The voltage values must be positive and strictly monotonic increasing. At least one value is required.

Charge values

A vector of positive charge values in As defining the piece-wise linear saturation characteristic. The charge values must be positive and strictly monotonic increasing. The number of charge values must match the number of voltage values.

Initial voltage

The initial voltage across the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The positive pole is marked with a “+”. The initial voltage default is 0.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with “+” is greater than the potential at the unmarked terminal.

Capacitor current

The current flowing through the capacitor, in amperes (A).

Saturation level

The saturation level indicates which sector of the piece-wise linear characteristic is currently applied. During linear operation, i.e. operation in the

first sector, the saturation level is 0. The saturation level is negative for negative charge and voltage values.

Saturable Core

Purpose Magnetic core element with saturation

Library Magnetic

Description This component models a segment of a magnetic core. It establishes a non-linear relationship between the magnetic field strength H and the flux density B to model saturation effects. The user can choose between the following fitting functions:



atan fit

The atan fit is based on the arctangent function:

$$B = \frac{2}{\pi} B_{\text{sat}} \tan^{-1} \left(\frac{\pi H}{2a} \right) + \mu_{\text{sat}} H$$

coth fit

The coth fit was adapted from the Langevin equation for bulk magnetization without interdomain coupling, and is given as:

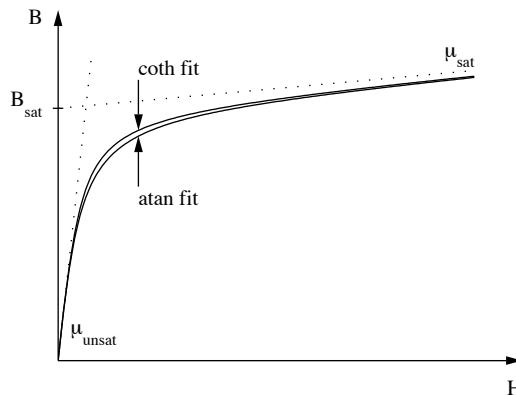
$$B = B_{\text{sat}} \left(\coth \frac{3H}{a} - \frac{a}{3H} \right) + \mu_{\text{sat}} H$$

Both fitting functions have three degrees of freedom which are set by the coefficients μ_{sat} , B_{sat} and a . μ_{sat} is the fully saturated permeability, which usually corresponds to the magnetic constant μ_0 , i.e. the permeability of air. B_{sat} defines the knee of the saturation transition between unsaturated and saturated permeability:

$$B_{\text{sat}} = (B - \mu_{\text{sat}} H) \Big|_{H \rightarrow \infty}$$

The coefficient a is determined by the unsaturated permeability μ_{unsat} at $H = 0$:

$$a = B_{\text{sat}} / (\mu_{\text{unsat}} - \mu_{\text{sat}})$$



The figure below illustrates the saturation characteristics for both fitting functions. The saturation curves differ only around the transition between unsaturated and saturated permeability. The coth fit expresses a slightly tighter transition than the atan fit.

Parameters

Fitting functions

Saturation characteristic modeled with atan or coth fit.

Cross-sectional area

Cross-sectional area A of the flux path, in m^2 .

Length of flux path

Length l of the flux path, in m.

Unsaturated rel. permeability

Relative permeability $\mu_{\text{r,unsat}} = \mu_{\text{unsat}}/\mu_0$ of the core material for $H \rightarrow 0$.

Saturated rel. permeability

Relative permeability $\mu_{\text{r,sat}} = \mu_{\text{sat}}/\mu_0$ of the core material for $H \rightarrow \infty$.

Flux density saturation

Knee B_{sat} of the saturation transition between unsaturated and saturated permeability.

Initial MMF

Magneto-motive force at simulation start, in ampere-turns (A).

Probe Signals

MMF

The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).

Flux

The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.

Field strength

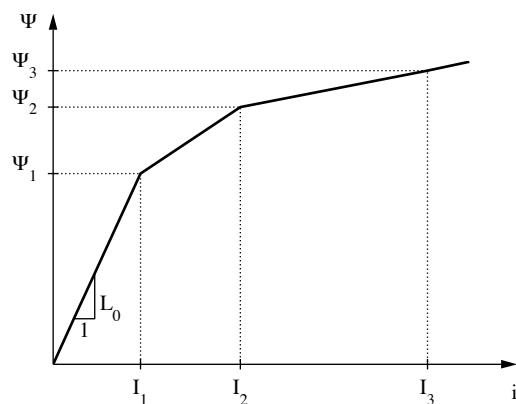
The magnetic field strength H in the core element, in A/m.

Flux density

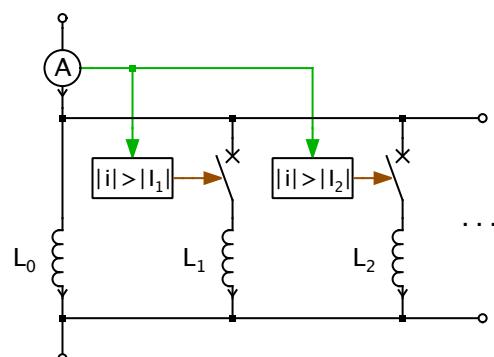
The magnetic flux density B in the core element, in teslas (T).

Saturable Inductor

Purpose	Inductor with piece-wise linear saturation
Library	Electrical / Passive Components
Description	This component provides a saturable inductor between its two electrical terminals. The inductor has a symmetrical piece-wise linear saturation characteristic defined by positive current/flux pairs.
	



The operating mode of the saturable inductor is illustrated in the schematic below. In the unsaturated state the current flows only through the main inductor L_0 . When the absolute value of the current exceeds the threshold I_1 ,



the breaker in series with the auxiliary inductor L_1 is closed. The *differential* inductivity of the component thus becomes $L_{\text{diff}} = \frac{L_0 L_1}{L_0 + L_1}$. On the other hand, the *total* inductivity is calculated as $L_{\text{tot}} = \frac{L_0 i_0 + L_1 i_1}{i_0 + i_1}$, where i_0 and i_1 are the momentary inductor currents.

Note In order to model a saturation characteristic with n segments, this component requires n ideal inductors and $2(n - 1)$ ideal switches. It is therefore advisable use as few segments as possible.

Parameters

Current values

A vector of positive current values I in amperes (A) defining the piece-wise linear saturation characteristic. The current values must be positive and strictly monotonic increasing. At least one value is required.

Flux values

A vector of positive flux values Ψ in Vs defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Initial current

The initial current through the inductor at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The direction of a positive initial current is indicated by a small arrow at one of the terminals. The initial current default is 0.

Probe Signals

Inductor current

The current flowing through the inductor, in amperes (A). The direction of a positive current is indicated with a small arrow at one of the terminals.

Inductor voltage

The voltage measured across the inductor, in volts (V).

Saturation level

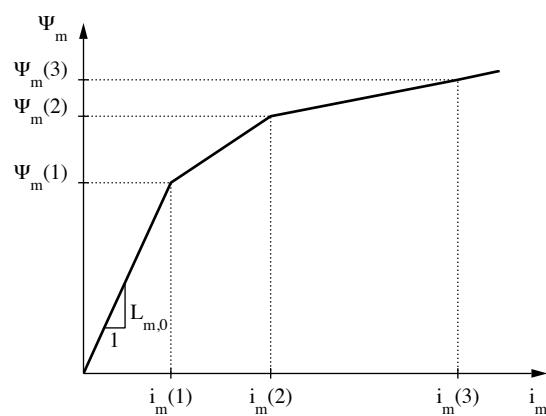
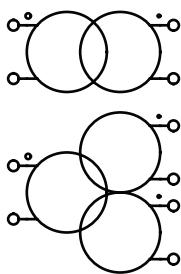
The saturation level indicates which sector of the piece-wise linear characteristic is currently applied. During linear operation, i.e. operation in the first sector, the saturation level is 0. The saturation level is negative for negative flux and current values.

Saturable Transformers

Purpose Single-phase transformers with two resp. three windings and core saturation

Library Electrical / Passive Components

Description These transformers model two or three coupled windings on the same core.



The core saturation characteristic is piece-wise linear and is modeled using the Saturable Inductor (see page 370). The magnetizing current i_m and flux Ψ_m value pairs are referred to the primary side. To model a transformer without saturation enter 1 as the magnetizing current values and the desired magnetizing inductance L_m as the flux values. A stiff Simulink solver is recommended if the iron losses are not negligible, i.e. R_{fe} is not infinite.

In the transformer symbol, the primary side winding is marked with a little circle. The secondary winding is marked with a dot at the outside terminal, the tertiary winding with a dot at the inside terminal.

Parameters

Leakage inductance

A vector containing the leakage inductance of the primary side L_1 , the secondary side L_2 and, if applicable, the tertiary side L_3 . The inductivity is given in henries (H).

Winding resistance

A vector containing the resistance of the primary winding R_1 , the secondary winding R_2 and, if applicable, the tertiary winding R_3 , in ohms (Ω).

No. of turns

A vector containing the number of turns of the primary winding n_1 , the secondary winding n_2 and the tertiary winding n_3 , if applicable.

Magnetizing current values

A vector of positive current values in amperes (A) defining the piece-wise linear saturation characteristic of the transformer legs. The current values must be positive and strictly monotonic increasing. At least one value is required.

Magnetizing flux values

A vector of positive flux values in Vs defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Core loss resistance

An equivalent resistance R_{fe} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

Initial current

A vector containing the initial currents on the primary side i_1 , the secondary side i_2 and the tertiary side i_3 , if applicable. The currents are given in amperes (A) and considered positive if flowing into the transformer at the marked terminals. The default is [0 0 0].

Saturation

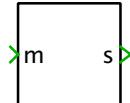
Purpose	Limit input signal to upper and/or lower value
Library	Control / Discontinuous
Description	The saturation block limits a signal to an upper and/or lower value. If the input signal is within the saturation limits the output signal is identical to the input signal.
	
Parameters	Upper limit The highest value that the input signal may reach before the output signal is clipped. If the value is set to <code>inf</code> the output is unlimited. Lower limit The lowest value that the input signal may reach before the output signal is clipped. If the value is set to <code>inf</code> the output is unlimited.
Probe Signals	Input The block input signal. Output The block output signal.

Sawtooth PWM

Purpose Generate PWM signal using sawtooth carrier

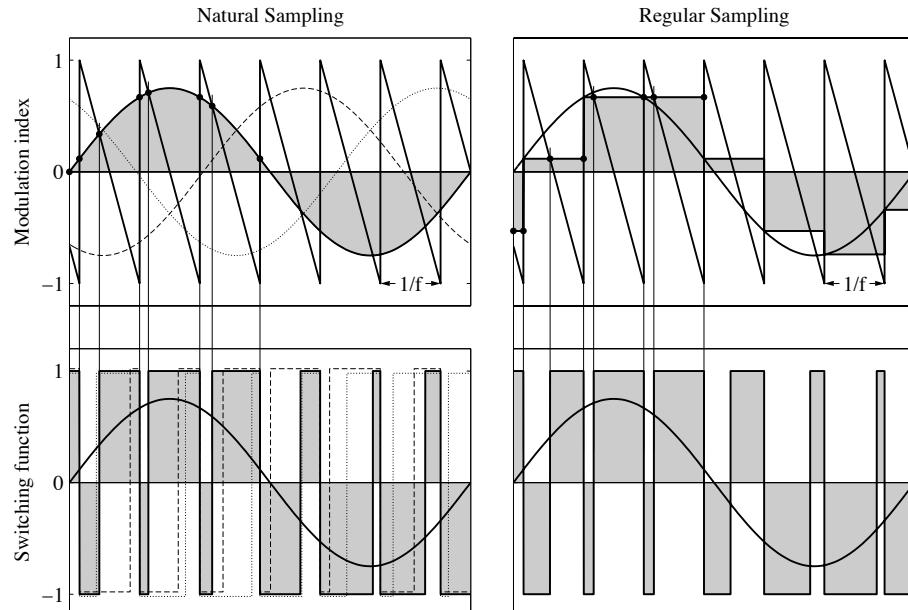
Library Control / Modulators

Description 2-level PWM generator with a sawtooth carrier. The input m is the modulation index, and the output s is the switching function. If the modulation index is a vector the switching function is also a vector of the same width.



The block can be used to control the IGBT Converter (see page 273) or the ideal Converter (see page 267). In these cases the modulation index must have a width of 3 to match the number of inverter legs.

The following figures illustrate different sampling methods offered by the modulator block. In the figure on the left, Natural Sampling is used. The right figure shows Regular Sampling, i.e. the modulation index is updated at the vertical flanks of the carrier. In both figures carrier signals with falling ramps are employed.



Parameters**Sampling**

Choose between Natural and Regular Sampling.

Ramp

Choose between rising and falling ramps in the carrier signal.

Carrier frequency

The frequency f of the carrier signal, in Hz.

Carrier offset

The time offset of the carrier signal, in p.u. of the carrier period.

Input limits

The range of the modulation index. The default is [-1 1].

Output values

Values of the switching function in off-state and on-state. The default is [-1 1].

Sawtooth PWM (3-Level)

Purpose

Generate 3-level PWM signal using sawtooth carriers

Library

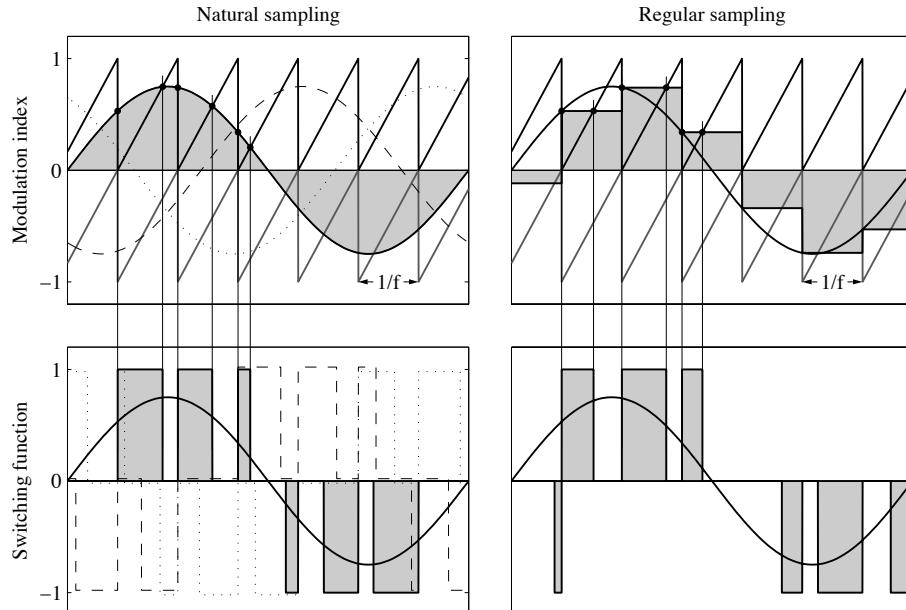
Control / Modulators

Description


3-level PWM generator with a sawtooth carrier. The input m is the modulation index. The switching function s outputs either 1, 0 or -1. If the modulation index is a vector the switching function is also a vector of the same width.

The block can be used to control the 3-Level IGBT Converter (see page 271) or the ideal 3-Level Converter (see page 266). In these cases the modulation index must have a width of 3 to match the number of inverter legs.

The figures below illustrate different sampling methods offered by the modulator block. In the left figure, Natural Sampling is used. The right figure shows Regular Sampling, i.e. the modulation index is updated at the vertical flanks of the carrier. In both figures carrier signals with rising ramps are employed.



Parameters**Sampling**

Choose between Natural and Regular Sampling.

Ramp

Choose between rising and falling ramps in the carrier signal.

Carrier frequency

The frequency f of the carrier signal, in Hz.

Carrier offset

The time offset of the carrier signal, in p.u. of the carrier period.

Input limits

The range of the modulation index. The default is [-1 1].

Scope

Purpose	Display simulation results versus time
Library	System
Description	The PLECS scope displays the measured signals of a simulation. It can be used in PLECS circuits as well as in Simulink models.
 	A number of analysis tools and data display options allow detailed analysis of the measured signals. For more information on how to work with the scope see section “Using the PLECS Scope” (on page 62).
Parameters	<p>Number of plots This parameter specifies the number of plots shown in the scope window. Each plot corresponds to a terminal on the outside of the block. For each plot, a tab is displayed in the lower part of the dialog where the plot settings can be edited.</p> <p>Display time axis The time axis is either shown underneath each plot or underneath the last plot only.</p> <p>Time axis label The time axis label is shown below the time axis in the scope.</p> <p>Limit samples If this option is selected, the PLECS scope will only save the last n sample values during a simulation. It can be used in long simulations to limit the amount of memory that is used by PLECS. If the option is unchecked all sample values are stored in memory.</p> <p>Time range The time range value determines the initial time range that is displayed in the scope. If set to auto, the simulation time range is used.</p> <p>The following items can be set for each plot independently:</p> <p>Title The name which is displayed above the plot.</p> <p>Axis label The axis label is displayed on the left of the y-axis.</p> <p>Y-limits The initial lower and upper bound of the y-axis. If set to auto, the y-axis is automatically scaled such that all data is visible.</p>

Set/Reset Switch

Purpose	Bistable on-off switch								
Library	Electrical / Switches								
Description	<p>This component provides an ideal short or open circuit between its two electrical terminals. The switch closes when the closing signal (the upper input in the component icon) becomes non-zero. It opens when the opening signal (the lower input) becomes non-zero. The Set/Reset Switch provides the basis for all other switches and power semiconductor models in PLECS.</p> 								
Parameters	<p>Initial conductivity Initial conduction state of the switch. The switch is initially open if the parameter evaluates to zero, otherwise closed. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.</p> <p>Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79).</p> <p>Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start. This parameter may either be a scalar or a vector corresponding to the implicit width of the component.</p> <p>Probe Signals</p> <table border="0"> <tr> <td>Switch conductivity</td> <td>Conduction state of the switch. The signal outputs 0 if the switch is open, and 1 if it is closed.</td> </tr> <tr> <td>Switch temperature</td> <td>Temperature of the first thermal capacitor in the equivalent Cauer network.</td> </tr> <tr> <td>Switch conduction loss</td> <td>Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.</td> </tr> <tr> <td>Switch switching loss</td> <td>Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.</td> </tr> </table>	Switch conductivity	Conduction state of the switch. The signal outputs 0 if the switch is open, and 1 if it is closed.	Switch temperature	Temperature of the first thermal capacitor in the equivalent Cauer network.	Switch conduction loss	Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.	Switch switching loss	Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.
Switch conductivity	Conduction state of the switch. The signal outputs 0 if the switch is open, and 1 if it is closed.								
Switch temperature	Temperature of the first thermal capacitor in the equivalent Cauer network.								
Switch conduction loss	Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.								
Switch switching loss	Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.								

Signal Demultiplexer

Purpose	Split vectorized signal
Library	System
Description	This demultiplexer extracts the components of a input signal and outputs them as separate signals. The output signals may be scalars or vectors. In the block icon, the first output is marked with a dot.
	
Parameter	Number of outputs This parameter allows you to specify the number and width of the output signals. You can choose between the following formats for this parameter: Scalar: A scalar specifies the number of scalar outputs. If this format is used all output signals have a width of 1. Vector: The length of the vector determines the number of outputs. Each element specifies the width of the corresponding output signal.

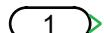
Signal From

Purpose	Reference signal from Signal Goto block by name
Library	System
Description	The Signal From block references another signal from a Signal Goto block. All Signal From blocks connect to the Signal Goto block with the same tag within the given scope. If no matching Signal Goto block is found an error message will be displayed when starting a simulation.
tag 	
Parameter	Tag name: The tag names of the Signal From and Signal Goto blocks must match to establish a connection. Scope: The scope specifies the search depth for the matching Signal Goto block. Using the value Global the complete PLECS circuit is searched. When set to Schematic only the schematic containing the Signal From block is searched. The setting Masked Subsystem causes a lookup within the hierarchy of the masked subcircuit in which the block is contained. If the block is not contained in a masked subsystem a global lookup is done.

Signal Goto

Purpose	Make signal available by name
Library	System
Description	The Signal Goto block forwards its input signal to a number of Signal From blocks within the same scope. All Signal From blocks connect to the Signal Goto block with the same tag within the given scope. It is not allowed to have multiple Signal From blocks with the same tag name within the same scope.
 tag	
Parameter	Tag name: The tag names of the Signal From and Signal Goto blocks must match to establish a connection. Scope: The scope specifies the search depth for the matching Signal From blocks. Using the value Global the complete PLECS circuit is searched. When set to Schematic only the schematic containing the Signal From block is searched. The setting Masked Subsystem causes a lookup within the hierarchy of the masked subcircuit in which the block is contained. If the block is not contained in a masked subsystem a global lookup is done.

Signal Import

Purpose	Add signal input connector to subsystem
Library	System
Description  1	Imports are used to feed signals from a schematic into a subschematic. In the PLECS Blockset, imports are also used to feed signals from a Simulink model into a PLECS circuit. If you copy an input block into a schematic an input terminal will be created on the corresponding subsystem block. The name of the input block will appear as the terminal label. If you choose to hide the block name by unselecting the show button in the dialog box the terminal label will also disappear.

Input Blocks in a Top-Level Circuit

If an input block is placed in a top-level schematic a unique number is assigned to the block. In Simulink, the relative position of the corresponding input terminals is determined by the order of block numbers. You may change the block number in order to change the relative terminal position.

For top-level inputs you can also specify whether the input signal is used as a *continuous signal* in order to control e.g. sources or as a discrete *gate signal* in order to feed control the gate of a switch or semiconductor. Continuous signal inputs have *direct feedthrough* which can lead to algebraic loops if there is a direct path from a circuit output to a (continuous) circuit input. In contrast, gate signal inputs *do not* have direct feedthrough. However, they are expected to change only at discrete instants. Using a gate signal input to feed a continuous signal into a circuit can lead to unexpected results. The standard setting auto causes PLECS to determine the signal type based on the internal connectivity.

Input Blocks in a Subsystem

If placed in a subschematic the inputs are not identified by numbers since terminals on subsystem blocks can be freely positioned. Which terminal corresponds to which input block can only be seen from the block name. In order to move a terminal with the mouse around the edges of a subsystem block hold down the **Shift** key while dragging the terminal with the left mouse button or use the middle mouse button.

Parameters**Width**

The width of the input signal. The default auto means that the width is inherited from connected blocks.

Signal type

The input signal type (see the description above). This parameter appears only in the PLECS Blockset if the block is placed in a top-level schematic.

Port number

The terminal number of the input block. This parameter appears only in the PLECS Blockset if the block is placed in a top-level schematic.

Signal Multiplexer

Purpose	Combine several signals into vectorized signal
Library	System
Description	This multiplexer combines several signals into a vectorized signal. The input signals may be scalars or vectors. In the block icon, the first input is marked with a dot.
	
Parameter	Number of inputs This parameter allows you to specify the number and width of the input signals. You can choose between the following formats for this parameter: Scalar: A scalar specifies the number of scalar inputs to the block. If this format is used the block accepts only signals with a width of 1. Vector: The length of the vector determines the number of inputs. Each element specifies the width of the corresponding input signal.

Signal Outport

Purpose	Add signal output connector to subsystem
Library	System
Description	Outputs are used to feed signals from a PLECS circuit back to Simulink or from a subschematic to the parent schematic. If you copy an output block into a schematic an output terminal will be created on the corresponding subsystem block. The name of the output block will appear as the terminal label. If you choose to hide the block name by unselecting the show button in the dialog box the terminal name will also disappear.
 1	

Output Blocks in a Top-Level Circuit

If an output block is placed in a top-level circuit a unique number is assigned to the block. In Simulink, the relative position of the corresponding input terminals is determined by the order of block numbers. You may change the block number in order to change the relative terminal position.

Output Blocks in a Subsystem

If placed in a subschematic the outputs are not identified by numbers since terminals on subsystem blocks can be freely positioned. Which terminal corresponds to which output block can only be seen from the block name. In order to move a terminal with the mouse around the edges of a subsystem hold down the **Shift** key while dragging the terminal with the left mouse button or use the middle mouse button.

Parameters

Width

The width of the output signal. The default auto means that the width is inherited from connected blocks.

Port number

The terminal number of the output block. This parameter appears only if the block is placed in a top-level circuit.

Signal Selector

Purpose Select or reorder elements from vectorized signal

Library System

Description The Signal Selector block generates an output vector signal that consists of the specified elements of the input vector signal.



Parameters

Input width

The width of the input signal vector.

Output indices

A vector with the indices of the input elements that the output vector should contain.

Signal Switch

Purpose

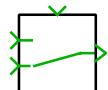
Select one of two input signals depending on control signal

Library

Control / Discontinuous

Description

While the Signal Switch is in the off-state the output is connected to the input terminal indicated in the icon. When the switch criteria is met, the switch changes to the on-state and the output is connected to the opposite input terminal.



Parameters

Criteria

The switch criteria which has to be met to put the switch in the on-state.

Available choices are

- $u \geq \text{Threshold}$,
- $u > \text{Threshold}$ and
- $u \approx \text{Threshold}$.

Threshold

The threshold value used for the switch criteria.

Probe Signals

Inputs

The block input signals.

Output

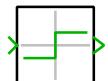
The block output signal.

Switch position

The state of the switch. The output is 0 while the switch is in the off-state and 1 while it is in the on-state.

Signum

Purpose	Provide sign of input signal
Library	Control / Math
Description	The Signum block outputs 1 for positive, -1 for negative and 0 for 0 input values.



Probe Signals	Input The block input signal. Output The block output signal.
----------------------	--

Sine Wave

Purpose	Generate time-based sine wave with optional bias
Library	Control / Sources
Description	The Sine Wave block generates a sinusoidal output signal with an optional bias.
	
Parameters	<p>Amplitude The amplitude (peak to peak) of the output signal.</p> <p>Bias An offset that is added to the sine wave signal.</p> <p>Frequency The frequency of the sine wave in hertz or rad/second (see below).</p> <p>Phase The phase of the sine wave in rad, per unit (p.u.) or degrees (see below). The parameter value should be in the range $[0 \ 2\pi]$, $[0 \ 1]$ or $[0 \ 360]$ respectively.</p> <p>Units for frequency and phase The frequency and phase can be expressed in terms of (rad/sec, rad), (Hz, p.u.) or (Hz, degrees). If the phase is expressed in per unit (p.u.), a value of 1 is equivalent to the period length.</p>
Probe Signals	Output The block output signal.

Small Signal Gain

This block is included only in the PLECS Standalone library.

Purpose	Measure loop gain of closed control loop using small signal analysis
Library	Control / Small Signal Analysis
Description	This block uses the Small Signal Perturbation block (see page 393) and the Small Signal Response block (see page 394) to inject a perturbation into a feedback loop and measure the system response. To see the implementation choose Look under mask from the context menu of the block. For detailed information regarding small signal analysis see chapter “Analysis Tools” (on page 105).
 Parameter	Compensate for negative feedback When set to on, the underlying Small Signal Response block inverts the reference input in order to compensate for a negative unity gain that is introduced when the feedback signal is <i>subtracted</i> from a reference signal. When set to off, the reference input is taken as is.

Small Signal Perturbation

This block is included only in the PLECS Standalone library.

Purpose

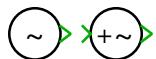
Generate perturbation signal for small signal analysis

Library

Control / Small Signal Analysis

Description

During a small-signal analysis that references this block, it generates the appropriate perturbation signal: a sinusoidal signal for an AC Sweep and a discrete pulse for an Impulse Response Analysis. At all other times the perturbation is zero.



For detailed information regarding small signal analysis see chapter “Analysis Tools” (on page 105).

Parameters**Show feed-through input**

When set to on, the block displays an input port. The output signal is the sum of the input signal and the perturbation. The default is off.

Small Signal Response

This block is included only in the PLECS Standalone library.

Purpose

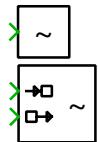
Measure system response for small signal analysis

Library

Control / Small Signal Analysis

Description

During a small-signal analysis that references this block, it records the signal(s) that are connected to the block input(s) in order to calculate the transfer function



$$G(s) = \frac{Y(s)}{U(s)}$$

If the reference input is shown, $U(s)$ is calculated from the signal that is connected to it. Otherwise, $U(s)$ is calculated from the perturbation signal generated by the corresponding Small Signal Perturbation block (see page 393).

For detailed information regarding small signal analysis see chapter “Analysis Tools” (on page 105).

Parameters

Show reference input

Specifies whether or not the block shows the reference input port.

Invert reference input

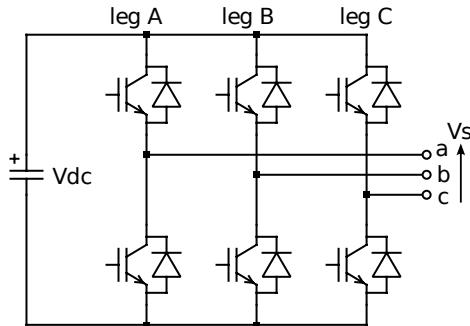
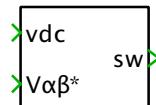
Specifies whether or not the reference input signal is inverted, i.e. multiplied with -1.

Space Vector Modulator

Purpose Generate PWM signals for 3-phase inverter using space-vector modulation technique

Library Control / Modulators

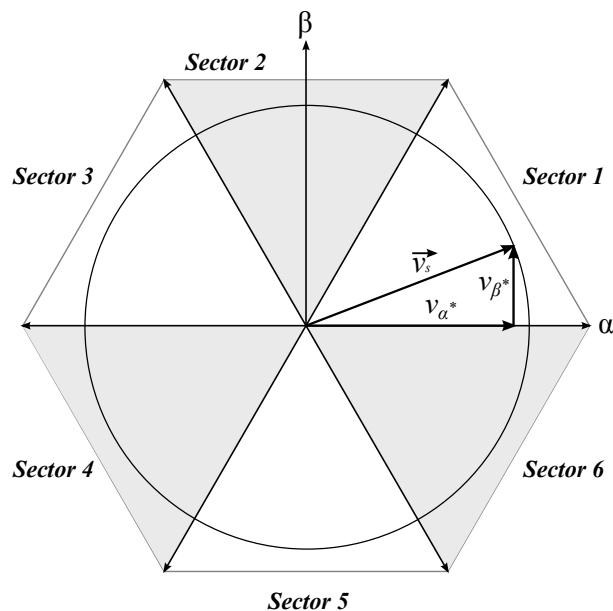
Description The space vector modulator generates a reference voltage vector, \vec{V}_s , at the ac terminals of a three phase voltage source converter shown below. The reference vector is defined in the $\alpha\beta$ coordinate system: $\vec{V}_s = V_\alpha^* + j V_\beta^*$.



Operation The construction of the reference voltage vector, \vec{V}_s , is graphically depicted below. Internally, the space vector modulator consists of a sector detection and vector timing calculation function that is executed at the beginning of the switching cycle. In this function, the operating sector and relative on-times of the switching vectors are calculated. During a switching cycle, a vector generation and sequencing function is called at the switching instants to update the switch output.

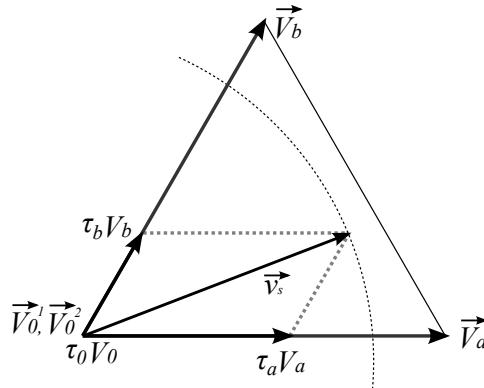
The sector detection calculation determines the sector in which the reference voltage vector \vec{V}_s resides. The relative on-times, τ_a, τ_b, τ_0 , for the switching vectors \vec{V}_a, \vec{V}_b and \vec{V}_0 are then calculated. In each sector, two unique switching vectors named \vec{V}_a and \vec{V}_b are available. Two zero vectors, named \vec{V}_0^1, \vec{V}_0^2 are also available. The relationship between the relative on-times and the reference vector is shown below for an arbitrary sector. The relative on-times are calculated by projecting the reference vector onto the vectors \vec{V}_a and \vec{V}_b .

The vector generation and sequencing function creates a switching cycle by time-averaging the switching vectors according to their on-time values. There



Construction of the reference vector \vec{V}_s .

are many possible switching sequences that can be implemented since the order in which the vectors \vec{V}_a , \vec{V}_b , \vec{V}_0 are applied during a switching cycle is arbitrary. In addition, one or both of the \vec{V}_0 vectors can be used. For further information, please read the documentation that accompanies the demo model "Space Vector Control of a Three Phase Rectifier using PLECS." This documentation can be found at <http://www.plexim.com/examples>.



Relationship between relative on times, τ_a, τ_b, τ_0 , switching vectors, $\vec{V}_a, \vec{V}_b, \vec{V}_0$, and reference vector, \vec{V}_s .

Parameters

Modulation strategy

The modulation strategy can be set to 'Alternating zero vector' or 'Symmetrical' using a combo box. With alternating zero vector modulation, only one of the two \vec{V}_0 switching vectors is used during a switching sequence. One switch leg is always clamped to the positive or negative dc bus voltage and only two of the three converter legs are switched.

With symmetrical modulation, the two \vec{V}_0 switching vectors are used: one at the beginning and one at the middle of a switching sequence. All three converter legs are switched during a switching sequence.

Switching frequency

The switching frequency in Hz.

Switch output values

The switch output values in the high and low state. The values should be selected to match the converter's gate control logic so that a high value turns on the upper switch in the leg and the low value turns on the lower switch. The default values are $[-1 1]$.

Inputs and Outputs

DC voltage

The input signal V_{dc} is the voltage measured on the dc side of the converter.

Theta

The angular position of the d axis in radians. The value supplied must

conform to: $\theta \geq 0$. If θ exceeds 2π , θ is shifted into the range $[0..2\pi]$ using a modulus function.

Reference voltage

This input, labelled $V_{\alpha\beta}^*$, is a two dimensional vector signal comprising the elements $[V_\alpha^*, V_\beta^*]$.

Switch output

The output labelled sw is formed from three switch control signals, $[S_a, S_b, S_c]$, which control the converter legs A, B, and C. Each switch signal controls the upper and lower switches in the respective leg.

Probe Signals

sector

A value in the set of [1..6] that indicates the sector in which the reference vector, \vec{V}_s , is located.

tau

A vector signal comprising the three relative on time values, $[\tau_a, \tau_b, \tau_0]$.

sw

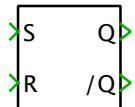
A vector signal consisting of the three gate signals for the converter legs, $[S_a, S_b, S_c]$.

SR Flip-flop

Purpose Implement set-reset flip-flop

Library Control / Logical

Description The SR Flip-flop behaves like a pair of cross-coupled NOR logic gates. The output values correspond to the following truth table:



S	R	Q	/Q
0	0	No change	No change
0	1	0	1
1	0	1	0
1	1	Restricted (0)	Restricted (0)

The combination $S = R = 1$ is restricted because both outputs will be set to 0, violating the condition $Q = \text{not}(/Q)$. If both inputs change from 1 to 0 in the same simulation step, Q will be set to 0 and $/Q$ to 1.

Parameters **Initial state**
The state of the flip-flop at simulation start.

Probe Signals **S**
The input signal S .
R
The input signal R .
Q
The output signals Q .
/Q
The output signals $/Q$.

State Space

Purpose Implement linear time-invariant system as state-space model

Library Control / Continuous

Description The State Space block models a state space system of the form $\dot{x} = Ax + Bu, y = Cx + Du$, where x is the state vector, u is the input vector, and y is the output vector.

$$\begin{array}{l} \boxed{x' = Ax + Bu} \\ \boxed{y = Cx + Du} \end{array}$$

Parameters

A,B,C,D

The coefficient matrices for the state space system. The dimensions for the coefficient matrices must conform to the dimensions shown in the diagram below:

$$\begin{matrix} & n & & m \\ n & \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right] \\ p & \end{matrix}$$

where n is the number of states, m is the width of the input signal and p is the width of the output signal.

Initial condition

A vector of initial values for the state vector, x .

Probe Signals

Input

The input vector, u .

Output

The output vector, y .

Step

Purpose Output a signal step.

Library Control / Sources

Description The Step block generates an output signal that changes its value at a given point in time.



Parameters

Step time

The time at which the output signal changes its value.

Initial output

The value of the output signal before the step time is reached.

Final output

The value of the output signal after the step time is reached.

Probe Signals

Output

The block output signal.

Subsystem

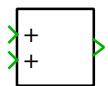
Purpose	Create functional entity in hierarchical simulation model
Library	System
Description	A subsystem block represents a system within another system. In order to create a subsystem, copy the subsystem block from the library into your schematic. You can then open the subsystem block and copy components into the subsystem's window. The input, output, and electrical terminals on the block icon correspond to the input, output, and electrical port blocks in the subsystem's schematic. If the block names are not hidden, they appear as terminal labels on the subsystem block. You can move terminals with the mouse around the edges of the subsystem by holding down the Shift key while dragging them with the left mouse button or by using the middle mouse button.
	
Parameters	You can create a dialog box for your Subsystem by masking the block (see “Mask Parameters” (on page 51) for more details).

Sum

Purpose Add and subtract input signals

Library Control / Math

Description



The Sum block adds or subtracts input signals. In case of a single input, all elements of the input vector are summed or subtracted. Vectorized input signals of the same width are added or subtracted element wise and result in a vectorized output signal. If vectorized and scalar input signals are mixed, the scalar input signals are expanded to the width of the vectorized input signals.

Parameters

Icon shape

Specifies whether the block is drawn with a round or a rectangular shape.
Round shape icons permit a maximum of three inputs.

List of signs or number of inputs

The inputs can be specified either with

- a string containing + or - for each input and | for spacers, or
- a positive integer declaring the number of summands.

Probe Signals

Input *i*

The *i*th input signal.

Output

The block output signal.

Switch

Purpose	On-off switch
Library	Electrical / Switches
Description	This Switch provides an ideal short or open circuit between its two electrical terminals. The switch is open when the input signal is zero, otherwise closed. 
Parameter	Initial conductivity Initial conduction state of the switch. The switch is initially open if the parameter evaluates to zero, otherwise closed. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.
Probe Signals	Switch conductivity Conduction state of the switch. The signal outputs 0 if the switch is open, and 1 if it is closed.

Switched Reluctance Machine

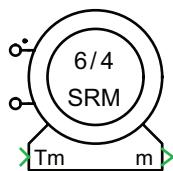
Purpose

Detailed model of switched reluctance machine with open windings

Library

Electrical / Machines

Description

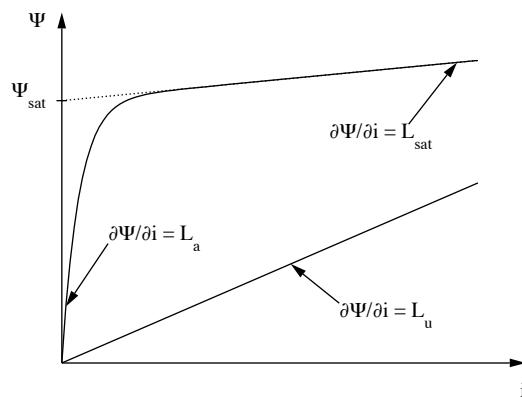


These components represent analytical models of three common switched reluctance machine types: three-phase 6/4 SRM, four-phase 8/6 SRM and five-phase 10/8 SRM.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. In the component icon, the positive terminals of the stator windings are marked with a dot.

Note The Switched Reluctance Machine models can only be simulated with the Continuous State-Space Method.

The machine flux linkage is modeled as a nonlinear function of the stator current and rotor angle $\Psi(i, \theta)$ accounting for both the magnetization characteristic of the iron and the variable air gap.



In the unaligned rotor position the flux linkage is approximated as a linear function:

$$\Psi_u(i) = L_u \cdot i$$

In the aligned rotor position the flux linkage is a nonlinear function of the stator current:

$$\Psi_a(i) = \Psi_{\text{sat}} \cdot (1 - e^{-K \cdot i}) + L_{\text{sat}} \cdot i$$

where

$$K = \frac{L_a - L_{\text{sat}}}{\Psi_{\text{sat}}}$$

For intermediate rotor positions the flux linkage is written as a weighted sum of these two extremes

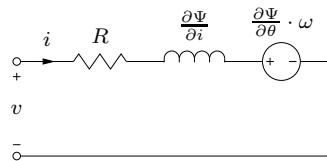
$$\Psi(i, \theta) = \Psi_u(i) + f(\theta) \cdot (\Psi_a(i) - \Psi_u(i))$$

using the weighting function

$$f(\theta) = \frac{1}{2} + \frac{1}{2} \cos \left(N_r \left[\theta + 2\pi \cdot \frac{x}{N_s} \right] \right)$$

where N_r is the number of rotor poles, N_s is the number of stator poles, and $x = 0 \dots (N_s/2 - 1)$ is the index of the stator phase.

Electrical System



The terminal voltage of a stator phase is determined by the equation

$$v = R \cdot i + \frac{d\Psi}{dt} = R \cdot i + \frac{\partial\Psi}{\partial i} \cdot \frac{di}{dt} + \frac{\partial\Psi}{\partial\theta} \cdot \frac{d\theta}{dt}$$

The electromagnetic torque produced by one phase is the derivative of the co-energy with respect to the rotor angle:

$$T(i, \theta) = \frac{\partial}{\partial\theta} \int_0^i \Psi(i', \theta) di'$$

The total torque T_e of the machine is given by the sum of the individual phase torques.

Mechanical System

Rotor speed:

$$\frac{d}{dt}\omega = \frac{1}{J} (T_e - F\omega - T_m)$$

Rotor angle:

$$\frac{d}{dt}\theta = \omega$$

Parameters

Stator resistance

Stator resistance R in ohms (Ω).

Unaligned stator inductance

Stator inductance L_u in the unaligned rotor position, in henries (H).

Initial aligned stator inductance

Initial stator inductance L_a in the aligned rotor position, in henries (H).

Saturated aligned stator inductance

Saturated stator inductance L_{sat} in the aligned rotor position, in henries (H).

Aligned saturation flux linkage

Flux linkage Ψ_{sat} at which the stator saturates in the aligned position, in Vs.

Inertia

Combined rotor and load inertia J in Nms².

Friction coefficient

Viscous friction F in Nms.

Initial rotor speed

Initial mechanical speed $\omega_{m,0}$ in radians per second (s^{-1}).

Initial rotor angle

Initial mechanical rotor angle $\theta_{m,0}$ in radians.

Initial stator currents

A three-element vector containing the initial stator currents $i_{a,0}$, $i_{b,0}$ and $i_{c,0}$ of phases a, b and c in amperes (A).

Inputs and Outputs

Mechanical torque

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector “m” contains the following 7 signals:

(1) Rotor speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

(2) Rotor position

The mechanical rotor angle θ_m in radians.

(3) Electrical torque

The electrical torque T_e of the machine in Nm.

(4-6) Flux linkages

The flux linkages in the individual phases of the machine in Vs.

References

- D.A. Torrey, J.A. Lang, "Modelling a nonlinear variable-reluctance motor drive", IEE Proceedings, Vol. 137, Pt. B, No. 5, Sept. 1990.
- D.A. Torrey, X.-M. Niu, E.J. Unkauf, "Analytical modelling of variable-reluctance machine magnetisation characteristic", IEE Proceedings Electric Power Applications, Vol. 142, No. 1, Jan. 1995.

Symmetrical PWM

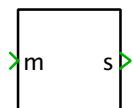
Purpose

Generate PWM signal using symmetrical triangular carrier

Library

Control / Modulators

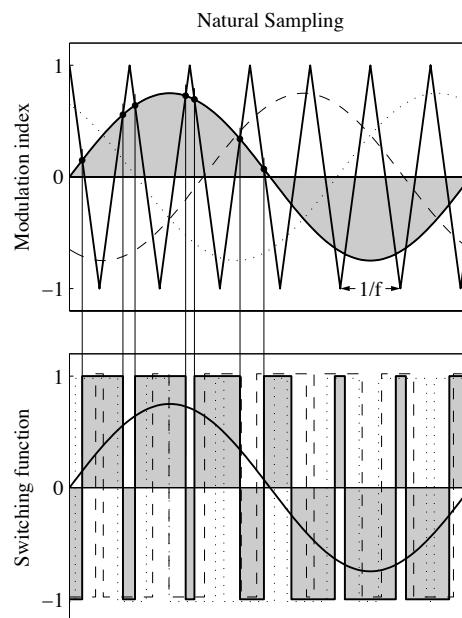
Description



2-level PWM generator with a symmetrical triangular carrier. The input m is the modulation index, and the output s is the switching function. If the modulation index is a vector the switching function is also a vector of the same width.

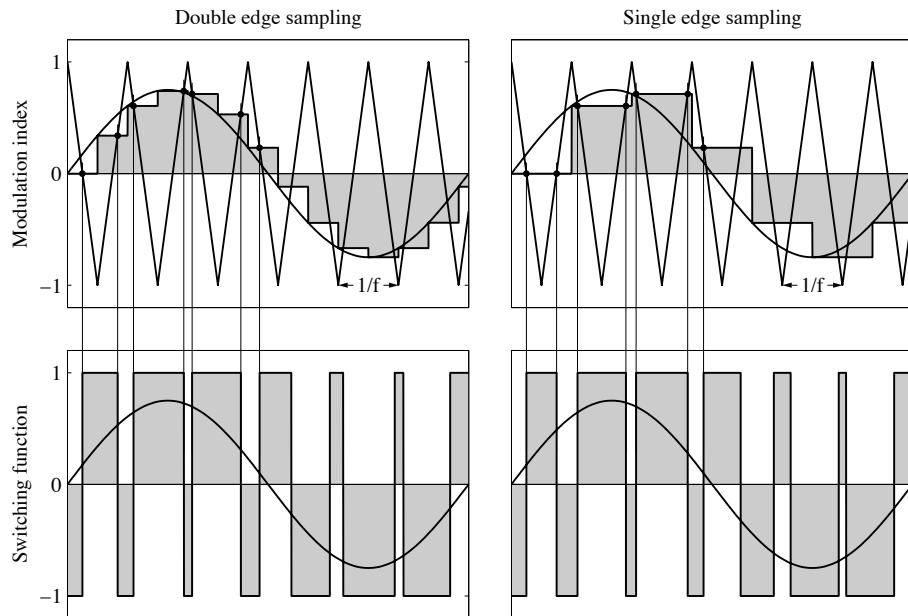
The block can be used to control the IGBT Converter (see page 273) or the ideal Converter (see page 267). In these cases the modulation index must have a width of 3 according to the number of inverter legs.

The block offers different sampling methods for the modulation index. The figure below illustrates Natural Sampling.



The following figures illustrate the different Regular Sampling methods. In the figure on the left, double edge sampling is used, i.e. the modulation index is updated at both tips of the triangular carrier. In the right figure, single

edge sampling is employed. Here, the modulation index is updated only at the upper tips of the carrier.



Parameters

Sampling

Select a sampling method. If you select Natural Sampling the carrier signal may begin with 0 or 1 at simulation start. The Regular Sampling method lets you choose between double edge or single edge sampling.

Carrier frequency

The frequency f of the triangular carrier signal, in Hz.

Carrier offset

The time offset of the carrier signal, in p.u. of the carrier period.

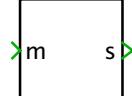
Input limits

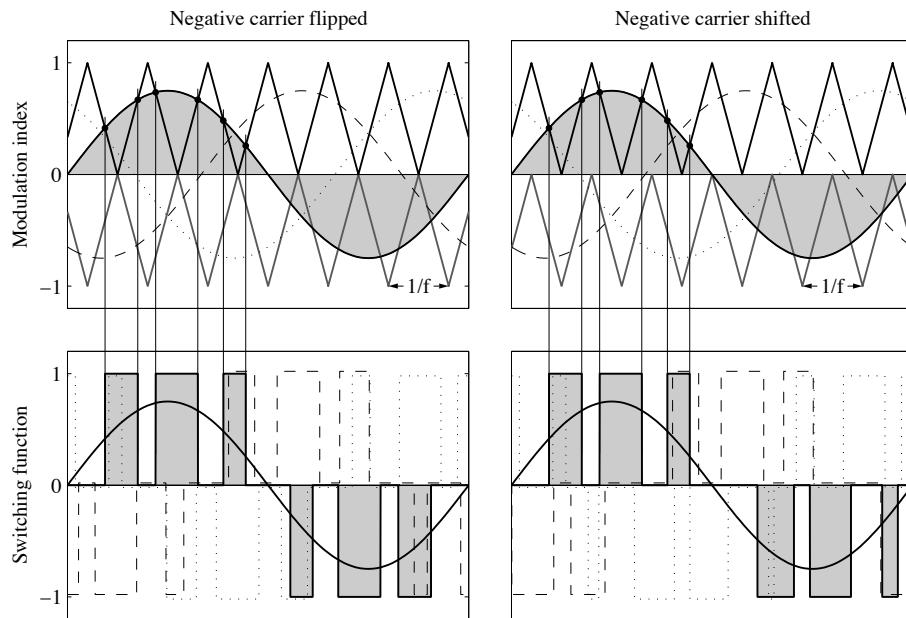
The range of the modulation index. The default is [-1 1].

Output values

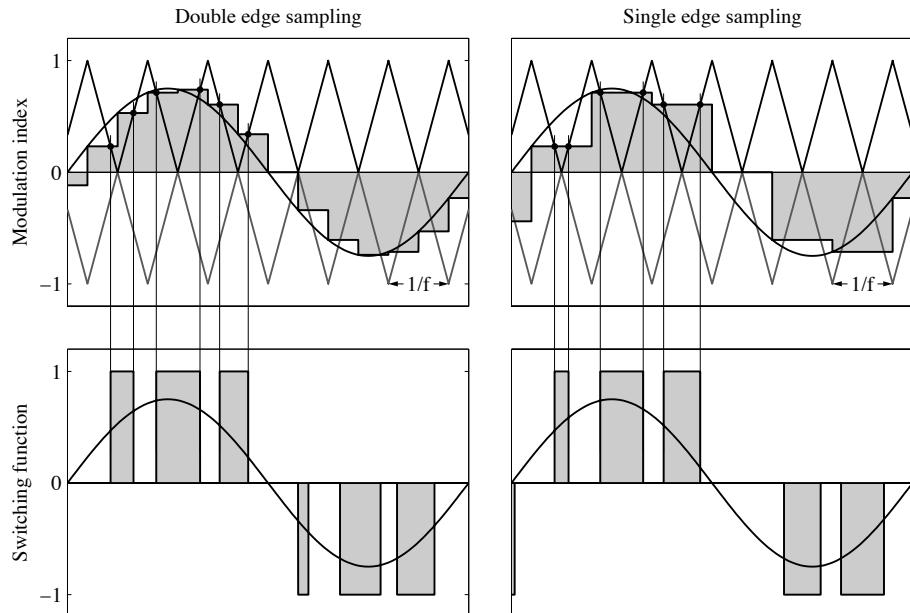
Values of the switching function in off-state and on-state. The default is [-1 1].

Symmetrical PWM (3-Level)

Purpose	Generate 3-level PWM signal using symmetrical triangular carriers
Library	Control / Modulators
Description	3-level PWM generator with two symmetrical triangular carriers. The input m is the modulation index. The switching function s outputs either 1, 0 or -1. If the modulation index is a vector the switching function is also a vector of the same width.
	The block can be used to control the 3-Level IGBT Converter (see page 271) or the ideal 3-Level Converter (see page 266). In these cases the modulation index must have a width of 3 according to the number of inverter legs. The figures below illustrate the Natural Sampling method. In the left figure, the negative carrier signal is obtained by flipping the positive carrier vertically around the time axis. In the right figure, the positive carrier is vertically shifted to construct the negative carrier. The latter technique reduces the switching frequency and hence the semiconductor stress in three-phase converters.



The figures below illustrate the different Regular Sampling methods offered by this block. With double edge sampling (left figure) the modulation index is updated at the carrier tips and zero-crossings. With single edge sampling (right figure) the modulation index is updated only at the outer tips.



Parameters

Sampling

Select a sampling method. If you select Natural Sampling the carrier signal may begin with 0 or 1 at simulation start. The Regular Sampling method lets you choose between double edge and single edge sampling.

Carrier frequency

The frequency f of the triangular carrier signals, in Hz.

Carrier offset

The time offset of the carrier signal, in p.u. of the carrier period.

Negative carrier

Select the phase shift between the negative and positive carrier signals. The negative carrier may be constructed from the positive carrier either by flipping or shifting.

Input limits

The range of the modulation index. The default is [-1 1].

Synchronous Machine (Round Rotor)

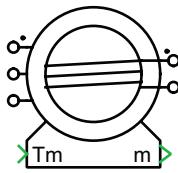
Purpose

Smooth air-gap synchronous machine with main-flux saturation

Library

Electrical / Machines

Description

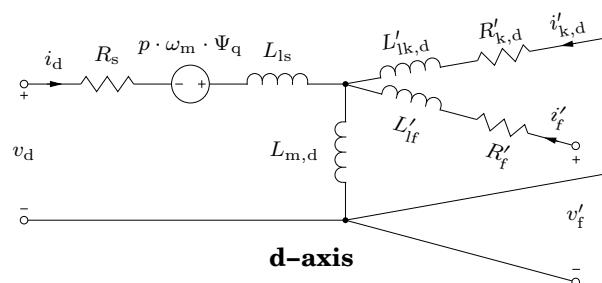


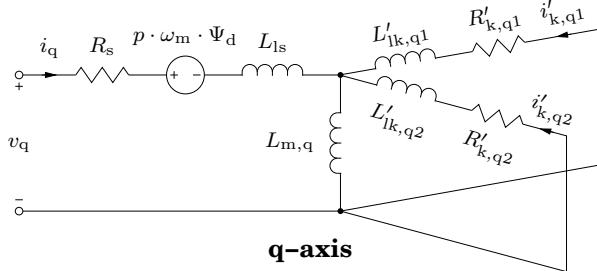
This synchronous machine has one damper winding on the direct axis and two damper windings on the quadrature axis of the rotor. Main flux saturation is modeled by means of a continuous function.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator winding and the positive pole of the field winding are marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit** menu. If you want to make changes, you must first choose **Break library link** and then **Unprotect**, both from the **Edit** menu.

Electrical System





Stator flux linkages:

$$\Psi_d = L_{ls} i_d + L_{m,d} (i_d + i'_f + i'_{k,d})$$

$$\Psi_q = L_{ls} i_q + L_{m,q} (i_q + i'_g + i'_{k,q})$$

The machine model offers two different implementations of the electrical system: a traditional rotor reference frame and a voltage-behind-reactance formulation.

Rotor Reference Frame Using Park's transformation, the 3-phase circuit equations in physical variables are transformed to the dq rotor reference frame. This results in constant coefficients in the stator and rotor equations making the model numerically efficient. However, interfacing the dq model with the external 3-phase network may be difficult. Since the coordinate transformations are based on voltage-controlled current sources, inductors and naturally commutated devices such as diode rectifiers may not be directly connected to the stator terminals. In these cases, fictitious RC snubbers are required to create the necessary voltages across the terminals.

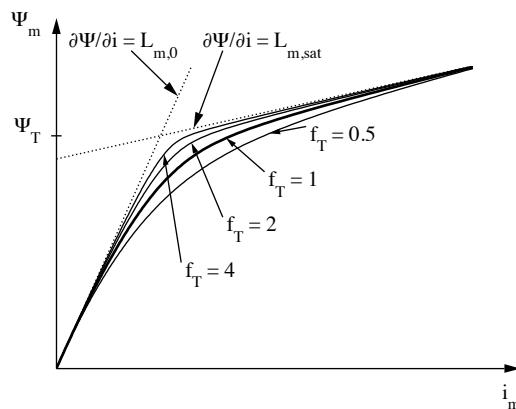
Voltage behind Reactance This formulation allows for direct interfacing of arbitrary external networks with the 3-phase stator terminals. The rotor dynamics are expressed using explicit state-variable equations while the stator branch equations are described in circuit form. However, due to the resulting time-varying inductance matrices, this implementation is numerically less efficient than the traditional rotor reference frame.

In both implementations, the value of the main flux inductance L_m is not constant but depends on the main flux linkage Ψ_m as illustrated in the Ψ_m/i_m diagram. For flux linkages Ψ_m far below the transition flux Ψ_T , the relationship between flux and current is almost linear and determined by the unsaturated magnetizing inductance $L_{m,0}$. For large flux linkages the relationship is governed by the saturated magnetizing inductance $L_{m,sat}$. Ψ_T defines the knee of the transition between unsaturated and saturated main flux inductance.

The tightness of the transition is defined with the form factor f_T . If you do not have detailed information about the saturation characteristic of your machine, $f_T = 1$ is a good starting value. The function

```
plsaturation(Lm0,Lmsat,PsiT,fT)
```

plots the main flux vs. current curve and the magnetizing inductance vs. current curve for the parameters specified.



The model accounts for steady-state cross-saturation, i.e. the steady-state magnetizing inductances along the d-axis and q-axis are functions of the currents in both axes. For rotating reference frame formulation, the stator currents, the field current and the main flux linkage are chosen as state variables. With this choice of state variables, the representation of dynamic cross-saturation could be neglected without affecting the performance of the machine. The computation of the time derivative of the main flux inductance was not required.

Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p (i_q \Psi_d - i_d \Psi_q)$$

Mechanical System

Mechanical rotor speed ω_m :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\dot{\theta}_m = \omega_m$$

Parameters

Most parameters for the Salient Pole Synchronous Machine (see page 418) are also applicable to this round rotor machine. The following parameters are different:

Unsaturated magnetizing inductance

The unsaturated magnetizing inductance $L_{m,0}$. The value in henries (H) is referred to the stator side.

Saturated magnetizing inductance

The saturated magnetizing inductance $L_{m,sat}$, in H. If no saturation is to be modeled, set $L_{m,sat} = L_{m,0}$.

Damper resistance

A three-element vector containing the damper winding resistance $R'_{k,d}$, $R'_{k,q1}$ and $R'_{k,q2}$ of the d-axis and the q-axis. The values in ohms (Ω) are referred to the stator side.

Damper leakage inductance

A three-element vector containing the damper winding leakage inductance $L'_{lk,d}$, $L'_{lk,q1}$ and $L'_{lk,q2}$ of the d-axis and the q-axis. The values in henries (H) are referred to the stator side.

Initial field/damper current

A two-element vector containing the initial currents $i'_{f,0}$ in the field winding and $i'_{k,q1,0}$ in one of the damper windings in amperes (A), referred to the stator side.

Inputs and Outputs

Same as for the Salient Pole Synchronous Machine (see page 418).

Probe Signals

Most probe signals for the Salient Pole Synchronous Machine (see page 418) are also available with this machine. Only the following probe signal is different:

Damper currents

The damper currents $i'_{k,d}$, $i'_{k,q1}$ and $i'_{k,q2}$ in the stationary reference frame in A, referred to the stator side.

References

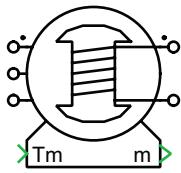
- D. C. Aliprantis, O. Wasyczuk, C. D. Rodriguez Valdez, "A voltage-behind-reactance synchronous machine model with saturation and arbitrary rotor network representation", IEEE Transactions on Energy Conversion, Vol. 23, No. 2, June 2008.
- K. A. Corzine, B. T. Kuhn, S. D. Sudhoff, H. J. Hegner, "An improved method for incorporating magnetic saturation in the Q-D synchronous machine model", IEEE Transactions on Energy Conversion, Vol. 13, No. 3, Sept. 1998.
- E. Levi, "Modelling of magnetic saturation in smooth air-gap synchronous machines", IEEE Transactions on Energy Conversion, Vol. 12, No. 2, March 1997.
- E. Levi, "Impact of cross-saturation on accuracy of saturated synchronous machine models", IEEE Transactions on Energy Conversion, Vol. 15, No. 2, June 2000.

Synchronous Machine (Salient Pole)

Purpose Salient pole synchronous machine with main-flux saturation

Library Electrical / Machines

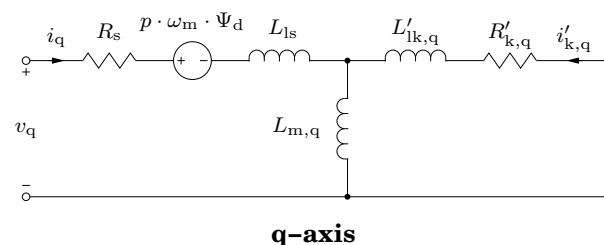
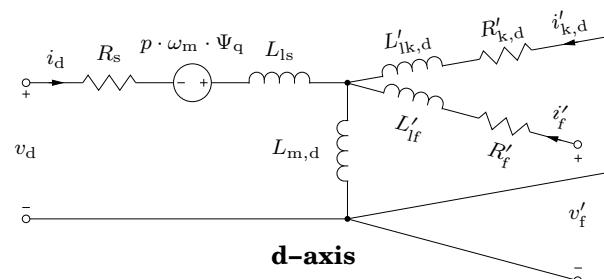
Description



This synchronous machine has one damper winding each on the direct and the quadrature axis of the rotor. Main flux saturation is modeled by means of a continuous function.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator winding and the positive pole of the field winding are marked with a dot.

Electrical System



Stator flux linkages:

$$\Psi_d = L_{ls} i_d + L_{m,d} (i_d + i'_f + i'_{k,d})$$

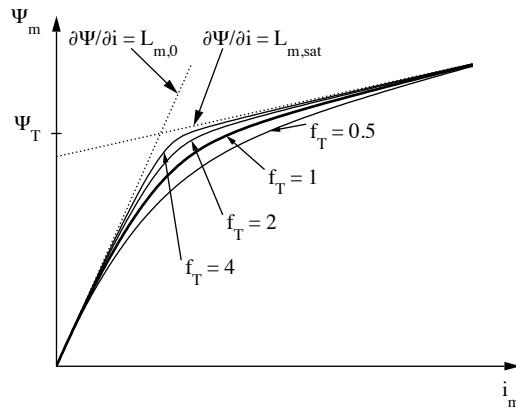
$$\Psi_q = L_{ls} i_q + L_{m,q} (i_q + i'_{k,q})$$

The machine model offers two different implementations of the electrical system: a traditional rotor reference frame and a voltage-behind-reactance formulation.

Rotor Reference Frame Using Park's transformation, the 3-phase circuit equations in physical variables are transformed to the dq rotor reference frame. This results in constant coefficients in the stator and rotor equations making the model numerically efficient. However, interfacing the dq model with the external 3-phase network may be difficult. Since the coordinate transformations are based on voltage-controlled current sources, inductors and naturally commutated devices such as diode rectifiers may not be directly connected to the stator terminals. In these cases, fictitious RC snubbers are required to create the necessary voltages across the terminals.

Voltage behind Reactance This formulation allows for direct interfacing of arbitrary external networks with the 3-phase stator terminals. The rotor dynamics are expressed using explicit state-variable equations while the stator branch equations are described in circuit form. However, due to the resulting time-varying inductance matrices, this implementation is numerically less efficient than the traditional rotor reference frame.

In both implementations, the value of the main flux inductances $L_{m,d}$ and $L_{m,q}$ are not constant but depend on the main flux linkage Ψ_m as illustrated in the Ψ_m/i_m diagram. In this machine model, the anisotropic factor



$$m = \sqrt{L_{m,q,0}/L_{m,d,0}} \equiv \sqrt{L_{m,q}/L_{m,d}} = \text{const.}$$

is assumed to be constant at all saturation levels. The equivalent magnetizing flux Ψ_m in an isotropic machine is defined as

$$\Psi_m = \sqrt{\Psi_{m,d}^2 + \Psi_{m,q}^2/m^2}.$$

For flux linkages Ψ_m far below the transition flux Ψ_T , the relationship between flux and current is almost linear and determined by the unsaturated magnetizing inductance $L_{m,0}$. For large flux linkages the relationship is governed by the saturated magnetizing inductance $L_{m,sat}$. Ψ_T defines the knee of the transition between unsaturated and saturated main flux inductance. The tightness of the transition is defined with the form factor f_T . If you do not have detailed information about the saturation characteristic of your machine, $f_T = 1$ is a good starting value. The function

`plsatisfaction(Lm0,Lmsat,PsiT,fT)`

plots the main flux vs. current curve and the magnetizing inductance vs. current curve for the parameters specified.

The model accounts for steady-state cross-saturation, i.e. the steady-state magnetizing inductances along the d-axis and q-axis are functions of the currents in both axes. For rotating reference frame formulation, the stator currents, the field current and the main flux linkage are chosen as state variables. With this choice of state variables, the representation of dynamic cross-saturation could be neglected without affecting the performance of the machine. The computation of the time derivative of the main flux inductance was not required.

Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p (i_q \Psi_d - i_d \Psi_q)$$

Mechanical System

Mechanical rotor speed ω_m :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\dot{\theta}_m = \omega_m$$

Parameters	Model
	Implementation in the rotor reference frame or as a voltage behind reactance.
Stator resistance	Armature or stator winding resistance R_s in ohms (Ω).
Stator leakage inductance	Armature or stator leakage inductance L_{ls} in henries (H).
Unsaturated magnetizing inductance	A two-element vector containing the unsaturated stator magnetizing inductance $L_{m,d,0}$ and $L_{m,q,0}$ of the d-axis and the q-axis. The values in henries (H) are referred to the stator side.
Saturated magnetizing inductance	The saturated stator magnetizing inductance $L_{m,d,sat}$ along the d-axis, in H. If no saturation is to be modeled, set $L_{m,d,sat} = L_{m,d,0}$.
Magnetizing flux at saturation transition	Transition flux linkage Ψ_T , in Vs, defining the knee between unsaturated and saturated main flux inductance.
Tightness of saturation transition	Form factor f_T defining the tightness of the transition between unsaturated and saturated main flux inductance. The default is 1.
Field resistance	d-axis field winding resistance R'_f in ohms (Ω), referred to the stator side.
Field leakage inductance	d-axis field winding leakage inductance L'_{lf} in henries (H), referred to the stator side.
Damper resistance	A two-element vector containing the damper winding resistance $R'_{k,d}$ and $R'_{k,q}$ of the d-axis and the q-axis. The values in ohms (Ω) are referred to the stator side.
Damper leakage inductance	A two-element vector containing the damper winding leakage inductance $L'_{lk,d}$ and $L'_{lk,q}$ of the d-axis and the q-axis. The values in henries (H) are referred to the stator side.
Inertia	Combined rotor and load inertia J in Nms ² .
Friction coefficient	Viscous friction F in Nms.

Number of pole pairs

Number of pole pairs p .

Initial rotor speed

Initial mechanical speed $\omega_{m,0}$ in radians per second (s^{-1}).

Initial rotor position

Initial mechanical rotor angle $\theta_{m,0}$ in radians. If $\theta_{m,0}$ is an integer multiple of $2\pi/p$ the d-axis is aligned with phase a of the stator windings at simulation start.

Initial stator currents

A two-element vector containing the initial stator currents $i_{a,0}$ and $i_{b,0}$ of phase a and b in amperes (A).

Initial field current

Initial current $i'_{f,0}$ in the field winding in amperes (A), referred to the stator side.

Initial stator flux

A two-element vector containing the initial stator flux $\Psi'_{d,0}$ and $\Psi'_{q,0}$ in the rotor reference frame in Vs.

Inputs and Outputs

Mechanical torque

The input signal T_m represents the mechanical torque at the rotor shaft, in Nm.

The output vector “m” contains the following 3 signals:

(1) Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

(2) Rotor position

The mechanical rotor angle θ_m in radians.

(3) Electrical torque

The electrical torque T_e of the machine in Nm.

Probe Signals

Stator phase currents

The three-phase stator winding currents i_a , i_b and i_c , in A. Currents flowing into the machine are considered positive.

Field currents

The excitation current i'_f in A, referred to the stator side.

Damper currents

The damper currents $i'_{k,d}$ and $i'_{k,q}$ in the stationary reference frame, in A.

Stator flux (dq)

The stator flux linkages Ψ_d and Ψ_q in the stationary reference frame in Vs.

Magnetizing flux (dq)

The magnetizing flux linkages $\Psi_{m,d}$ and $\Psi_{m,q}$ in the stationary reference frame in Vs.

Rotational speed

The rotational speed ω_m of the rotor in radians per second (s^{-1}).

Rotor position

The mechanical rotor angle θ_m in radians.

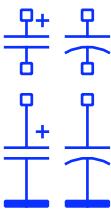
Electrical torque

The electrical torque T_e of the machine in Nm.

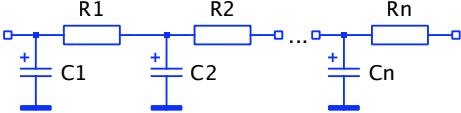
References

- D. C. Aliprantis, O. Wasyczuk, C. D. Rodriguez Valdez, "A voltage-behind-reactance synchronous machine model with saturation and arbitrary rotor network representation", IEEE Transactions on Energy Conversion, Vol. 23, No. 2, June 2008.
- K. A. Corzine, B. T. Kuhn, S. D. Sudhoff, H. J. Hegner, "An improved method for incorporating magnetic saturation in the Q-D synchronous machine model", IEEE Transactions on Energy Conversion, Vol. 13, No. 3, Sept. 1998.
- E. Levi, "Saturation modelling in D-Q axis models of salient pole synchronous machines", IEEE Transactions on Energy Conversion, Vol. 14, No. 1, March 1999.
- E. Levi, "Impact of cross-saturation on accuracy of saturated synchronous machine models", IEEE Transactions on Energy Conversion, Vol. 15, No. 2, June 2000.

Thermal Capacitor

Purpose	Thermal capacitance of piece of material
Library	Thermal
Description	This component provides an ideal thermal capacitance between its two thermal ports or between the thermal port and the thermal reference. See section “Configuring PLECS” (on page 35) for information on how to change the graphical representation of thermal capacitors.
	
Parameters	Capacitance The value of the capacitor, in J/K. All finite positive and negative values are accepted, including 0. The default is 1. Initial temperature The initial temperature difference between the thermal ports or between the thermal port and thermal reference at simulation start, in kelvin (K). The default is 0.
Probe Signals	Temperature The temperature difference measured across the capacitance. A positive value is measured when the temperature at the terminal marked with “+” is greater than the temperature at the unmarked terminal.

Thermal Chain

Purpose	Thermal impedance implemented as RC chain
Library	Thermal
Description	<p>This component implements a thermal RC chain of variable length. Using the elements of the vectors provided in the component parameters Thermal resistances and Thermal capacitances a subsystem is built as shown below. The thermal capacitor C1 is connected to the terminal marked with a dot.</p> 
Parameters	<p>Thermal resistances A vector containing the values of the thermal resistors R1 ... Rn, in K/W.</p> <p>Thermal capacitances A vector containing the values of the thermal capacitors C1 ... Cn, in J/K.</p> <p>Initial temperature A scalar value specifying the initial temperature of all thermal capacitors at simulation start, in kelvin (K). The default is 0.</p>

Thermal Ground

Purpose	Connect to common reference temperature
Library	Thermal
Description	The Thermal Ground implements a connection to the thermal reference.



Thermal Port

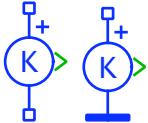
Purpose	Add thermal connector to subsystem
Library	Thermal
Description	<p>Thermal ports are used to establish thermal connections between a PLECS circuit and a subsystem (see page 402). If you copy a Thermal Port block into the schematic of a subsystem a terminal will be created on the subsystem block. The name of the port block will appear as the terminal label. If you choose to hide the block name by unselecting the show button in the dialog box the terminal label will also disappear.</p> <p>Terminals can be moved around the edges of the subsystem by holding down the Shift key or by using the middle mouse button.</p>
	

Note Thermal Port blocks cannot be placed in top-level circuits nor may they be used in schematics that contain Ambient Temperature blocks (see page 197).

Thermal Resistor

Purpose	Thermal resistance of piece of material
Library	Thermal
Description	This component provides an ideal one-dimensional thermal resistor between its two thermal ports. See section “Configuring PLECS” (on page 35) for information on how to change the graphical representation of thermal resistors.
	
	
Parameter	Thermal resistance The resistance in K/W. All positive and negative values are accepted, including 0 and inf (∞). The default is 1.

Thermometer

Purpose	Output measured temperature as signal
Library	Thermal
Description	The Thermometer measures the temperature difference between its two thermal ports or between the thermal port and thermal reference and provides it as a signal at the output of the component. The output signal can be made accessible in Simulink with a Output block (see page 387) or by dragging the component into the dialog box of a Probe block.
	

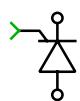
Probe Signals	Measured temperature
	The measured temperature in kelvin (K).

Thyristor

Purpose Ideal thyristor (SCR) with optional forward voltage and on-resistance

Library Electrical / Power Semiconductors

Description The Thyristor can conduct current only in one direction—like the diode. In addition to the diode it can be controlled by an external gate signal. The thyristor is modeled by an ideal switch that closes if the voltage between anode and cathode is positive and a non-zero gate signal is applied. The switch remains closed until the current passes through zero. A thyristor cannot be switched off via the gate.



Parameters The following parameters may either be scalars or vectors corresponding to the implicit width of the component:

Forward voltage

Additional dc voltage V_f in volts (V) between anode and cathode when the thyristor is conducting. The default is 0.

On-resistance

The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.

Initial conductivity

Initial conduction state of the thyristor. The thyristor is initially blocking if the parameter evaluates to zero, otherwise it is conducting.

Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79). If no thermal description is given the losses are calculated based on the voltage drop $v_{on} = V_f + R_{on} \cdot i$.

Initial temperature

Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.

Probe Signals

Thyristor voltage

The voltage measured between anode and cathode.

Thyristor current

The current through the thyristor flowing from anode to cathode.

Thyristor gate signal

The gate input signal of the thyristor.

Thyristor conductivity

Conduction state of the internal switch. The signal outputs 0 when the thyristor is blocking, and 1 when it is conducting.

Thyristor junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

Thyristor conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

Thyristor switching loss

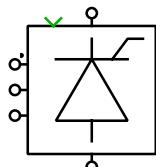
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

Thyristor Rectifier/Inverter

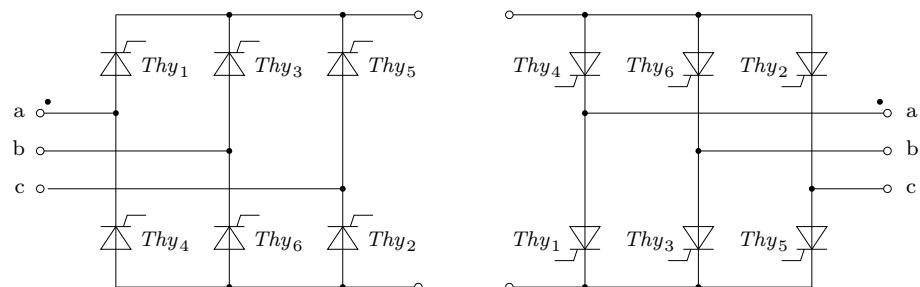
Purpose 3-phase thyristor rectifier/inverter

Library Electrical / Converters

Description



Implements a three-phase rectifier or inverter based on the Thyristor model (see page 430). The gate input is a vector of six signals ordered according to the natural sequence of commutation. This sequence corresponds to the numbering of the thyristors in the electrical circuits below. The rectifier is shown on the left side, the inverter on the right:



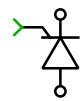
Parameters For a description of the parameters see the documentation of the Thyristor (on page 430).

Probe Signals

The thyristor converters provide six probe signals, each a vector containing the appropriate quantities of the six individual thyristors: voltage, current, conduction loss and switching loss. The vector elements are ordered according to the natural sequence of commutation.

Thyristor with Reverse Recovery

Purpose	Dynamic thyristor (SCR) model with reverse recovery
Library	Electrical / Power Semiconductors
Description	<p>This component is a behavioral model of a thyristor which reproduces the effect of reverse recovery. The effect can be observed when a forward biased thyristor is rapidly turned off. It takes some time until the excess charge stored in the thyristor during conduction is removed. During this time the thyristor represents a short circuit instead of an open circuit, and a negative current can flow through the thyristor. The thyristor finally turns off when the charge is swept out by the reverse current and lost by internal recombination. The same effect is modeled in the Diode with Reverse Recovery (see page 234) and described there in detail.</p>



Note

- Due to the small time-constant introduced by the turn-off transient a stiff solver is recommended for this device model.
- If multiple thyristors are connected in series, the off-resistance may not be infinite.

Parameters

Forward voltage

Additional dc voltage V_f in volts (V) between anode and cathode when the thyristor is conducting. The default is 0.

On-resistance

The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0.

Off-resistance

The resistance R_{off} of the blocking device, in ohms (Ω). The default is inf. If thyristors are connected in series, the off-resistance must have a large finite value.

Continuous forward current

The continuous forward current I_{f0} under test conditions.

Current slope at turn-off

The turn-off current slope dI_r/dt under test conditions.

Reverse recovery time

The turn-off time t_{rr} under test conditions.

Peak recovery current

The absolute peak value of the reverse current I_{rrm} under test conditions.

Reverse recovery charge

The reverse recovery charge Q_{rr} under test conditions. If both t_{rr} and I_{rrm} are specified, this parameter is ignored.

Lrr

This inductance acts as a probe measuring the di/dt . It should be set to a very small value. The default is 10e-10.

Probe Signals**Thyristor voltage**

The voltage measured between anode and cathode.

Thyristor current

The current through the thyristor flowing from anode to cathode.

Thyristor conductivity

Conduction state of the internal switch. The signal outputs 0 when the thyristor is blocking, and 1 when it is conducting.

References

- A. Courtay, "MAST power diode and thyristor models including automatic parameter extraction", SABER User Group Meeting Brighton, UK, Sept. 1995.

To File

Purpose Write time stamps and signal values to a file.

Library System

Description While a simulation is running, the To File block writes the time stamps and the values of its input signals to a file. The file format can be either a text file with comma separated values (csv) or a MATLAB data file (mat). CSV files can be imported by all common spreadsheet tools like Microsoft Excel.



In a csv file a new row is appended for each time step. When writing to a MATLAB file the resulting data contains a column for each time step.

The first value for each data record is the simulation time of the current simulation step. The value is followed by the signal values of the input signal.

Parameters

Filename

The name of the data file to write to. Files will be stored relative to the model directory unless an absolute file path is given. The data file will be created if it doesn't exist. An existing file of the same name will be overwritten.

File type

The file format to use for the data file. The file can be written as a text file with comma separated values (csv) or as a MATLAB data file (mat).

Sample time

For positive values the input data will be written to the data file once in the given simulation interval. If the value is set to 0 the input data is written to the data file in each simulation step. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 32).

Transfer Function

Purpose Model linear time-invariant system as transfer function

Library Control / Continuous

Description The Transfer Function models a linear time-invariant system that is expressed in the Laplace domain in terms of the argument s :

$$\frac{1}{s+1}$$

$$\frac{Y(s)}{U(s)} = \frac{n_n s^n + \dots + n_1 s + n_0}{d_n s^n + \dots + d_1 s + d_0}$$

Parameters

Numerator coefficients

A vector of the s term coefficients $[n_n \dots n_1, n_0]$ for the numerator, written in descending order of powers of s . For example, the numerator $s^3 + 2s$ would be entered as $[1, 0, 2, 0]$.

The output of the Transfer Function is vectorizable by entering a matrix for the numerator.

Denominator coefficients

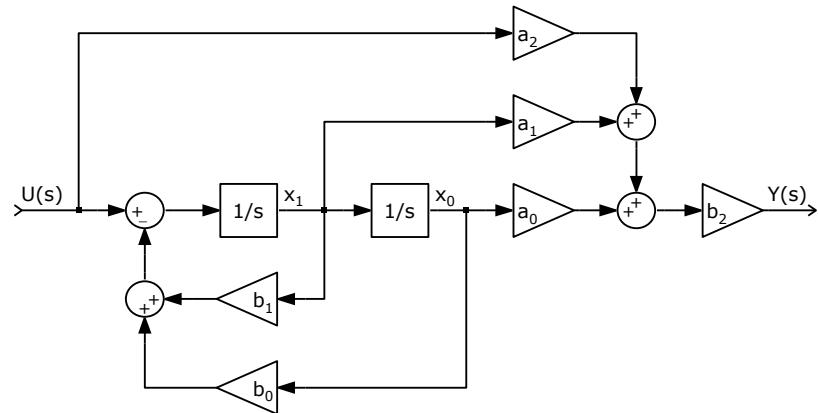
A vector of the s term coefficients $[d_n \dots d_1, d_0]$ for the denominator, written in descending order of powers of s .

Note The order of the denominator (highest power of s) must be greater than or equal to the order of the numerator.

Initial condition

The initial condition vector of the internal states of the Transfer Function in the form $[x_n \dots x_1, x_0]$. The initial conditions must be specified for the controller normal form, depicted below for the transfer function

$$\frac{Y(s)}{U(s)} = \frac{n_2 s^2 + n_1 s + n_0}{d_2 s^2 + d_1 s + d_0}$$



where

$$b_i = \frac{d_i}{d_n} \quad \text{for } i < n$$

$$b_n = \frac{1}{d_n}$$

$$a_i = n_i - \frac{n_n d_i}{d_n} \quad \text{for } i < n$$

$$a_n = n_n$$

For the normalized transfer function (with $n_n = 0$ and $d_n = 1$) this simplifies to $b_i = d_i$ and $a_i = n_i$.

Probe Signals

Input

The input signal.

Output

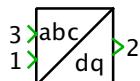
The output signal.

Transformation 3ph->RRF

Purpose Transform 3-phase signal to rotating reference frame

Library Control / Transformations

Description This block transforms a three-phase signal $[x_a \ x_b \ x_c]$ into a two-dimensional vector $[y_d \ y_q]$ in a rotating reference frame. The first input is the three-phase signal. The second input is the rotation angle φ of the rotating reference frame. φ is given in radians.

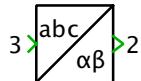


$$\begin{bmatrix} y_d \\ y_q \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \cos(\varphi - 120^\circ) & -\sin(\varphi - 120^\circ) \\ \cos(\varphi + 120^\circ) & -\sin(\varphi + 120^\circ) \end{bmatrix}^T \cdot \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

Any zero-sequence component in the three-phase signals is discarded.

Transformation 3ph->SRF

Purpose	Transform 3-phase signal to stationary reference frame
Library	Control / Transformations
Description	This block transforms a three-phase signal $[x_a \ x_b \ x_c]$ into a two-dimensional vector $[y_\alpha \ y_\beta]$ in the stationary reference frame:

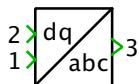


$$\begin{bmatrix} y_\alpha \\ y_\beta \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

Any zero-sequence component in the three-phase signals is discarded.

Transformation RRF->3ph

Purpose	Transform vector in rotating reference frame into 3-phase signal
Library	Control / Transformations
Description	This block transforms a two-dimensional vector $[x_d \ x_q]$ in a rotating reference frame into a three-phase signal $[y_a \ y_b \ y_c]$. The first input of the block is the vector $[x_d \ x_q]$. The second input is the rotation angle φ of the rotating reference frame. φ is given in radians.



$$\begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix} = \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \cos (\varphi - 120^\circ) & -\sin (\varphi - 120^\circ) \\ \cos (\varphi + 120^\circ) & -\sin (\varphi + 120^\circ) \end{bmatrix} \cdot \begin{bmatrix} x_d \\ x_q \end{bmatrix}$$

The resulting three-phase signal does not have any zero-sequence component.

Transformation RRF->SRF

Purpose

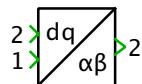
Transform vector from rotating to stationary reference frame

Library

Control / Transformations

Description

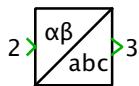
This block transforms a two-dimensional vector $[x_d \ x_q]$ from a rotating reference frame into a vector $[y_\alpha \ y_\beta]$ in the stationary reference frame. The first input of the block is the vector $[x_d \ x_q]$. The second input is the angle φ between the rotating and the stationary frame. φ is given in radians.



$$\begin{bmatrix} y_\alpha \\ y_\beta \end{bmatrix} = \begin{bmatrix} \cos \omega_1 t & -\sin \omega_1 t \\ \sin \omega_1 t & \cos \omega_1 t \end{bmatrix} \cdot \begin{bmatrix} x_d \\ x_q \end{bmatrix}$$

Transformation SRF->3ph

Purpose	Transform vector in stationary reference frame into 3-phase signal
Library	Control / Transformations
Description	This block transforms a two-dimensional vector $[x_\alpha \ x_\beta]$ in the stationary reference frame into a three-phase signal $[y_a \ y_b \ y_c]$.



$$\begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}$$

The resulting three-phase signal does not have any zero-sequence component.

Transformation SRF->RRF

Purpose

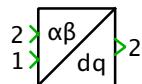
Transform vector from stationary to rotating reference frame

Library

Control / Transformations

Description

This block transforms a two-dimensional vector $[x_\alpha \ x_\beta]$ in the stationary reference frame into a vector $[y_d \ y_q]$ in a rotating reference frame. The first input is the vector $[x_\alpha \ x_\beta]$. The second input is the angle φ between the rotating and the stationary frame. φ is given in radians.



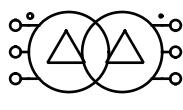
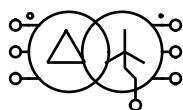
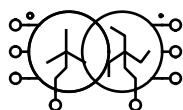
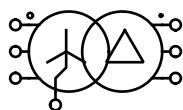
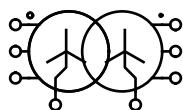
$$\begin{bmatrix} y_d \\ y_q \end{bmatrix} = \begin{bmatrix} \cos \omega_1 t & \sin \omega_1 t \\ -\sin \omega_1 t & \cos \omega_1 t \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}$$

Transformers (3ph, 2 Windings)

Purpose 3-phase transformers in Yy, Yd, Yz, Dy, Dd and Dz connection

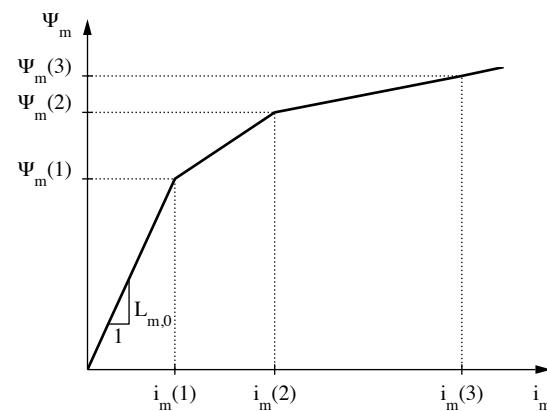
Library Electrical / Transformers

Description



This group of components implements two-winding, three-phase transformers with a three-leg or five-leg core. The transformer core is assumed symmetrical, i.e. all phases have the same parameters. Depending on the chosen component, the windings are wired in star (Y) or delta (D) connection on the primary side. On the secondary side, the windings are either in star (y), delta (d) or zig-zag (z) connection. Star and zig-zag windings have an accessible neutral point.

The phase angle difference between the primary and the secondary side can be chosen. For Yy and Dd connections, the phase lag must be an integer multiple of 60° . For Yd and Dy connections the phase lag must be an odd integer multiple of 30° . The phase lag of zig-zag windings can be chosen arbitrarily. The windings of the secondary side are allocated to the transformer legs according to the phase lag. Please note that the phase-to-phase voltage of delta windings is by a factor of $1/\sqrt{3}$ lower than the voltage of star or delta windings if the number of turns are equal.



The core saturation characteristic of the transformer legs is piece-wise linear and is modeled using the Saturable Inductor (see page 370). The magnetizing current i_m and flux Ψ_m value pairs are referred to the primary side. To model a transformer without saturation enter 1 as the magnetizing current

values and the desired magnetizing inductance L_m as the flux values. A stiff Simulink solver is recommended if the iron losses are not negligible, i.e. R_{fe} is not infinite.

Parameters

Leakage inductance

A two-element vector containing the leakage inductance of the primary side L_1 and the secondary side L_2 . The inductivity is given in henries (H).

Winding resistance

A two-element vector containing the resistance of the primary winding R_1 and the secondary winding R_2 , in ohms (Ω).

No. of turns

A two-element vector containing the number of turns of the primary winding n_1 and the secondary winding n_2 .

Magnetizing current values

A vector of positive current values in amperes (A) defining the piece-wise linear saturation characteristic of the transformer legs. The current values must be positive and strictly monotonic increasing. At least one value is required.

Magnetizing flux values

A vector of positive flux values in Vs defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Core loss resistance

An equivalent resistance R_{fe} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

No. of core legs

The number of legs of the transformer core. This value may either be 3 or 5.

Phase lag of secondary side

The phase angle between the primary side and the secondary side, in degrees. Unless the secondary side is in zig-zag connection, the angle can only be varied in steps of 60° .

Initial currents wdg. 1

A vector containing the initial currents on the primary side $i_{1,a}$, $i_{1,b}$ and, if the winding has a neutral point, $i_{1,c}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is [0 0 0].

Initial currents wdg. 2

A vector containing the initial currents on the secondary side $i_{2,a}$, $i_{2,b}$ and, if the winding has a neutral point, $i_{2,c}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is [0 0 0].

Transformers (3ph, 3 Windings)

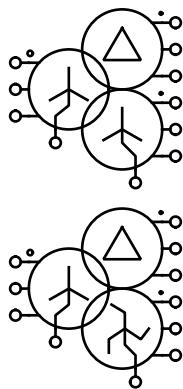
Purpose

Three-phase transformers in Ydy and Ydz connection.

Library

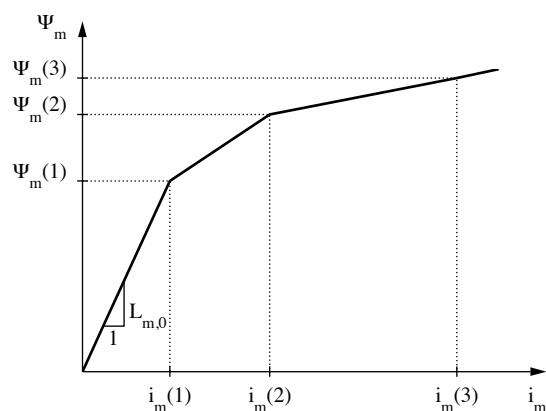
Electrical / Transformers

Description



This group of components implements three-winding, three-phase transformers with a three-leg or five-leg core. The transformer core is assumed symmetrical, i.e. all phases have the same parameters. The primary winding is in star connection with an accessible neutral point and the secondary winding is in delta connection. Depending on the chosen component, the tertiary winding is wired either in star (y) or zig-zag (z) connection.

The phase angle difference between the primary and the secondary side must be an odd integer multiple of 30° . If the tertiary winding is in star connection the phase lag against the primary side must be an integer multiple of 60° . If it is in zig-zag connection, the phase lag can be chosen arbitrarily. The windings of the secondary and tertiary side are allocated to the transformer legs according to the phase lags. Please note that the phase-to-phase voltage of delta windings is by a factor of $1/\sqrt{3}$ lower than the voltage of star or delta windings if the number of turns are equal.



The core saturation characteristic of the transformer legs is piece-wise linear and is modeled using the Saturable Inductor (see page 370). The magnetizing current i_m and flux Ψ_m value pairs are referred to the primary side. To model a transformer without saturation enter 1 as the magnetizing current values and the desired magnetizing inductance L_m as the flux values. A stiff

Simulink solver is recommended if the iron losses are not negligible, i.e. R_{fe} is not infinite.

Parameters

Leakage inductance

A three-element vector containing the leakage inductance of the primary side L_1 , the secondary side L_2 and the tertiary side L_3 . The inductivity is given in henries (H).

Winding resistance

A three-element vector containing the resistance of the primary winding R_1 , the secondary winding R_2 and the tertiary winding R_3 , in ohms (Ω).

No. of turns

A three-element vector containing the number of turns of the primary winding n_1 , the secondary winding n_2 and the tertiary winding n_3 .

Magnetizing current values

A vector of positive current values in amperes (A) defining the piece-wise linear saturation characteristic of the transformer legs. The current values must be positive and strictly monotonic increasing. At least one value is required.

Magnetizing flux values

A vector of positive flux values in Vs defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Core loss resistance

An equivalent resistance R_{fe} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

No. of core legs

The number of legs of the transformer core. This value may either be 3 or 5.

Phase lag of secondary side

The phase angle between the primary side and the secondary side, in degrees. Unless the secondary side is in zig-zag connection, the angle can only be varied in steps of 60° .

Initial currents wdg. 1

A vector containing the initial currents on the primary side $i_{1,a}$, $i_{1,b}$ and $i_{1,c}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is [0 0 0].

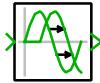
Initial currents wdg. 2

A vector containing the initial currents on the secondary side $i_{2,a}$ and $i_{2,b}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is [0 0 0].

Initial currents wdg. 3

A vector containing the initial currents on the tertiary side $i_{3,a}$, $i_{3,b}$ and $i_{3,c}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is [0 0 0].

Transport Delay

Purpose	Delay continuous input signal by fixed time
Library	Control / Delays
Description	The Transport Delay outputs a continuously changing input signal with a fixed-time delay. The output values are computed from the delayed input values with a first order (linear) interpolation. The signal can be a scalar or vector. For delaying signals that change at discrete instants please use the Pulse Delay (see page 355).
	
Parameters	
Time delay	Time by which the input signal is delayed.
Initial output	Output value after simulation start before the input values appear at the output.
Initial buffer size	Size of the internal ring buffer at simulation start. The buffer size will be increased during the simulation if required.

TRIAC

Purpose	Ideal TRIAC with optional forward voltage and on-resistance
Library	Electrical / Power Semiconductors
Description	The TRIAC can conduct current in both directions. It is built using two anti-parallel thyristors (see page 430) and controlled by an external gate signal. The TRIAC is modeled by two ideal switches that close if the voltage is positive and a non-zero gate signal is applied. The conducting switch remains closed until the current passes through zero. A TRIAC cannot be switched off via the gate.
	
Parameters	The following parameters may either be scalars or vectors corresponding to the implicit width of the component: <ul style="list-style-type: none"> Forward voltage Additional dc voltage V_f in volts (V) when one of the thyristors is conducting. The default is 0. On-resistance The resistance R_{on} of the conducting device, in ohms (Ω). The default is 0. Initial conductivity Initial conduction state of the TRIAC. The TRIAC is initially blocking if the parameter evaluates to zero, otherwise it is conducting. Thermal description Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 79). If no thermal description is given the losses are calculated based on the voltage drop $v_{on} = V_f + R_{on} \cdot i$. Initial temperature Temperature of all thermal capacitors in the equivalent Cauer network at simulation start.
Probe Signals	<ul style="list-style-type: none"> TRIAC voltage The voltage measured between the terminals. TRIAC current The current flowing through the device to the terminal with the gate. TRIAC gate signal The gate input signal of the device.

TRIAC conductivity

Conduction state of the internal switch. The signal outputs 0 when the TRIAC is blocking, and 1 when it is conducting.

TRIAC junction temperature

Temperature of the first thermal capacitor in the equivalent Cauer network.

TRIAC conduction loss

Continuous thermal conduction losses in watts (W). Only defined if the component is placed on a heat sink.

TRIAC switching loss

Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

Triangular Wave Generator

Purpose Generate periodic triangular or sawtooth waveform

Library Control / Sources

Description The Triangular Wave Generator produces a signal that periodically changes between a minimum and a maximum value and vice versa in a linear way.



Parameters

Minimum signal value

The minimum value of the signal.

Maximum signal value

The maximum value of the signal.

Frequency

The frequency of the signal in Hertz.

Duty cycle

The ratio of the rising edge to the period length. The value must be in the range [0 1]. A value of 1 produces a sawtooth waveform with a perpendicular falling edge. A value of 0 produces a reverse sawtooth waveform with a perpendicular rising edge. A value of 0.5 produces a symmetrical triangular wave.

Phase delay

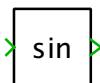
The phase delay of the triangular wave in seconds. If the phase is set to 0, the waveform begins at the rising edge.

Probe Signals

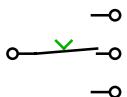
Output

The block output signal.

Trigonometric Function

Purpose	Apply specified trigonometric function
Library	Control / Math
Description	The Trigonometric Function calculates the specified function using the input signal as argument. The atan2 function calculates the principal value of the arc tangent of y/x . The quadrant of the return value is determined by the signs of x and y. The y input is marked with a small black dot.
 sin	
Parameters	Function Chooses which trigonometric function is calculated. Available functions are sin, cos, tan, asin, acos, atan and atan2. Unit Specifies the unit of the input signal (for sin, cos and tan) or output signal (for asin, acos and atan). The unit can be radians [0... 2π] or degrees [0...360].
Probe Signals	Input The block input signal. Output The block output signal.

Triple Switch

Purpose	Changeover switch with three positions
Library	Electrical / Switches
Description	This changeover switch provides an ideal short or open circuit. The switch position drawn in the icon applies if the input signal is zero. For values greater than zero the switch is the lower position. For values less than zero it is in the upper position.
	
Parameter	<p>Initial position Initial position of the switch. The switch is initially in the middle position if the parameter evaluates to zero. For values greater than zero it is in the lower position, for values less than zero it is in the upper position. This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The default value is 0.</p>
Probe Signals	<p>Switch position State of the internal switches. The signal outputs 0 if the switch is in the middle position, 1 if it is in the lower position and -1 if it is in the upper position.</p>

Turn-on Delay

Purpose	Delay rising flank of input pulses by fixed dead time
Library	Control / Delays
Description	This block is used to delay the turn-on command for power semiconductors:  <ul style="list-style-type: none">• When the input signal changes from 0 to 1 the output signal will follow after the dead time has passed, provided that the input signal has remained 1.• When the input signal becomes 0 the output is immediately set to 0.
Parameters	Dead time Time by which the turn-on event is delayed.
Probe Signals	Input The block input signal. Output The block output signal.

Variable Capacitor

Purpose Capacitance controlled by signal

Library Electrical / Passive Components

Description This component models a variable capacitor. The capacitance is determined by the signal fed into the input of the component. The current through a variable capacitance is determined by the equation



$$i = \frac{d}{dt} C \cdot v + C \cdot \frac{dv}{dt}$$

Since v is the state variable the equation above must be solved for $\frac{dv}{dt}$. The control signal must provide the values of both C and $\frac{d}{dt}C$ in the following form: $[C_1 \ C_2 \ \dots \ C_n \ \frac{d}{dt}C_1 \ \frac{d}{dt}C_2 \ \dots \ \frac{d}{dt}C_n]$. It is the responsibility of the user to provide the appropriate signals for a particular purpose (see further below).

If the component has multiple phases you can choose to include the capacitive coupling of the phases. In this case the control signal vector must contain the elements of the capacitance matrix (row by row) and their derivatives with respect to time. The control signal thus has a width of $2 \cdot n^2$, n being the number of phases.

Note The momentary capacitance may not be set to zero. In case of coupled capacitors, the capacitance matrix may not be singular.

There are two common use cases for variable capacitors, which are described in detail below: saturable capacitors, in which the capacitance is a function of the voltage and electrostatic actuators, in which the capacitance is a function of an external quantity, such as a capacitor with movable plates.

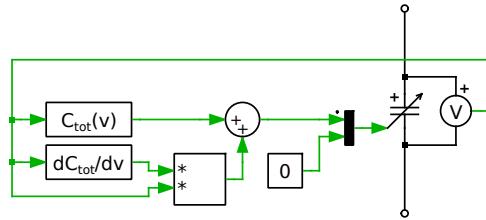
Saturable Capacitor Modeling

When specifying the characteristic of a saturable capacitor, you need to distinguish carefully between the *total* capacitance $C_{\text{tot}}(v) = Q/v$ and the *differential* capacitance $C_{\text{diff}}(v) = dQ/dv$.

With the *total* capacitance $C_{\text{tot}}(v) = Q/v$ you have

$$\begin{aligned} i &= \frac{dQ}{dt} \\ &= \frac{d}{dt}(C_{\text{tot}} \cdot v) \\ &= C_{\text{tot}} \cdot \frac{dv}{dt} + \frac{dC_{\text{tot}}}{dt} \cdot v \\ &= C_{\text{tot}} \cdot \frac{dv}{dt} + \frac{dC_{\text{tot}}}{dv} \cdot \frac{dv}{dt} \cdot v \\ &= \left(C_{\text{tot}} + \frac{dC_{\text{tot}}}{dv} \cdot v \right) \cdot \frac{dv}{dt}, \end{aligned}$$

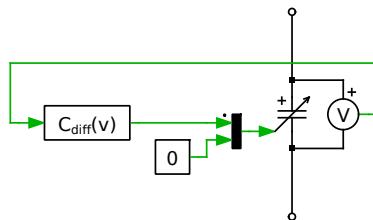
which can be implemented as follows:



With the *differential* capacitance $C_{\text{diff}}(v) = dQ/dv$ you have

$$\begin{aligned} i &= \frac{dQ}{dt} \\ &= \frac{dQ}{dv} \cdot \frac{dv}{dt} \\ &= C_{\text{diff}} \cdot \frac{dv}{dt}, \end{aligned}$$

which can be implemented as follows:



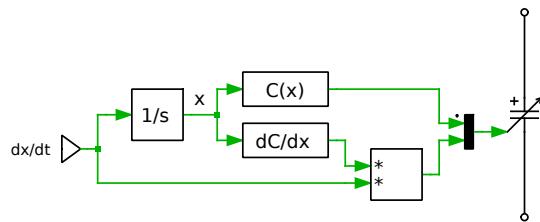
Note that in both cases the $\frac{d}{dt}C$ -input of the Variable Capacitor is zero!

Actuator Modeling

In an electrostatic actuator the capacitance is determined by an external quantity such as the distance x between the movable plates of a capacitor: $C = C(x)$. Therefore you have

$$\begin{aligned} i &= C \cdot \frac{dv}{dt} + \frac{dC}{dt} \cdot v \\ &= C \cdot \frac{dv}{dt} + \frac{dC}{dx} \cdot \frac{dx}{dt} \cdot v \quad , \end{aligned}$$

which can be implemented as follows:



Note that x is preferably calculated as the integral of dx/dt rather than calculating dx/dt as the derivative of x .

Parameters

Capacitive coupling

Specifies whether the phases should be coupled capacitively. This parameter determines how the elements of the control signal are interpreted. The default is off.

Initial voltage

The initial voltage of the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The positive pole is marked with a "+". The initial voltage default is 0.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with "+" is greater than the potential at the unmarked terminal.

Capacitor current

The current flowing through the capacitor, in amperes (A).

Variable Inductor

Purpose Inductance controlled by signal

Library Electrical / Passive Components

Description This component models a variable inductor. The inductance is determined by the signal fed into the input of the component. The voltage across a variable inductance is determined by the equation



$$v = L \cdot \frac{di}{dt} + \frac{dL}{dt} \cdot i$$

Since i is the state variable the equation above must be solved for $\frac{di}{dt}$. The control signal must provide the values of both L and $\frac{dL}{dt}$ in the following form: $[L_1 \ L_2 \dots L_n \ \frac{d}{dt}L_1 \ \frac{d}{dt}L_2 \dots \frac{d}{dt}L_n]$. It is the responsibility of the user to provide the appropriate signals for a particular purpose (see further below).

If the component has multiple phases you can choose to include the inductive coupling of the phases. In this case the control signal vector must contain the elements of the inductivity matrix (row by row) and their derivatives with respect to time. The control signal thus has a width of $2 \cdot n^2$, n being the number of phases.

Note The momentary inductance may not be set to zero. In case of coupled inductors, the inductivity matrix may not be singular.

There are two common use cases for variable inductors, which are described in detail below: saturable inductors, in which the inductance is a function of the current and actuators, in which the inductance is a function of an external quantity, such as a solenoid with a movable core.

For a more complex example of a variable inductor that depends on both the inductor current and an external quantity see the Switched Reluctance Machine (on page 405).

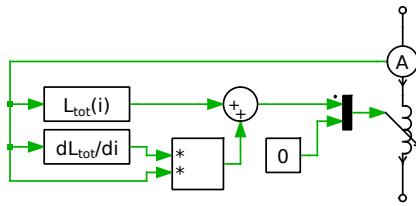
Saturable Inductor Modeling

When specifying the characteristic of a saturable inductor, you need to distinguish carefully between the *total* inductivity $L_{\text{tot}}(i) = \Psi/i$ and the *differential* inductivity $L_{\text{diff}}(i) = d\Psi/di$. See also the piece-wise linear Saturable Inductor (on page 370).

With the *total* inductivity $L_{\text{tot}}(i) = \Psi/i$ you have

$$\begin{aligned} v &= \frac{d\Psi}{dt} \\ &= \frac{d}{dt}(L_{\text{tot}} \cdot i) \\ &= L_{\text{tot}} \cdot \frac{di}{dt} + \frac{dL_{\text{tot}}}{dt} \cdot i \\ &= L_{\text{tot}} \cdot \frac{di}{dt} + \frac{dL_{\text{tot}}}{di} \cdot \frac{di}{dt} \cdot i \\ &= \left(L_{\text{tot}} + \frac{dL_{\text{tot}}}{di} \cdot i \right) \cdot \frac{di}{dt}, \end{aligned}$$

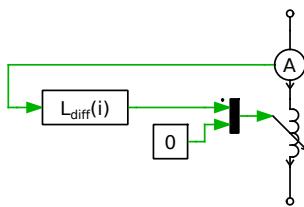
which can be implemented as follows:



With the *differential* inductivity $L_{\text{diff}}(i) = d\Psi/di$ you have

$$\begin{aligned} v &= \frac{d\Psi}{dt} \\ &= \frac{d\Psi}{di} \cdot \frac{di}{dt} \\ &= L_{\text{diff}} \cdot \frac{di}{dt}, \end{aligned}$$

which can be implemented as follows:



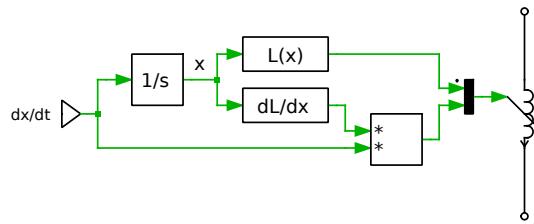
Note that in both cases the $\frac{d}{dt}L$ -input of the Variable Inductor is zero!

Actuator Modeling

In an actuator the inductivity is determined by an external quantity such as the position x of the movable core in a solenoid: $L = L(x)$. Therefore you have

$$\begin{aligned} v &= L \cdot \frac{di}{dt} + \frac{dL}{dt} \cdot i \\ &= L \cdot \frac{di}{dt} + \frac{dL}{dx} \cdot \frac{dx}{dt} \cdot i \quad , \end{aligned}$$

which can be implemented as follows:



Note that x is preferably calculated as the integral of dx/dt rather than calculating dx/dt as the derivative of x .

Parameters

Inductive coupling

Specifies whether the phases should be coupled inductively. This parameter determines how the elements of the control signal are interpreted. The default is off.

Initial current

The initial current through the inductor at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The direction of a positive initial current is indicated by a small arrow in the component symbol. The default of the initial current is 0.

Probe Signals

Inductor current

The current flowing through the inductor, in amperes (A). The direction of a positive current is indicated with a small arrow in the component symbol.

Inductor voltage

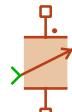
The voltage measured across the inductor, in volts (V).

Variable Magnetic Permeance

Purpose Variable permeance controlled by external signal

Library Magnetic

Description This component provides a magnetic flux path with a variable permeance. The component is used to model non-linear magnetic material properties such as saturation and hysteresis. The permeance is determined by the signal fed into the input of the component. The flux-rate through a variable permeance $\mathcal{P}(t)$ is governed by the equation:



$$\dot{\Phi} = \frac{d}{dt} (\mathcal{P} \cdot F) = \mathcal{P} \cdot \frac{dF}{dt} + \frac{d}{dt} \mathcal{P} \cdot F$$

Since F is the state variable the equation above must be solved for $\frac{dF}{dt}$. The control signal must provide the values of $\mathcal{P}(t)$, $\frac{d}{dt} \mathcal{P}(t)$ and Φ as a vector. It is the responsibility of the user to provide the appropriate signals.

Modeling non-linear material properties

When specifying the characteristic of a non-linear permeance, we need to distinguish carefully between the *total* permeance $\mathcal{P}_{\text{tot}}(F) = \Phi/F$ and the *differential* permeance $\mathcal{P}_{\text{diff}}(F) = d\Phi/dF$.

If the *total* permeance $\mathcal{P}_{\text{tot}}(F)$ is known the flux-rate $\dot{\Phi}$ through a time-varying permeance is calculated as:

$$\begin{aligned}\dot{\Phi} &= \frac{d\Phi}{dt} \\ &= \frac{d}{dt} (\mathcal{P}_{\text{tot}} \cdot F) \\ &= \mathcal{P}_{\text{tot}} \cdot \frac{dF}{dt} + \frac{d\mathcal{P}_{\text{tot}}}{dt} \cdot F \\ &= \mathcal{P}_{\text{tot}} \cdot \frac{dF}{dt} + \frac{d\mathcal{P}_{\text{tot}}}{dF} \cdot \frac{dF}{dt} \cdot F \\ &= \left(\mathcal{P}_{\text{tot}} + \frac{d\mathcal{P}_{\text{tot}}}{dF} \cdot F \right) \cdot \frac{dF}{dt}\end{aligned}$$

In this case, the control signal for the variable permeance component is:

$$\begin{bmatrix} \mathcal{P}(t) \\ \frac{d}{dt}\mathcal{P}(t) \\ \Phi(t) \end{bmatrix} = \begin{bmatrix} \mathcal{P}_{\text{tot}} + \frac{d}{dF}\mathcal{P}_{\text{tot}} \cdot F \\ 0 \\ \mathcal{P}_{\text{tot}} \cdot F \end{bmatrix}$$

In most cases, however, the *differential* permeance $\mathcal{P}_{\text{diff}}(F)$ is provided to characterize magnetic saturation and hysteresis. With

$$\begin{aligned} \dot{\Phi} &= \frac{d\Phi}{dt} \\ &= \frac{d\Phi}{dF} \cdot \frac{dF}{dt} \\ &= \mathcal{P}_{\text{diff}} \cdot \frac{dF}{dt} \end{aligned}$$

the control signal is

$$\begin{bmatrix} \mathcal{P}(t) \\ \frac{d}{dt}\mathcal{P}(t) \\ \Phi(t) \end{bmatrix} = \begin{bmatrix} \mathcal{P}_{\text{diff}} \\ 0 \\ \mathcal{P}_{\text{tot}} \cdot F \end{bmatrix}$$

Parameters

Initial MMF

Magneto-motive force at simulation start, in ampere-turns (A).

Probe Signals

MMF

The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).

Flux

The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.

Variable Resistor with Constant Capacitor

Purpose

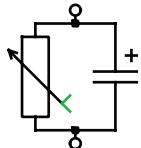
Controlled resistance in parallel with constant capacitance

Library

Electrical / Passive Components

Description

This component models a variable resistor with a constant capacitor connected in parallel. The resistance is determined by the signal fed into the input of the component. It may not be set to zero.



Note In this component the resistor is implemented as a voltage-dependent current source. Without the parallel capacitor, which fixes the momentary voltage, this would result in an algebraic loop. Therefore, the capacitance may not be set to zero.

Parameters

Capacitance

The value of the capacitor, in farads (F). All finite positive and negative values are accepted, excluding 0. The default is 100e-6.

In a vectorized component, all internal capacitors have the same value if the parameter is a scalar. To specify the capacitances individually use a vector $[C_1 \ C_2 \dots \ C_n]$. The length n of the vector determines the width of the component.

Initial voltage

The initial voltage of the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the width of the component. The positive pole is marked with a "+". The initial voltage default is 0.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with "+" is greater than the potential at the unmarked terminal.

Variable Resistor with Constant Inductor

Purpose

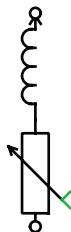
Controlled resistance in series with constant inductance

Library

Electrical / Passive Components

Description

This component models a variable resistor with a constant inductor connected in series. The resistance is determined by the signal fed into the input of the component.



Note In this component the resistor is implemented as a current-dependent voltage source. Without the series inductor, which fixes the momentary current, this would result in an algebraic loop. Therefore, the inductance may not be set to zero.

Parameters

Inductance

The inductance in henries (H). All finite positive and negative values are accepted, excluding 0. The default is 1e-3.

In a vectorized component, all internal inductors have the same inductance if the parameter is a scalar. To specify the inductances individually use a vector $[L_1 \ L_2 \dots \ L_n]$. The length n of the vector determines the width of the component.

Initial current

The initial current through the component at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the width of the component. The direction of a positive initial current is indicated by a small arrow in the component symbol. The default of the initial current is 0.

Probe Signals

Inductor current

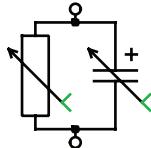
The current flowing through the inductor, in amperes (A). The direction of a positive current is indicated with a small arrow in the component symbol.

Variable Resistor with Variable Capacitor

Purpose Controlled resistance in parallel with controlled capacitance

Library Electrical / Passive Components

Description This component models a variable resistor with a variable capacitor connected in parallel. The resistance and capacitance are determined by the signals fed into the inputs of the component. The current through this component is determined by the equation



$$i = \left(\frac{1}{R} + \frac{d}{dt} C \right) \cdot v + C \cdot \frac{d}{dt} v$$

The control signal for the capacitor must provide the values of both C and $\frac{d}{dt} C$ in the following form: $[C_1 \ C_2 \ \dots \ C_n \ \frac{d}{dt} C_1 \ \frac{d}{dt} C_2 \ \dots \ \frac{d}{dt} C_n]$. It is the responsibility of the user to provide the appropriate signals for a particular purpose. For detailed information see the Variable Capacitor (on page 457).

If the component has multiple phases you can choose to include the capacitive coupling of the phases. In this case the control signal vector must contain the elements of the capacitance matrix (row by row) and their derivatives with respect to time. The control signal thus has a width of $2 \cdot n^2$, n being the number of phases.

Note The momentary capacitance and the resistance may not be set to zero. In case of coupled capacitors, the capacitance matrix may not be singular.

Parameters

Capacitive coupling

Specifies whether the phases should be coupled capacitively. This parameter determines how the elements of the control signal are interpreted. The default is off.

Initial voltage

The initial voltage of the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The positive pole is marked with a '+'. The initial voltage default is 0.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with “+” is greater than the potential at the unmarked terminal.

Variable Resistor with Variable Inductor

Purpose	Controlled resistance in series with controlled inductance
Library	Electrical / Passive Components
Description	<p>This component models a variable resistor with a variable inductor connected in series. The resistance and inductance are determined by the signals fed into the inputs of the component. The voltage across this component is determined by the equation</p> $v = \left(R + \frac{d}{dt} L \right) \cdot i + L \cdot \frac{d}{dt} i$ <p>The control signal for the inductor must provide the values of both L and $\frac{d}{dt} L$ in the following form: $[L_1 \ L_2 \dots L_n \ \frac{d}{dt} L_1 \ \frac{d}{dt} L_2 \dots \frac{d}{dt} L_n]$. It is the responsibility of the user to provide the appropriate signals for a particular purpose. For detailed information see the Variable Inductor (on page 460).</p> <p>If the component has multiple phases you can choose to include the inductive coupling of the phases. In this case the control signal vector must contain the elements of the inductivity matrix (row by row) and their derivatives with respect to time. The control signal thus has a width of $2 \cdot n^2$, n being the number of phases.</p>
	<p>Note</p> <ul style="list-style-type: none"> • The momentary inductance may not be set to zero. In case of coupled inductors, the inductivity matrix may not be singular. • The control signal for the momentary inductance values must be continuous. Discontinuous changes will produce non-physical results.

Parameters	<p>Inductive coupling Specifies whether the phases should be coupled inductively. This parameter determines how the elements of the control signal are interpreted. The default is off.</p> <p>Initial current The initial current through the component at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The direction of a positive initial current</p>
-------------------	---

is indicated by a small arrow in the component symbol. The default of the initial current is 0.

Probe Signals

Inductor current

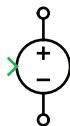
The current flowing through the inductor, in amperes (A). The direction of a positive current is indicated with a small arrow in the component symbol.

Voltage Source (Controlled)

Purpose Generate variable voltage

Library Electrical / Sources

Description The Controlled Voltage Source generates a variable voltage between its two electrical terminals. The voltage is considered positive at the terminal marked with a “+”. The momentary voltage is determined by the signal fed into the input of the component.



Note A voltage source may not be short-circuited or connected in parallel to a capacitor or any other voltage source.

Probe Signals

Source voltage

The source voltage in volts (V).

Source current

The current flowing through the source, in amperes (A).

Source power

The instantaneous output power of the source, in watts (W).

Voltage Source AC

Purpose	Generate sinusoidal voltage
Library	Electrical / Sources
Description	<p>The AC Voltage Source generates a sinusoidal voltage between its two electrical terminals. The voltage is considered positive at the terminal marked with a “+”. The momentary voltage v is determined by the equation</p> $v = A \cdot \sin(\omega \cdot t + \varphi)$ <p>where t is the simulation time.</p>
	<p>Note A voltage source may not be short-circuited or connected in parallel to a capacitor or any other voltage source.</p>
Parameters	<p>Each of the following parameters may either be a scalar or a vector corresponding to the implicit width of the component:</p> <p>Amplitude The amplitude A of the voltage, in volts (V). The default is 1.</p> <p>Frequency The angular frequency ω, in s^{-1}. The default is $2\pi \cdot 50$ which corresponds to 50 Hz.</p> <p>Phase The phase shift φ, in radians. The default is 0.</p>
Probe Signals	<p>Source voltage The source voltage in volts (V).</p> <p>Source current The current flowing through the source, in amperes (A).</p> <p>Source power The instantaneous output power of the source, in watts (W).</p>

Voltage Source AC (3-Phase)

Purpose	Generate 3-phase sinusoidal voltage
Library	Electrical / Sources
Description	<p>The three phase Voltage Source generates three sinusoidal voltages between its electrical terminals. The first phase is marked with a small black dot. The momentary voltages v_i are determined by the equation</p>  $v_i = A_i \cdot \sin(2\pi \cdot f \cdot t + \varphi_i + \Delta\varphi_i)$ <p>where t is the simulation time and $\varphi_0 = 0$, $\varphi_1 = -2/3 \cdot \pi$ and $\varphi_2 = 2/3 \cdot \pi$.</p>

Note A voltage source may not be short-circuited or connected in parallel to a capacitor or any other voltage source.

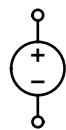
Parameters	<p>Amplitude The amplitude A of the voltage, in volts (V). The value can be given as a scalar or as a vector with three elements $[A_0, A_1, A_2]$.</p> <p>Frequency The frequency f, in Hertz (Hz).</p> <p>Phase offset The phase offset $\Delta\varphi$, in radians. The value can be given as a scalar or as a vector with three elements $[\Delta\varphi_0, \Delta\varphi_1, \Delta\varphi_2]$.</p> <p>Neutral point Show or hide the neutral point terminal.</p>
Probe Signals	<p>Source voltage The source voltages in volts (V) as a vectorized signal.</p> <p>Source current The currents flowing through the source, in amperes (A) as a vectorized signal.</p> <p>Source power The combined instantaneous output power of the source, in watts (W).</p>

Voltage Source DC

Purpose Generate constant voltage

Library Electrical / Sources

Description The DC Voltage Source generates a constant voltage between its two electrical terminals. The voltage is considered positive at the terminal marked with a “+”.



Note A voltage source may not be short-circuited or connected in parallel to a capacitor or any other voltage source.

Parameter **Voltage**

The magnitude of the constant voltage, in volts (V). This parameter may either be a scalar or a vector defining the width of the component. The default value is 1.

Probe Signals

Source voltage

The source voltage in volts (V).

Source current

The current flowing through the source, in amperes (A).

Source power

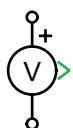
The instantaneous output power of the source, in watts (W).

Voltmeter

Purpose Output measured voltage as signal

Library Electrical / Meters

Description The Voltmeter measures the voltage between its two electrical terminals and provides it as a signal at the output of the component. A positive voltage is measured when the potential at the terminal marked with a “+” is greater than at the unmarked one. The output signal can be made accessible in Simulink with an Output block (see page 387) or by dragging the component into the dialog box of a Probe block.



Note The Voltmeter is ideal, i.e. it has an infinite internal resistance. Hence, if multiple voltmeters are connected in series the voltage across an individual voltmeter is undefined. This produces a run-time error.

Likewise, if switches connected in series are all in open position the voltages across the individual switches are not properly defined. Although this does not produce a run-time error it may lead to unexpected simulation results.

Probe Signals

Measured voltage

The measured voltage in volts (V).

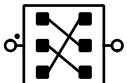
Winding

Purpose	Ideal winding defining an electro-magnetic interface
Library	Magnetic
Description	The Winding forms the interface between the electrical and the magnetic domain. A winding of N turns is described with the equations:  $v = N \dot{\Phi}$ $i = \frac{F}{N}$
	where v and i are voltage and current at its electrical terminals. F is the magneto-motive force (MMF) and $\dot{\Phi}$ is the rate-of-change of the magnetic flux through the winding. The left-hand side of the equations above refers to the electrical domain, the right-hand side to the magnetic domain. A current entering the winding at the marked terminal is counted as positive; a magnetic flux entering at the marked terminal is counted as negative. The potentials of both voltage and MMF are considered positive at the marked terminal. Because the Winding converts through-quantities ($\dot{\Phi}$ resp. i) in one domain into across-quantities (v resp. F) in the other domain, it is implemented as a gyrator, in which N is the gyrator resistance.
Parameters	Number of turns Specifies the number of winding turns.
Probe Signals	Winding voltage The voltage measured from the positive (marked) to the negative electrical terminal of the winding, in volts (V). Winding current The current flowing through the winding, in amperes (A). A current entering the winding at the marked terminal is counted as positive. MMF The magneto-motive force measured from the marked to the unmarked magnetic terminal, in ampere-turns (A).

Wire Multiplexer

Purpose	Bundle several wires into bus
Library	System
Description	This multiplexer combines several individual wires into a wire bus. The individual wires may themselves be buses. In the block icon, the first individual wire is marked with a dot.
	
Parameter	Width This parameter allows you to specify the number and/or width of the individual wires. You can choose between the following formats for this parameter: Scalar: A scalar specifies the number of individual wires each having a width of 1. Vector: The length of the vector determines the number of individual wires. Each element specifies the width of the corresponding individual wire.

Wire Selector

Purpose	Select or reorder elements from wire bus
Library	System
Description	The Wire Selector block connects the individual elements of the output bus to the specified elements of the input bus. The input bus is marked with a dot.
	
Parameters	Input width The width of the input bus. Output indices A vector with the indices of the input elements that the output bus should contain.

XY Plot

Purpose	Display correlation between two signals
Library	System
Description	The XY Plot displays the relationship between two signals. It can be used in PLECS circuits as well as in Simulink models. For detailed information on how to work with the XY Plot see section “Using the XY Plot” (on page 71).
	
Parameters	<p>Title The name which is displayed above the plot.</p> <p>Limit samples If this option is selected, the XY Plot will only save the last n sample values during a simulation. It can be used in long simulations to limit the amount of memory that is used by PLECS. If the option is unchecked all sample values are stored in memory.</p> <p>Time range This option may be used for periodic systems to limit the displayed data to a given number of periods. The time range value determines the time range that is displayed in the plot. If set to auto, the data over the whole simulation time range is used. If a limit is given and the simulation time reaches an integer multiple of this limit the plot is cleared except for the data covering the last n time ranges, where n is the number entered under Show last. The plot is appended until the simulation time reaches the next integer multiple of the time range.</p> <p>Axis labels The axis labels that are displayed on the x- and y-axis.</p> <p>X- and Y-limits The initial lower and upper bound of the x- and y-axis. If set to auto, the axes are automatically scaled such that all data is visible. Note that setting any of the limits to auto is computationally expensive and may have a considerable impact on the simulation speed.</p>

Zener Diode

Purpose	Zener diode with controlled reverse breakdown voltage
Library	Electrical / Power Semiconductors
Description	The Zener diode is a type of diode that permits current to flow in forward direction like a normal diode (see page 232), but also in reverse direction if the voltage is larger than the rated breakdown or Zener voltage. Zener diodes are widely used to regulate the voltage across a circuit.
	
Parameters	<p>Zener voltage Breakdown voltage V_z in reverse direction, in volts (V). If the diode is reverse conducting the voltage drop across the diode is determined by this Zener voltage plus the voltage across the Zener resistance.</p> <p>Zener resistance The resistance R_z, in ohms (Ω), if the diode is reverse conducting.</p> <p>Forward voltage Additional dc voltage V_f in volts (V) between anode and cathode when the diode is forward conducting. The default is 0.</p> <p>On-resistance The resistance R_f of the forward conducting device, in ohms (Ω). The default is 0.</p>
Probe Signals	<p>Diode voltage The voltage measured between anode and cathode.</p> <p>Diode current The current through the diode flowing from anode to cathode.</p> <p>Forward conductivity Conduction state of the positive internal switch. The signal outputs 1 when the diode is conducting in forward direction, and 0 otherwise.</p> <p>Reverse conductivity Conduction state of the negative internal switch. The signal outputs 1 when the diode is conducting in reverse direction, and 0 otherwise.</p>

Zero Order Hold

Purpose	Sample and hold input signal periodically
Library	Control / Discrete
Description	The Zero Order Hold samples the input signal and holds this value at its output for a specified sample time.
	
Parameter	Sample time The length of the hold time in seconds. See also the Discrete-Periodic sample time type in section “Sample Times” (on page 32).
Probe Signals	Input The input signal. Output The output signal.

Additional Simulink Blocks

This chapter lists the contents of the PLECS Extras library in alphabetical order.

AC Sweep

Purpose	Perform AC sweep
Library	PLECS Extras / Analysis Tools
Description	<p>The AC Sweep block enables you to determine the transfer function of a generic system from a single input to one or more outputs. The analysis is performed by injecting a small sinusoidal signal at different frequencies into the system and extracting the same frequencies from the system output(s) by Fourier analysis. The perturbation signal is available at the block output. The system outputs to be analyzed must be fed into the block's input port.</p> <p>An ac sweep can be started either by clicking the button Start analysis or with the MATLAB command</p>
	<pre>placsweep(block);</pre>
	<p>where <i>block</i> is the Simulink handle or the full block path of the AC Sweep block. The block handle or path can be followed by parameter/value pairs that override the settings in the dialog box.</p> <p>For additional information see section “AC Analysis” (on page 107).</p>

Parameters

System period length

The period length of the unperturbed system.

Simulation start time

The simulation start time for the ac sweep.

Frequency sweep range

A vector containing the lowest and highest perturbation frequency.

Frequency sweep scale

Specifies whether the sweep frequencies should be distributed on a linear or logarithmic scale.

Number of points

The number of data points generated.

Amplitude at first freq

The amplitude of the perturbation signal at the lowest frequency. The amplitudes at the other frequencies are calculated as

$$A_i = A_1 \cdot \sqrt{f_i/f_1}$$

Method

Specifies the method to use for obtaining the steady-state operating point of the system for each perturbation frequency.

Brute force simulation simply simulates the system on a cycle-by-cycle basis until the difference between the state variables at the beginning and end of a cycle become sufficiently small. With this setting the parameter **Max number of iterations** actually limits the number of cycles until a steady state is reached.

Steady-state analysis performs a steady-state analysis for each perturbation frequency.

Start from model initial state uses the initial state values specified in the model – either in the individual blocks or in the simulation parameters.

Start from unperturbed steady state performs a steady-state analysis of the unperturbed system to determine the initial state vector for the ac sweep.

Termination tolerance

The relative error bound for all state variables. The analysis continues until

$$\frac{|x(t_0) - x(t_0 + T)|}{\max |x|} \leq rtol$$

for each state variable.

Max number of iterations

The maximum number of iterations allowed.

Output variable

The name of a MATLAB variable used to store the transfer function at the end of an analysis. If the analysis was run interactively from the GUI, the variable is assigned in the MATLAB base workspace. If the analysis was run with the `placsweep` command, the variable is assigned in the caller's workspace.

Plot bode diagram

Specifies whether to plot the transfer function in a bode diagram.

Display level

Specifies the level of detail of the diagnostic messages displayed in the command window (`iteration`, `final`, `off`).

Hidden model states

Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (error, warning, none).

Discrete Analysis

Please refer to the documentation on the following components:

- Discrete Fourier Transform (see page 238)
- Discrete Mean Value (see page 239)
- Discrete RMS Value (see page 240)
- Discrete Total Harmonic Distortion (see page 241)

Impulse Response Analysis

Purpose	Perform impulse response analysis
Library	PLECS Extras / Analysis Tools
Description	<p>The Impulse Response Analysis block enables you to determine the transfer function of a generic system from a single input to one or more outputs. The analysis is performed by injecting a small rectangular pulse into the system and computing the inverse Laplace transform of the system response(s). The perturbation signal is available at the block output. The system outputs to be analyzed must be fed into the block's input port.</p> <p>An analysis can be started either by clicking the button Start analysis or with the MATLAB command</p>
	<pre>plimpulseresponse(block);</pre>
	<p>where <i>block</i> is the Simulink handle or the full block path of the Impulse Response Analysis block. The block handle or path can be followed by parameter/value pairs that override the settings in the dialog box.</p> <p>For additional information see section “Impulse Response Analysis” (on page 108).</p>
Parameters	<p>System period length The period length of the unperturbed system.</p> <p>Simulation start time The simulation start time for the impulse response analysis.</p> <p>Frequency sweep range A vector containing the lowest and highest perturbation frequency.</p> <p>Frequency sweep scale Specifies whether the sweep frequencies should be distributed on a linear or logarithmic scale.</p> <p>Number of points The number of data points generated.</p> <p>Perturbation The amplitude of the perturbation signal.</p>

Compensation for discrete pulse

Specifies whether and how the effect of the sampling should be compensated. See section “Compensation for Discrete Pulse” (on page 108) for an explanation of the parameter values.

Termination tolerance

The relative error bound used in the initial steady-state analysis.

Max number of iterations

The maximum number of iterations allowed during the initial steady-state analysis.

Output variable

The name of a MATLAB variable used to store the transfer function at the end of an analysis. If the analysis was run interactively from the GUI, the variable is assigned in the MATLAB base workspace. If the analysis was run with the placsweep command, the variable is assigned in the caller’s workspace.

Plot bode diagram

Specifies whether to plot the transfer function in a bode diagram.

Display level

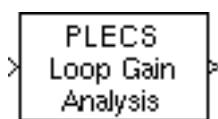
Specifies the level of detail of the diagnostic messages displayed in the command window (iteration, final, off).

Hidden model states

Specifies how to handle Simulink blocks with ‘hidden’ states, i.e. states that are not stored in the state vector (error, warning, none).

Loop Gain Analysis

Purpose	Determine loop gain of closed control loop
Library	PLECS Extras / Analysis Tools
Description	<p>The Loop Gain Analysis block enables you to determine the gain of a closed control loop. To measure the loop gain, insert the block anywhere in the control loop. The loop gain is determined by adding a small sinusoidal signal at various frequencies and extracting the same frequencies from the system before and after the summation point by Fourier analysis.</p> <p>An analysis can be started either by clicking the button Start analysis or with the MATLAB command</p>
	<pre>placsweep(block);</pre>



where *block* is the Simulink handle or the full block path of the Loop Gain Analysis block. Otherwise, the block remains inactive and does not influence the control loop.

For additional information see section “AC Analysis” (on page 107).

Note The Loop Gain Analysis block works only on scalar signals. In order to analyze the gain of a vectorized control loop you need to demultiplex the vector signal into individual scalar signals before inserting the Loop Gain Analysis block.

Parameters	The parameters are identical to those of the AC Sweep block (see page 484).
-------------------	---

Modulators

Please refer to the documentation on the following components:

- 2-Pulse Generator (see page 192)
- 3-Phase Overmodulation (see page 194)
- 6-Pulse Generator (see page 195)
- Blanking Time (see page 200)
- Blanking Time (3-Level) (see page 201)
- Sawtooth PWM (see page 375)
- Sawtooth PWM (3-Level) (see page 377)
- Symmetrical PWM (see page 409)
- Symmetrical PWM (3-Level) (see page 411)
- Peak Current Controller (see page 342)

Steady-State Analysis

Purpose Determine periodic steady-state operating point

Library PLECS Extras / Analysis Tools

Description The Steady-State Analysis block enables you to determine the steady-state operating point of a generic periodic system. Copy this block anywhere into the model that you want to analyze.

PLECS
Steady-State
Analysis

A steady-state analysis can be started either by clicking the button **Start analysis** or with the MATLAB command

```
plsteadystate(block);
```

where *block* is the Simulink handle or the full block path of the Steady-State Analysis block. The block handle or path can be followed by parameter/value pairs that override the settings in the dialog box.

For additional information see section “Steady-State Analysis” (on page 105).

Parameters

System period

Specifies whether the system period is **fixed**, i.e. predetermined and constant, or **variable** (e.g. in case of a hysteresis type controller). If **variable** is selected, a trigger input will be drawn which is used to determine the end of a period.

Trigger type

Specifies which trigger event on the input signal (**rising**, **falling**) marks the end of a variable system period.

System period length/Max simulation time span

For a fixed system period, the period length; for a variable system period, the maximum time span during which to look for a trigger event marking the end of a period.

Simulation start time

The simulation start time for the steady-state analysis.

Termination tolerance

The relative error bound. The analysis continues until both the maximum relative error in the state variables and the maximum relative change from one iteration to the next are smaller than this bound for each state variable.

Max number of iterations

The maximum number of iterations allowed.

Steady-state variable

The name of a MATLAB variable used to store the periodic steady-state vector at the end of an analysis. If the analysis was run interactively from the GUI, the variable is assigned in the MATLAB base workspace. If the analysis was run with the `p1steadystate` command, the variable is assigned in the caller's workspace.

Show steady-state cycles

The number of cycles shown in the Simulink scopes at the end of an analysis.

Display level

Specifies the level of detail (`iteration`, `final`, `off`) of the diagnostic messages displayed in the command window.

Hidden model states

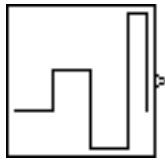
Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (`error`, `warning`, `none`).

Timer

Purpose Generate piece-wise constant signal

Library PLECS Extras / Control Blocks

Description The Timer block generates a signal that changes at discrete instants and is otherwise constant. You can use the Timer block e.g. in order to control switches such as circuit breakers.



Note This block is only available for MATLAB 7.0 or newer.

Parameters

Time values

A vector containing the transition times. This vector must have the same length as the vector of output values. Before the first transition time the output is zero.

Output values

A vector containing the output values corresponding to the transition times. This vector must have the same length as the vector of time values.

Transformations

Please refer to the documentation on the following components:

- Transformation 3ph->RRF (see page 438)
- Transformation 3ph->SRF (see page 439)
- Transformation RRF->3ph (see page 440)
- Transformation RRF->SRF (see page 441)
- Transformation SRF->3ph (see page 442)
- Transformation SRF->RRF (see page 443)

Plexim GmbH +41 44 445 24 10 info@plexim.com www.plexim.com

