



**THE SIMULATION PLATFORM FOR  
POWER ELECTRONIC SYSTEMS**

**User Manual** Version 4.3

## How to Contact Plexim:

☎	+41 44 533 51 00	Phone
	+41 44 533 51 01	Fax
✉	Plexim GmbH Technoparkstrasse 1 8005 Zurich Switzerland	Mail
@	info@plexim.com	Email
	<a href="http://www.plexim.com">http://www.plexim.com</a>	Web

### *PLECS User Manual*

© 2002–2019 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 Simulink Coder are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.

# Contents

<b>Contents</b>	<b>iii</b>
<b>Before You Begin</b>	<b>1</b>
Installing PLECS Blockset . . . . .	1
Installation on Microsoft Windows . . . . .	1
Installation on macOS . . . . .	1
Installation on Linux . . . . .	2
Licensing . . . . .	2
Configuring the MATLAB Search Path . . . . .	3
Configuring PLECS . . . . .	3
Installing Different Versions of PLECS Blockset in Parallel . . . . .	4
Uninstalling PLECS Blockset . . . . .	4
Installing PLECS Standalone . . . . .	5
Installation on Microsoft Windows . . . . .	5
Installation on macOS . . . . .	5
Installation on Linux . . . . .	5
Licensing . . . . .	6
License File Location . . . . .	6
Network Licensing . . . . .	7
What's New in Version 4.3 . . . . .	8
Major New Features . . . . .	8
Further Enhancements . . . . .	8

<b>1</b>	<b>Getting Started</b>	<b>9</b>
	Getting Started with 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
<b>2</b>	<b>How PLECS Works</b>	<b>25</b>
	Modeling Dynamic Systems . . . . .	25
	System Equations . . . . .	26
	Block Diagrams . . . . .	26
	Physical Models . . . . .	27
	Simulating Dynamic Systems . . . . .	27
	Model Initialization . . . . .	28
	Model Execution . . . . .	31
	Fixed-Step Simulation . . . . .	33
	Sampled Data Systems . . . . .	36
	Sample Times . . . . .	36
	Sample Time Inheritance . . . . .	37
	Multirate Systems . . . . .	38
	Troubleshooting . . . . .	38
<b>3</b>	<b>Using PLECS</b>	<b>41</b>
	Configuring PLECS . . . . .	41
	General . . . . .	41
	Libraries . . . . .	42
	Thermal . . . . .	43
	Scope Colors . . . . .	43
	Update . . . . .	43
	Coder . . . . .	44

---

Installing Extensions . . . . .	45
Creating a New Circuit with PLECS Blockset . . . . .	46
Customizing the Circuit Block . . . . .	47
Using the Library Browser . . . . .	48
Components . . . . .	50
Specifying Component Parameters . . . . .	50
Displaying Parameters in the Schematic . . . . .	51
Changing Parameters of Multiple Components . . . . .	51
Changing Parameters During a Simulation . . . . .	51
Changing Component Names . . . . .	52
Changing the Orientation of Components . . . . .	52
Getting Component Help . . . . .	52
Libraries . . . . .	53
Creating a New Library in PLECS Blockset . . . . .	53
Creating a New Library in PLECS Standalone . . . . .	53
Creating a Library Reference . . . . .	53
Updating a Library Reference . . . . .	54
Breaking a Library Reference . . . . .	54
Connections . . . . .	55
Signal Connections and Physical Connections . . . . .	55
Creating Connections . . . . .	55
Creating Branches . . . . .	55
Automatic Creation of Connections . . . . .	56
Editing Connections . . . . .	57
Annotations . . . . .	57
Subsystems . . . . .	58
Creating a Subsystem by Adding the Subsystem Block . . . . .	58
Creating a Subsystem by Grouping Existing Blocks . . . . .	58
Arranging Subsystem Terminals . . . . .	59
Resizing a Subsystem Block . . . . .	59
Placing the Subsystem Label . . . . .	60

Masking Subsystems . . . . .	61
Mask Icon . . . . .	61
Mask Dialog . . . . .	66
Mask Workspace . . . . .	70
Mask Probe Signals . . . . .	72
Mask Documentation . . . . .	73
Unprotecting Masked Subsystems . . . . .	74
Getting Started with Lua . . . . .	74
Circuit Browser . . . . .	79
Viewing Options . . . . .	80
PLECS Probe . . . . .	82
Copying a Probe . . . . .	83
Assertions . . . . .	85
Assertion Blocks . . . . .	86
Component Assertions . . . . .	86
Locating Assertions . . . . .	87
Controlling Access to Circuits and Subsystems . . . . .	88
Encrypting Circuits and Subsystems . . . . .	88
Exporting Circuits for the PLECS Viewer . . . . .	89
Exporting Schematics . . . . .	90
Using the PLECS Scope . . . . .	91
Getting Started . . . . .	91
Zoom Operations . . . . .	92
Scrolling . . . . .	93
Y-Axis Auto-Scaling . . . . .	93
Changing Curve Properties . . . . .	94
Spreading Signals . . . . .	94
Cursors . . . . .	94
Fourier Analysis . . . . .	96
Saving a View . . . . .	96
Adding Traces . . . . .	96

Saving and Loading Trace Data . . . . .	96
Scope Parameters . . . . .	97
Printing and Exporting . . . . .	97
Using the Fourier Analysis . . . . .	98
Calculation Parameters . . . . .	98
Display Parameters . . . . .	99
Zoom, Export and Print . . . . .	100
Calculation of the Fourier coefficients . . . . .	100
Using the XY Plot . . . . .	101
Time Range Window . . . . .	101
Zoom, Save View, Export and Print . . . . .	102
Simulation Parameters . . . . .	103
PLECS Blockset Parameters . . . . .	103
PLECS Standalone Parameters . . . . .	107
<b>4 Thermal Modeling</b>	<b>115</b>
Heat Sink Concept . . . . .	115
Implementation . . . . .	116
Thermal Loss Dissipation . . . . .	116
Semiconductor Losses . . . . .	116
Ohmic Losses . . . . .	120
Heat Sinks and Subsystems . . . . .	120
Temperature Initialization . . . . .	121
Thermal Description Parameter . . . . .	123
Assigning Thermal Data Sheets . . . . .	123
Using Reference Variables . . . . .	123
Thermal Library . . . . .	126
Library Structure . . . . .	126
Global and Local Data Sheets . . . . .	126
Creating New Data Sheets . . . . .	127
Browsing the Thermal Library . . . . .	127

Thermal Editor . . . . .	128
Editing Switching Losses . . . . .	129
Editing Conduction Losses . . . . .	130
Editing the Thermal Equivalent Circuit . . . . .	131
Adding Custom Variables . . . . .	132
Adding Custom Lookup Tables . . . . .	133
Editing Lookup Tables . . . . .	134
Importing Data from Graphical Datasheets . . . . .	136
Semiconductor Loss Specification . . . . .	141
Single Semiconductor Switch Losses . . . . .	141
Diode Losses . . . . .	141
Losses of Semiconductor Switch with Diode . . . . .	142
<b>5 Magnetic Modeling</b>	<b>145</b>
Equivalent circuits for magnetic components . . . . .	145
Coupled inductors . . . . .	146
Reluctance-resistance analogy . . . . .	146
Permeance-capacitance analogy . . . . .	148
Magnetic Circuit Domain in PLECS . . . . .	149
Modeling Non-Linear Magnetic Material . . . . .	150
Saturation Curves for Soft-Magnetic Material . . . . .	151
<b>6 Mechanical Modeling</b>	<b>153</b>
Flanges and Connections . . . . .	154
Force/Torque Flows and Sign Conventions . . . . .	154
Positions and Angles . . . . .	155
Initial Conditions . . . . .	155
Angle Wrapping . . . . .	156
Ideal Clutches . . . . .	156
Inelastic Collisions . . . . .	157



<b>7 Analysis Tools</b>	<b>161</b>
Steady-State Analysis . . . . .	161
Algorithm . . . . .	161
Fast Jacobian Calculation for Thermal States . . . . .	162
Non-periodic Case . . . . .	163
Limitations . . . . .	163
Reference . . . . .	163
AC Analysis . . . . .	164
Impulse Response Analysis . . . . .	164
Algorithm . . . . .	164
Compensation for Discrete Pulse . . . . .	165
Reference . . . . .	165
Multitone Analysis . . . . .	166
Algorithm . . . . .	166
Remarks . . . . .	167
References . . . . .	167
Usage in PLECS Standalone . . . . .	168
Steady-State Analysis . . . . .	168
AC Sweep . . . . .	170
Impulse Response Analysis . . . . .	171
Multitone Analysis . . . . .	171
Extraction of State-Space Matrices . . . . .	172
Application Example . . . . .	173
Usage in PLECS Blockset . . . . .	176
Steady-State Analysis . . . . .	176
AC Sweep / Loop Gain Analysis . . . . .	178
Impulse Response Analysis . . . . .	181
Multitone / Loop Gain Analysis . . . . .	183
Extraction of State-Space Matrices . . . . .	184
Application Example . . . . .	185

<b>8 C-Scripts</b>	<b>195</b>
How C-Scripts Work . . . . .	195
C-Script Functions . . . . .	196
Modeling Discontinuities . . . . .	198
Sample Time . . . . .	200
User Parameters . . . . .	202
Runtime Checks . . . . .	203
C-Script Examples . . . . .	203
A Simple Function – Times Two . . . . .	204
Discrete States – Sampled Delay . . . . .	204
Continuous States – Integrator . . . . .	205
Event Handling – Wrapping Integrator . . . . .	205
Piecewise Smooth Functions – Saturation . . . . .	206
Multiple Sample Times – Turn-on Delay . . . . .	208
C-Script Macros . . . . .	210
Deprecated Macros . . . . .	213
<b>9 State Machines</b>	<b>215</b>
Working with State Machines . . . . .	216
Working with States . . . . .	217
Working with Transitions . . . . .	218
Working with Junctions . . . . .	220
Working with Annotations . . . . .	220
State Machine Configuration . . . . .	221
State Machine Execution . . . . .	225
Transition Evaluation . . . . .	225
Trigger Types . . . . .	226
Trigger Lifetime . . . . .	227
Execution of Hierarchical State Machines . . . . .	228
State Machine Examples . . . . .	229
Oven Control . . . . .	229
Constant On-Time Control . . . . .	231

<b>10 Simulation Scripts</b>	<b>233</b>
Command Line Interface in PLECS Blockset . . . . .	233
Simulation Scripts in PLECS Standalone . . . . .	239
Overview of PLECS Scripting Extensions . . . . .	240
Example Script . . . . .	246
XML-RPC Interface in PLECS Standalone . . . . .	247
Establishing an XML-RPC Connection to PLECS . . . . .	247
Overview of XML-RPC Commands . . . . .	247
Example Script . . . . .	250
Scripted Simulation and Analysis Options . . . . .	251
<b>11 Code Generation</b>	<b>257</b>
Code Generation for Physical Systems . . . . .	257
Reducing the Code Size . . . . .	258
Maximum Number of Switches . . . . .	258
Handling Naturally Commutated Devices . . . . .	258
Data Types . . . . .	261
Unsupported Components . . . . .	263
Code Generation with PLECS Standalone . . . . .	264
Generating Code . . . . .	264
Simulating a Subsystem in CodeGen Mode . . . . .	268
Code Generation with PLECS Blockset . . . . .	269
Standalone Code Generation . . . . .	269
Integration with Simulink Coder . . . . .	269
Simulink Coder Options . . . . .	269
Code Generation Targets . . . . .	270
Real-Time Target . . . . .	271
Rapid Simulation Target . . . . .	271

<b>12 Processor-in-the-Loop</b>	<b>275</b>
Motivation . . . . .	275
How PIL Works . . . . .	276
PIL Modes . . . . .	278
Configuring PLECS for PIL . . . . .	279
Target Manager . . . . .	279
Communication Links . . . . .	280
PIL Block . . . . .	282
<b>13 Components by Category</b>	<b>287</b>
System . . . . .	287
Assertions . . . . .	288
Control . . . . .	288
Sources . . . . .	288
Math . . . . .	289
Continuous . . . . .	289
Delays . . . . .	290
Discontinuous . . . . .	290
Discrete . . . . .	290
Filters . . . . .	291
Functions & Tables . . . . .	291
Logical . . . . .	292
Modulators . . . . .	292
Transformations . . . . .	293
State Machine . . . . .	293
Small Signal Analysis . . . . .	294
Electrical . . . . .	294
Sources . . . . .	294
Meters . . . . .	294
Passive Components . . . . .	294
Power Semiconductors . . . . .	295

Power Modules . . . . .	296
Switches . . . . .	297
Transformers . . . . .	297
Machines . . . . .	298
Converters . . . . .	299
Electronics . . . . .	299
Model Settings . . . . .	299
Thermal . . . . .	299
Magnetic . . . . .	300
Mechanical . . . . .	301
Translational . . . . .	301
Rotational . . . . .	302
Additional Simulink Blocks . . . . .	303
<b>14 Component Reference</b>	<b>305</b>
1D Look-Up Table . . . . .	306
2D Look-Up Table . . . . .	308
2-Pulse Generator . . . . .	309
3D Look-Up Table . . . . .	310
3-Phase Overmodulation . . . . .	312
6-Pulse Generator . . . . .	313
Abs . . . . .	314
Algebraic Constraint . . . . .	315
Ambient Temperature . . . . .	316
Air Gap . . . . .	317
Ammeter . . . . .	318
Angle Sensor . . . . .	319
Assert Dynamic Lower Limit . . . . .	320
Assert Dynamic Range . . . . .	321
Assert Dynamic Upper Limit . . . . .	322
Assertion . . . . .	323

Assert Lower Limit . . . . .	324
Assert Range . . . . .	325
Assert Upper Limit . . . . .	326
Blanking Time . . . . .	327
Blanking Time (3-Level) . . . . .	328
Breaker . . . . .	329
Brushless DC Machine . . . . .	330
Brushless DC Machine (Simple) . . . . .	333
C-Script . . . . .	337
Capacitor . . . . .	340
Clock . . . . .	342
Combinatorial Logic . . . . .	343
Comparator . . . . .	344
Compare to Constant . . . . .	345
Configurable Subsystem . . . . .	346
Constant . . . . .	347
Constant Heat Flow . . . . .	348
Constant Temperature . . . . .	349
Controlled Heat Flow . . . . .	350
Controlled Temperature . . . . .	351
Current Source (Controlled) . . . . .	352
Current Source AC . . . . .	353
Current Source DC . . . . .	354
D Flip-flop . . . . .	355
Data Type . . . . .	356
DC Machine . . . . .	357
Dead Zone . . . . .	359
Delay . . . . .	360
Diode . . . . .	361
Diode with Reverse Recovery . . . . .	363
Diode Rectifier (3ph) . . . . .	366

Discrete Fourier Transform . . . . .	367
Discrete Integrator . . . . .	368
Discrete Mean Value . . . . .	372
Discrete RMS Value . . . . .	373
Discrete State Space . . . . .	374
Discrete Total Harmonic Distortion . . . . .	375
Discrete Transfer Function . . . . .	376
Display . . . . .	378
DLL . . . . .	379
Double Switch . . . . .	383
Edge Detection . . . . .	384
Electrical Algebraic Component . . . . .	385
Electrical Ground . . . . .	387
Electrical Label . . . . .	388
Electrical Model Settings . . . . .	389
Electrical Port . . . . .	390
Enable . . . . .	392
Flux Rate Meter . . . . .	393
Force (Constant) . . . . .	394
Force (Controlled) . . . . .	395
Force Sensor . . . . .	396
Fourier Series . . . . .	397
Function . . . . .	398
Gain . . . . .	399
Gear . . . . .	400
GTO . . . . .	401
GTO (Reverse Conducting) . . . . .	403
Heat Flow Meter . . . . .	405
Heat Sink . . . . .	406
Hit Crossing . . . . .	407
Hysteretic Core . . . . .	408

Ideal 3-Level Converter (3ph) . . . . .	410
Ideal Converter (3ph) . . . . .	411
Ideal Transformer . . . . .	412
IGBT . . . . .	414
IGBT 3-Level Converter (3ph) . . . . .	416
IGBT 3-Level Half Bridge (NPC) . . . . .	418
IGBT Chopper (High-Side Switch) . . . . .	420
IGBT Chopper (High-Side Switch with Reverse Diode) . . . . .	421
IGBT Chopper (Low-Side Switch) . . . . .	423
IGBT Chopper (Low-Side Switch with Reverse Diode) . . . . .	424
IGBT Converter (3ph) . . . . .	426
IGBT Full Bridges (Series Connected) . . . . .	428
IGBT Half Bridge . . . . .	430
IGBT Half Bridges (Low-/High-Side Connected) . . . . .	432
IGBT with Diode . . . . .	434
IGBT with Limited di/dt . . . . .	436
IGCT (Reverse Blocking) . . . . .	440
IGCT (Reverse Conducting) . . . . .	442
Induction Machine (Slip Ring) . . . . .	444
Induction Machine (Open Stator Windings) . . . . .	448
Induction Machine (Squirrel Cage) . . . . .	451
Induction Machine with Saturation . . . . .	454
Inductor . . . . .	460
Inertia . . . . .	462
Initial Condition . . . . .	463
Integrator . . . . .	464
JK Flip-flop . . . . .	466
Leakage Flux Path . . . . .	468
Linear Core . . . . .	469
Linear Transformer (2 Windings) . . . . .	470
Linear Transformer (3 Windings) . . . . .	472



---

Logical Operator . . . . .	474
Magnetic Permeance . . . . .	475
Magnetic Port . . . . .	476
Magnetic Resistance . . . . .	478
Manual Double Switch . . . . .	479
Manual Signal Switch . . . . .	480
Manual Switch . . . . .	481
Manual Triple Switch . . . . .	482
Mass . . . . .	483
Math Function . . . . .	484
Memory . . . . .	485
Meter (3-Phase) . . . . .	486
Minimum / Maximum . . . . .	487
MMF Meter . . . . .	488
MMF Source (Constant) . . . . .	489
MMF Source (Controlled) . . . . .	490
Monoflop . . . . .	491
MOSFET . . . . .	493
MOSFET Converter (3ph) . . . . .	495
MOSFET with Diode . . . . .	496
MOSFET with Limited di/dt . . . . .	498
Moving Average . . . . .	500
Mutual Inductor . . . . .	501
Mutual Inductance (2 Windings) . . . . .	503
Mutual Inductance (3 Windings) . . . . .	505
Non-Excited Synchronous Machine . . . . .	507
Offset . . . . .	511
Op-Amp . . . . .	512
Op-Amp with Limited Output . . . . .	513
Pause / Stop . . . . .	514
Peak Current Controller . . . . .	515

Periodic Average . . . . .	516
Periodic Impulse Average . . . . .	517
Permanent Magnet Synchronous Machine . . . . .	518
Pi-Section Line . . . . .	522
Piece-wise Linear Resistor . . . . .	524
Planetary Gear Set . . . . .	526
Polar to Rectangular . . . . .	528
Position Sensor . . . . .	529
Product . . . . .	530
Pulse Delay . . . . .	531
Pulse Generator . . . . .	533
Quantizer . . . . .	534
Rack and Pinion . . . . .	535
Ramp . . . . .	536
Random Numbers . . . . .	537
Rate Limiter . . . . .	538
Rectangular to Polar . . . . .	539
Relational Operator . . . . .	540
Relay . . . . .	541
Resistor . . . . .	542
Rotational Algebraic Component . . . . .	543
Rotational Backlash . . . . .	545
Rotational Clutch . . . . .	546
Rotational Damper . . . . .	547
Rotational Friction . . . . .	548
Rotational Hard Stop . . . . .	550
Rotational Model Settings . . . . .	552
Rotational Port . . . . .	553
Rotational Reference . . . . .	554
Rotational Speed (Constant) . . . . .	555
Rotational Speed (Controlled) . . . . .	556

Rotational Speed Sensor . . . . . 557

Rounding . . . . . 558

Saturable Capacitor . . . . . 559

Saturable Core . . . . . 561

Saturable Inductor . . . . . 564

Saturable Transformers . . . . . 566

Saturation . . . . . 568

Sawtooth PWM . . . . . 569

Sawtooth PWM (3-Level) . . . . . 571

Scope . . . . . 573

Set/Reset Switch . . . . . 575

Signal Demultiplexer . . . . . 577

Signal From . . . . . 578

Signal Goto . . . . . 579

Signal Inport . . . . . 580

Signal Multiplexer . . . . . 582

Signal Outport . . . . . 583

Signal Selector . . . . . 584

Signal Switch . . . . . 585

Signum . . . . . 586

Sine Wave . . . . . 587

Small Signal Gain . . . . . 588

Small Signal Perturbation . . . . . 589

Small Signal Response . . . . . 590

Space Vector PWM . . . . . 591

Space Vector PWM (3-Level) . . . . . 595

SR Flip-flop . . . . . 599

State Machine . . . . . 600

State Space . . . . . 601

Step . . . . . 602

Subsystem . . . . . 603

Sum . . . . .	606
Switch . . . . .	607
Switched Reluctance Machine . . . . .	608
Symmetrical PWM . . . . .	612
Symmetrical PWM (3-Level) . . . . .	614
Synchronous Machine (Round Rotor) . . . . .	616
Synchronous Machine (Salient Pole) . . . . .	621
Synchronous Reluctance Machine . . . . .	627
Thermal Capacitor . . . . .	631
Thermal Chain . . . . .	632
Thermal Ground . . . . .	633
Thermal Model Settings . . . . .	634
Thermal Port . . . . .	635
Thermal Resistor . . . . .	637
Thermometer . . . . .	638
Thyristor . . . . .	639
Thyristor Rectifier/Inverter . . . . .	641
Thyristor with Reverse Recovery . . . . .	642
To File . . . . .	644
Torque (Constant) . . . . .	645
Torque (Controlled) . . . . .	646
Torque Sensor . . . . .	647
Torsion Spring . . . . .	648
Transfer Function . . . . .	649
Transformation 3ph->RRF . . . . .	651
Transformation 3ph->SRF . . . . .	652
Transformation RRF->3ph . . . . .	653
Transformation RRF->SRF . . . . .	654
Transformation SRF->3ph . . . . .	655
Transformation SRF->RRF . . . . .	656
Transformers (3ph, 2 Windings) . . . . .	657

---

Transformers (3ph, 3 Windings) . . . . .	660
Translational Algebraic Component . . . . .	663
Translational Backlash . . . . .	665
Translational Clutch . . . . .	666
Translational Damper . . . . .	667
Translational Friction . . . . .	668
Translational Hard Stop . . . . .	670
Translational Model Settings . . . . .	672
Translational Port . . . . .	673
Translational Reference . . . . .	674
Translational Speed (Constant) . . . . .	675
Translational Speed (Controlled) . . . . .	676
Translational Speed Sensor . . . . .	677
Translational Spring . . . . .	678
Transmission Line (3ph) . . . . .	679
Transport Delay . . . . .	686
TRIAC . . . . .	687
Triangular Wave Generator . . . . .	689
Trigonometric Function . . . . .	690
Triple Switch . . . . .	691
Trigger . . . . .	692
Turn-on Delay . . . . .	694
Variable Capacitor . . . . .	695
Variable Inductor . . . . .	698
Variable Magnetic Permeance . . . . .	702
Variable Resistor . . . . .	704
Variable Resistor with Constant Capacitor . . . . .	705
Variable Resistor with Constant Inductor . . . . .	706
Variable Resistor with Variable Capacitor . . . . .	707
Variable Resistor with Variable Inductor . . . . .	709
Voltage Source (Controlled) . . . . .	711

Voltage Source AC . . . . .	712
Voltage Source AC (3-Phase) . . . . .	713
Voltage Source DC . . . . .	714
Voltmeter . . . . .	715
White Noise . . . . .	716
Winding . . . . .	718
Wire Multiplexer . . . . .	720
Wire Selector . . . . .	721
XY Plot . . . . .	722
Zener Diode . . . . .	724
Zero Order Hold . . . . .	725
<b>15 Additional Simulink Blocks</b>	<b>727</b>
AC Sweep . . . . .	728
Discrete Analysis . . . . .	731
Impulse Response Analysis . . . . .	732
Loop Gain Analysis (AC Sweep) . . . . .	734
Loop Gain Analysis (Multitone) . . . . .	735
Modulators . . . . .	736
Multitone Analysis . . . . .	737
Steady-State Analysis . . . . .	739
Timer . . . . .	741
Transformations . . . . .	742

# Before You Begin

## Installing PLECS Blockset

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

### Installation on Microsoft Windows

- 1** If you already have a license file \*.lic, copy it to your harddisk.
- 2** 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.
- 3** After the installer has finished, it will automatically start the **PLECS Blockset Installation Wizard**.
- 4** On the **License File** page you can choose to copy the license file from your harddisk, install a license for the **PLECS Viewer** or proceed without a license file (e.g. if you are going to request a trial of student license).
- 5** Review the **MATLAB Search Path** page and click **Continue**.
- 6** Start MATLAB and enter plecslib or choose the entry PLECS in the Simulink Library Browser to open the PLECS Library.

### Installation on macOS

- 1** If you already have a license file \*.lic, copy it to your harddisk.

- 2** Open the diskimage by double-clicking it and copy the folder PLECS Blockset x.y to a location of your choice.
- 3** Run the application PLECS.app inside the folder PLECS Blockset x.y by double-clicking it. This will start the **PLECS Blockset Installation Wizard**.
- 4** On the **License File** page you can choose to copy the license file from your harddisk, install a license for the **PLECS Viewer** or proceed without a license file (e.g. if you are going to request a trial of student license).
- 5** Review the **MATLAB Search Path** page and click **Continue**.
- 6** Start MATLAB and enter plecslib or choose the entry PLECS in the Simulink Library Browser to open the PLECS Library.

## Installation on Linux

- 1** If you already have a license file \*.lic, copy it to your harddisk.
- 2** Open a terminal and expand the package plecs-blockset-x-y-z\_linux64.tar.gz by entering the command

```
tar plecs-blockset-x-y-z_linux64.tar.gz
```

in a directory of your choice.
- 3** Still within the terminal execute the program PLECS.setup inside the folder plecs/bin/glnxa64. This will start the **PLECS Blockset Installation Wizard**.
- 4** On the **License File** page you can choose to copy the license file from your harddisk, install a license for the **PLECS Viewer** or proceed without a license file (e.g. if you are going to request a trial of student license).
- 5** Review the **MATLAB Search Path** page and click **Continue**.
- 6** Start MATLAB and enter plecslib or choose the entry PLECS in the Simulink Library Browser to open the PLECS Library.

## Licensing

If PLECS cannot locate a valid license when it is loaded by MATLAB, it will open the **PLECS License Manager**, which shows the exact problem.



Choose **Install license file...** if you have a received a license file `license.lic` that you want to install or if you want to install the PLECS Viewer license that is bundled with PLECS.

Choose **Request trial license...** to request a time-limited trial or student license.

Without a valid license 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** PLECS 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.

---

## Configuring the MATLAB Search Path

The recommended method to register PLECS Blockset with MATLAB is to add appropriate `addpath` commands to the startup file `startup.m` in your MATLAB startup folder. For information on the `startup.m` file, enter `doc startup` in MATLAB. The **PLECS Blockset Installation Wizard** will assist you in creating or updating this file.

Using this method has the advantage that if you update MATLAB after having installed PLECS, the new MATLAB version will automatically know about PLECS. The disadvantage is that each user must setup their startup file individually.

As an alternative method you can register PLECS with a specific MATLAB installation using the MATLAB Path Browser or by directly editing the file `pathdef.m` in the directory `matlabroot/toolbox/local/`. This method may be appropriate if PLECS will be used by multiple users sharing the same computer. You need to add the PLECS directory and its subdirectory `demos` to the MATLAB search path.

## Configuring PLECS

For information about setting global configuration options for PLECS see “Configuring PLECS” (on page 41).

## Installing Different Versions of 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 PLECS Blockset

Uninstalling 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 your MATLAB startup file `startup.m`.
- 3 Quit MATLAB.
- 4 On Windows, deinstall PLECS Blockset by choosing the appropriate entry in the Windows control panel. On macOS 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 If you already have a license file \*.lic, copy it to your harddisk.
- 2 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.
- 3 Start PLECS.

### Installation on macOS

- 1 If you already have a license file \*.lic, copy it to your harddisk.
- 2 Open the disk image by double-clicking it.
- 3 Copy PLECS to the Application folder.
- 4 Start PLECS.

### Installation on Linux

- 1 If you already have a license file \*.lic, copy it to your harddisk.
- 2 Open a terminal and expand the package `plecs-standalone-x-y-z_linux64.tar.gz` by entering the command

```
tar plecs-standalone-x-y-z_linux64.tar.gz
```

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

- 3 Start PLECS by executing PLECS in the folder `plecs`.

## Licensing

If PLECS cannot locate any license file when you start it, it will show a message that it is unlicensed.

Choose **Start in demo mode** to use PLECS in a restricted demo mode that lets you build models and run simulations for a duration of 60 minutes. Saving models or data is disabled in this mode.

Choose **Open license manager...** to open the License Manager, which lets you install a license file or request a time-limited trial or student license.

If PLECS does locate license files but they do not contain a valid license (e.g. because it has expired) it will immediately open the License Manager without the option to start the demo mode.

## License File Location

Both PLECS Standalone and PLECS Blockset search for license files named \*.lic in the following directories:

### License File Search Paths

Platform	Search Paths
Windows	C:\Users\ <user>\AppData\Local\Plexim\PLECS\licenses C:\ProgramData\Plexim\PLECS\licenses</user>
macOS	~/Library/Application Support/Plexim/PLECS/licenses /Library/Application Support/Plexim/PLECS/licenses
Linux	~/.local/share/Plexim/PLECS/licenses /usr/local/share/Plexim/PLECS/licenses

The License Manager will install license files in the first directory listed for each platform because this location is usually writable by the user. However, an administrator may choose to install license files to be used for *all* users in the other directory.

If none of the search directories contains any license file \*.lic, PLECS uses the environment variables PLEXIM\_LICENSE\_FILE and LM\_LICENSE\_FILE to locate the license file.

## Network Licensing

If you purchase one or more floating licenses for PLECS, the license server program FlexNet Publisher is employed to control access to PLECS. FlexNet Publisher is a product of Flexera Software. The license file 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 floating licenses you have purchased.

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

```
SERVER licenseserver ANY
USE_SERVER
```

where *licenseserver* is the IP address or the hostname or fully qualified domain name (FQDN) of the server computer running the license manager. If the hostname or FQDN is used, verify that the client computer can resolve it to the correct IP address. If the license manager uses a TCP port other than 27000-27009, the port number must be specified on the SERVER line after the keyword ANY, e.g.:

```
SERVER licenseserver ANY 3456
USE_SERVER
```

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 – e.g. because the server is down or unreachable or because the licensed number of concurrent users is already reached – PLECS will open the License Manager to report the problem. In order to retry to obtain a license you need to restart PLECS Standalone or clear PLECS Blockset from the MATLAB memory using the MATLAB command `plecsclear`. Once granted, a license is returned to the server when quit PLECS Standalone or clear PLECS Blockset from the MATLAB memory.

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.

## What's New in Version 4.3

### Major New Features

- The new **Import Wizard** of the thermal editor facilitates the import of data from graphs. The wizard also includes a fitting algorithm to calculate Foster network coefficients for the thermal impedance from heating curves. See “Importing Data from Graphical Datasheets” (on page 136).
- The Target Support Package framework of the PLECS Coder has been extended to facilitate code generation and external mode operation for embedded targets (to be released separately).

### Further Enhancements

- The Monoflop (see page 491) and Turn-on Delay (see page 694) now have an option to specify the pulse duration resp. delay time via a control signal instead of a parameter.
- The dialog callback of a masked subsystem now lets you hide, show and move subsystem terminals depending on mask parameters. See “Mask Dialog” (on page 66).
- The schematic editor now has two additional zoom modes: For smooth zooming hold the **Ctrl** key (**cmd** on macOS) while turning the scroll wheel on the mouse. To fit the entire schematic into the editor window, select **Zoom to fit** from the **View** menu or press **Ctrl-\*** key (**cmd-\*** on macOS).

# 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 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).

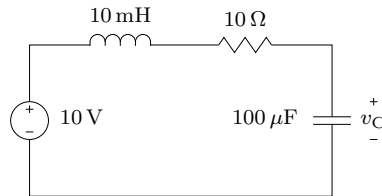
## Getting Started with PLECS Blockset

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

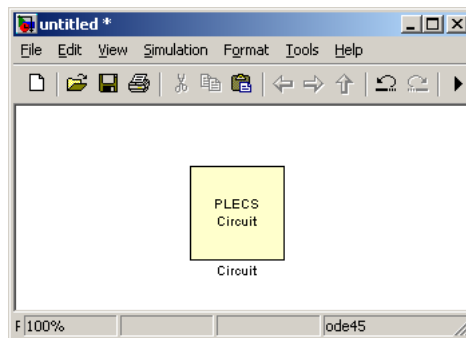
The only way to become familiar with a new program is by using it. For this reason we are presenting here two example circuits that you can reconstruct on your computer. The examples are based on each other, since the features of PLECS will be explained step by step.

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.



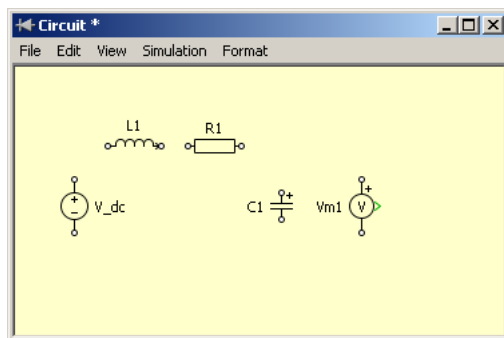
**Figure 1.2: Simulink model**

## 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.

---

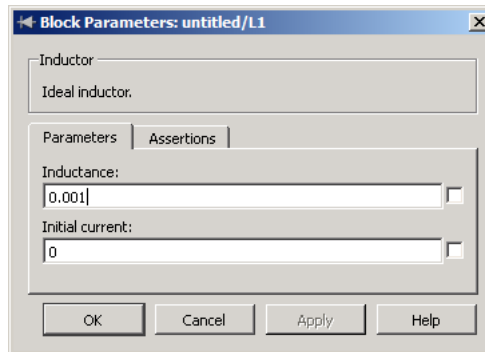
## Connections

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 a 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.

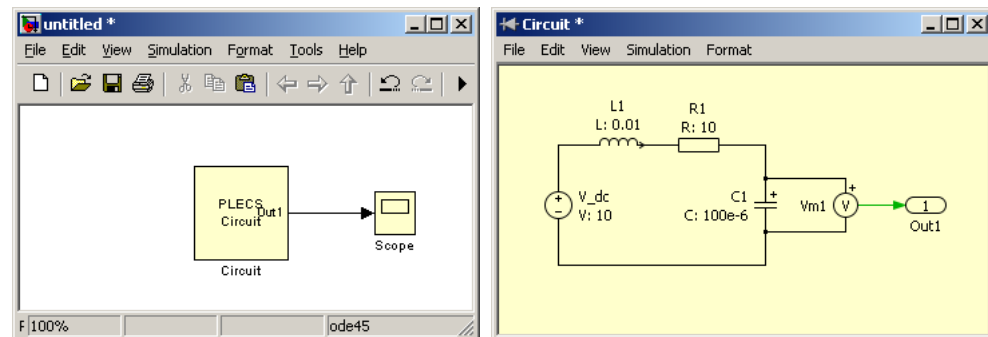
For every component enter the values according to the schematic in Fig. 1.1. In the dialog boxes of the inductor and the capacitor you can additionally set the initial current resp. the initial voltage. Please leave both values at zero.

## 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 Output” 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.

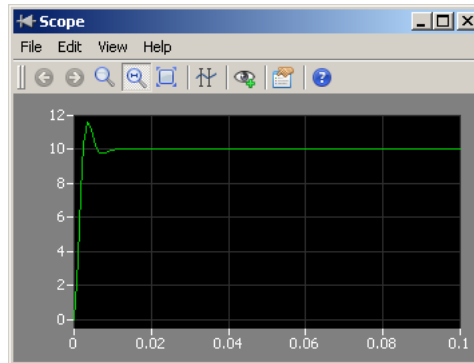
Connect the output of the voltmeter with the input of the port block. In Simulink, connect a Scope to the output of the PLECS block and start the simulation. In order to see something of the more interesting part of the simulation you probably need to set the stop time to 0.1. By this time you should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

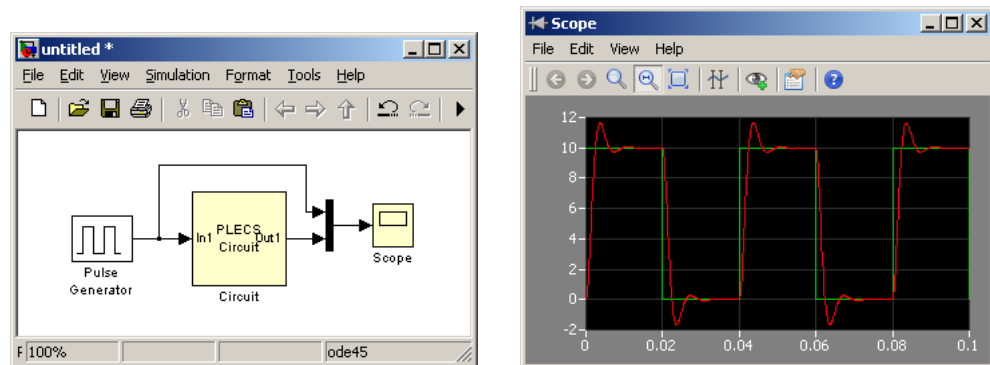
You can also add scopes in the PLECS schematic directly. The “Scope” block can be found in the library “System”.

## Importing Signals

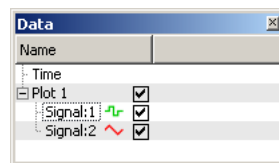
You have already learned how to export signals from the electrical circuit to Simulink via the output block. In the same manner you can also import signals from Simulink into your circuit, usually to control sources.

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 Inport” 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).



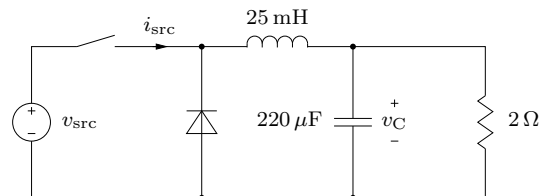
**Figure 1.7: RLC network with a pulsed voltage source**



**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.



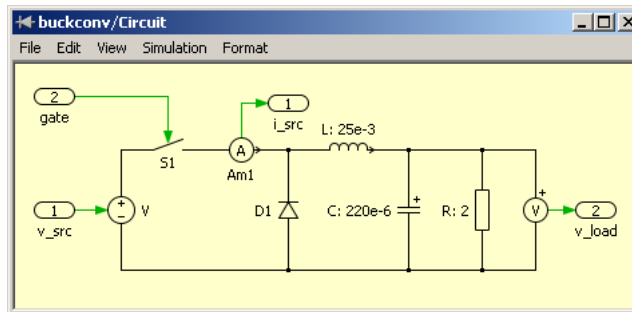
**Figure 1.9: Schematic of buck converter**

## 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.

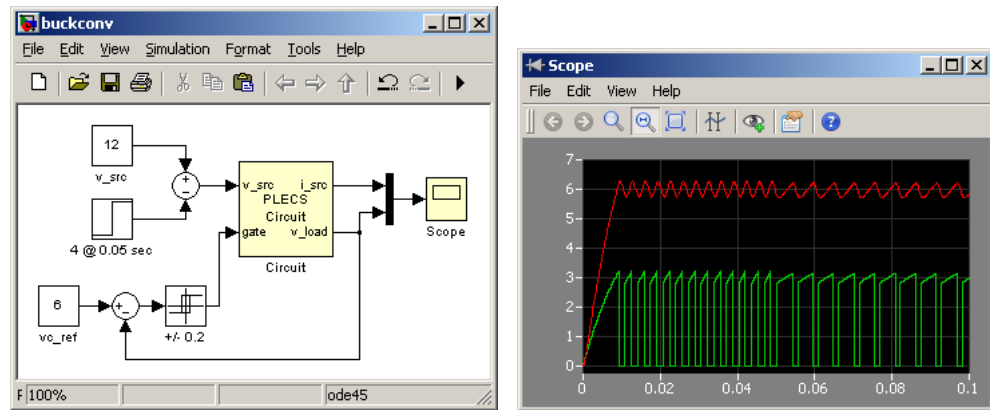
The diode can be found in the library “Electrical / Power Semiconductors” and the switch in the library “Electrical / Switches”. All components in these libraries are based on ideal switches that have zero on-resistance and infinite off-resistance. They open and close instantaneously. In some components 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  $\pm 0.2\text{V}$  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**

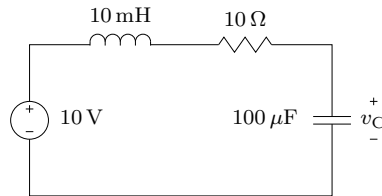
## Getting Started with PLECS Standalone

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

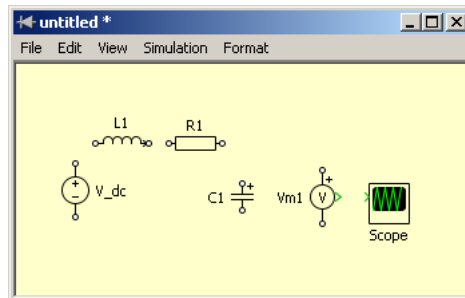
The components required for our circuit must be copied into this window from the Library Browser. This is done by dragging them with the mouse. If you want to copy components already placed in the window hold down the **Ctrl** control key or use the right mouse button.

The electrical components that you need for the RLC network can be found in 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

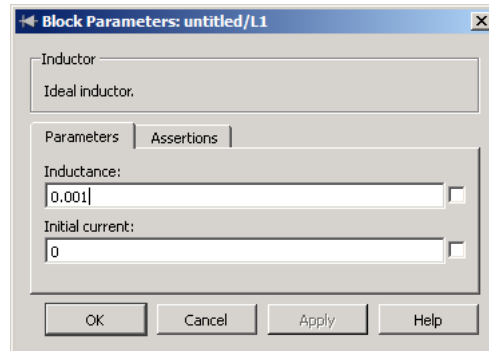
The unconnected electrical terminals of a component are marked with little hollow circles. If we bring the mouse pointer close to such a terminal the pointer shape changes from an arrow to a cross. We now can drag a connection to another component by holding the left mouse button down. When we approach 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 a 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

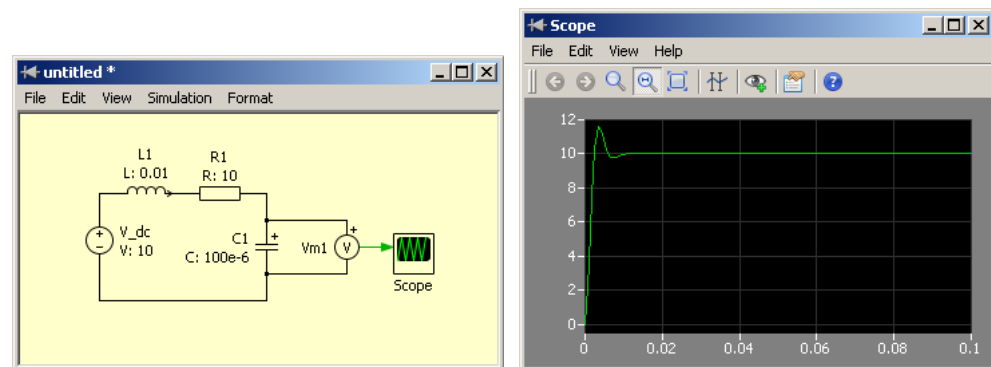
In addition to the electrical connections (wires) that are used to connect electrical components PLECS also makes use of unidirectional signals. The signals are painted in green and have an arrowhead to indicate their direction.

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**.

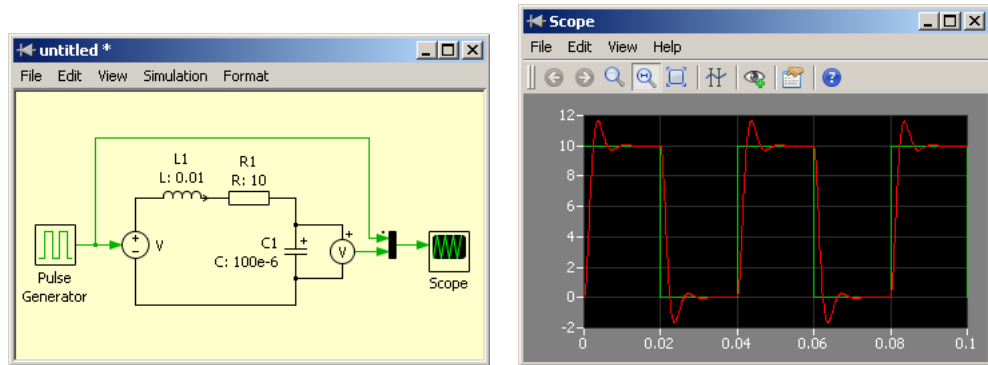
You should now get the simulation results shown in below.



**Figure 1.15: Complete model and simulation result**

## Adding Control Blocks

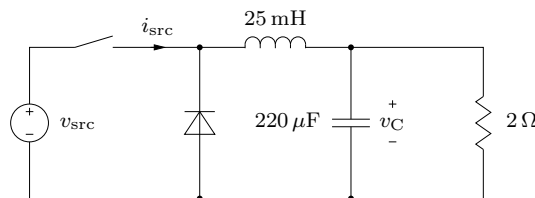
To enhance our model we would like to add some dynamic behavior into our static electrical model. Let us see how the capacitor in our example charges and discharges if we apply a pulsed voltage. In the schematic we replace the DC voltage source by a controlled one. The input of the voltage source can 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

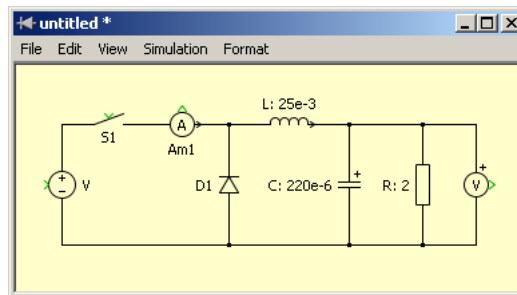
In the buck converter outlined in Fig. 1.17 we will model the transistor as an entirely controllable switch and bear in mind that it may conduct current only in one direction. We also need a free-wheeling diode. The diode is a switch that closes as the voltage across it becomes positive, and opens as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

braries 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.

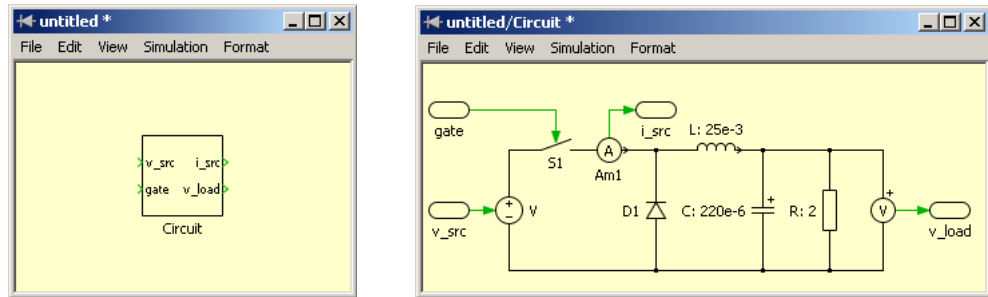


**Figure 1.18: Electrical part of buck converter**

## 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.

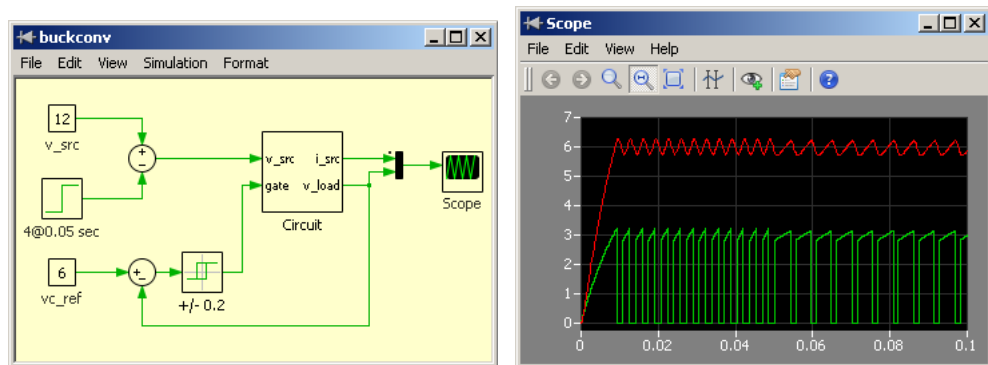
To connect the subsystem to the outer schematic we need to place ports into it. Drag two Signal Inports and two Signal Outports into the subsystem schematic and connect them to the voltage source, the switch, the volt- and



**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  $\pm 0.2\text{V}$  band around  $6\text{V}$ . To make things a bit more interesting we apply a step change from  $12\text{V}$  down to  $8\text{V}$  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.

# 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*.

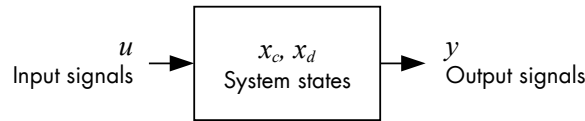
The term *modeling* refers to the process of extracting knowledge from the system to be simulated and representing this knowledge in some formal way. The second part – i.e. the representation of knowledge – can be more or less straightforward depending on the formalism used. PLECS offers three different 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 internal states.
- If the system has continuous states, a derivative function describes their derivatives with respect to time.

Symbolically, these functions can be expressed as follows:

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

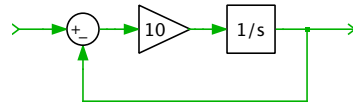
Such a description is most convenient for implementation in a procedural programming language like C.

### 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, magnetic, mechanical and thermal domains (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

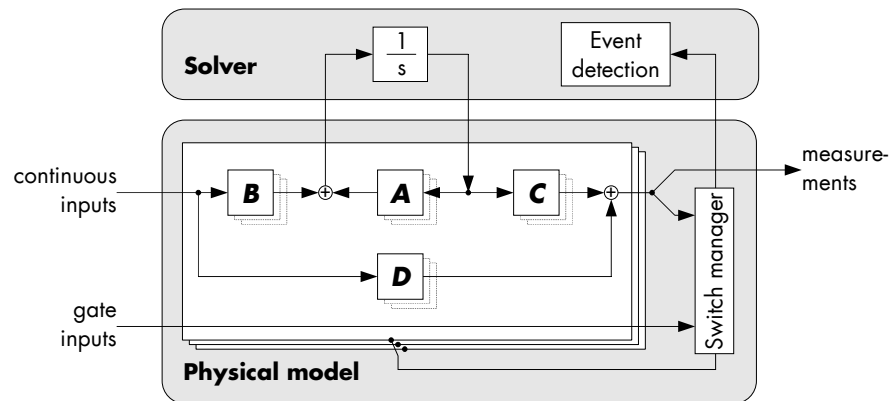
PLECS first sets up the system equations for the physical model according to e.g. Kirchhoff's current and voltage laws. If the physical model contains only ideal linear and/or switching elements it can be described by a set of piecewise linear state-space equations:

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

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

The subscript  $\sigma$  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.

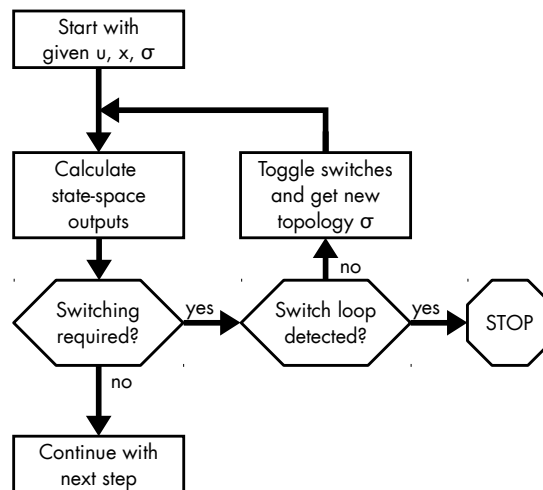


### Switched state-space implementation

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. 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.

A flowchart of the Switch Manager is shown in the figure below. In every simulation step, after the physical measurements have been calculated, the Switch Manager evaluates the switching conditions of all switches in the physical model. If a switching action is necessary, it initiates the calculation of a new set of state-space matrices or fetches a previously calculated set from a cache. Afterwards, it recalculates the physical measurements with the new state-space matrices to check whether further switching actions of naturally commutated devices are required. It will iterate through this process until all switches have reached a stable position. If a set of switch states  $\sigma$  is encountered repeatedly in this process, PLECS is unable to determine stable conditions and aborts the simulation.



**Switch Manager flowchart**

## 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.

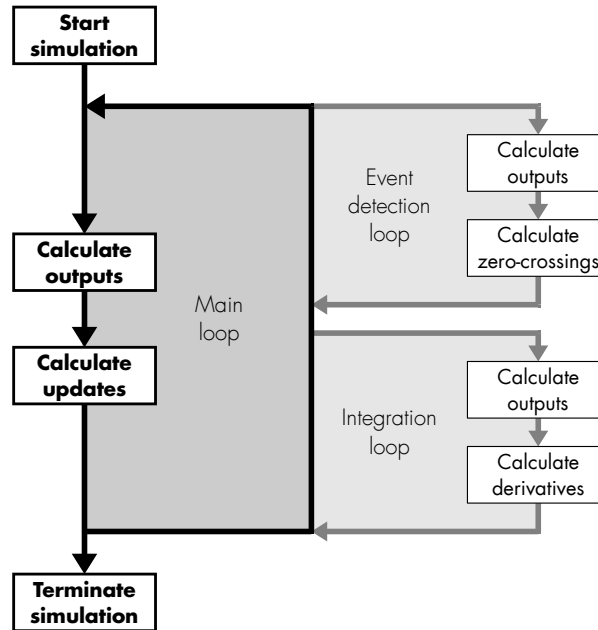
For such a group it is impossible to find a sequence in which to compute their output functions because each computation involves an unknown variable (the output of the previous block). Instead, the output functions of these blocks must be *solved simultaneously*. PLECS uses a Newton-type equation solver for this purpose. Since the solver performs iterations in order to find a solution consistent with all blocks, models with algebraic loops may run more slowly than models without algebraic loops. Failure to find a solution brings the simulation to a halt with an error message.

See “Simulation Parameters” (on page 103) for a list of parameters that influence the solution of algebraic loops.

The Initial Condition block (see page 463) can be used to provide a guess to the equation solver at the start of a simulation.

## Model Execution

The figure below illustrates the workflow of the actual simulation.



### Simulation loop

#### 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, a variable-step 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 a variable-step 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, the solver will reduce the current step size so that the next major time step is taken *just after* the event.

## Fixed-Step Simulation

As indicated in the previous paragraphs, certain important aspects of the minor simulation loops require a variable-step solver that can change its step size during a simulation. Using a solver with a fixed step size has two serious implications.

**Integration Error** A fixed-step solver does not have any control over the integration error. The integration error is a function of the model time constants, the step size and the integration method. The first parameter is obviously given by the model, but the second and possibly the third parameter must be provided by the user. One strategy for determining an appropriate step size is to iteratively run simulations and reduce the step size until the simulation results stabilize.

**Event Handling** Discontinuities in a physical model – such as the turn-on or turn-off of a diode or the transition from static to dynamic friction – typically do not coincide with a fixed simulation step. Postponing such non-sampled events until the following fixed simulation step will produce jitter and may lead to subsequent runtime errors, e.g. because a physical state variable becomes discontinuous.

For these reasons it is generally recommended to use a variable-step solver. However, if you plan to generate code from a model to run it on a real-time system you will need to use a fixed-step solver.

## Physical Model Discretization

In order to mitigate the problems due to non-sampled events, PLECS transforms the physical model into a discrete state-space model when it is simulated with a fixed-step solver. The continuous state-space equations of the electrical and magnetic domains are discretized and replaced with the following update rule:

$$\mathbf{x}_n = \mathbf{A}_d \cdot \mathbf{x}_{n-1} + \mathbf{B}_{d1} \cdot \mathbf{u}_{n-1} + \mathbf{B}_{d2} \cdot \mathbf{u}_n$$

By default, a first-order hold is applied to the input signals, i.e. it is assumed that the inputs change linearly from  $\mathbf{u}_{n-1}$  in the previous step to  $\mathbf{u}_n$  in the current step. As a consequence, the inputs of the electro-magnetic model now have direct feedthrough because their current values must be known before the current model states and the model output can be calculated. This will result in an algebraic loop if the value of a controlled voltage or current source depends on a measurement in the electro-magnetic model.

To avoid this problem, the Controlled Current Source (see page 352) and the Controlled Voltage Source (see page 711) can be configured to apply a zero-order hold on the input signal when the model is discretized. In this case only the input value from the previous simulation step  $u_{n-1}^{(i)}$  is required to calculate the current state values.

By default, the discrete state-space matrices  $A_d$ ,  $B_{d1}$  and  $B_{d2}$  are calculated from the continuous matrices  $A$  and  $B$  using a proprietary 4th order accurate, stiffly stable back-interpolation algorithm called BI45. Alternatively, the bilinear transformation known as Tustin's method can be chosen. It is only 2nd order accurate and has poor damping characteristics for time constants that are smaller than the discretization step size, but it is cheaper to compute because  $B_{d1}$  equals  $B_{d2}$ . Therefore, it can be useful for real-time simulations where the calculation time is essential. The discretization method can be chosen in the Simulation Parameters dialog (see page 103).

Note that this applies only to the discretization of the electro-magnetic domain. The state variables both from other physical domains and from the control block diagram are integrated with Euler's method.

### Interpolation of Non-Sampled Switching Events

With the physical model discretized like this, non-sampled switching events can be handled efficiently using the following algorithm:

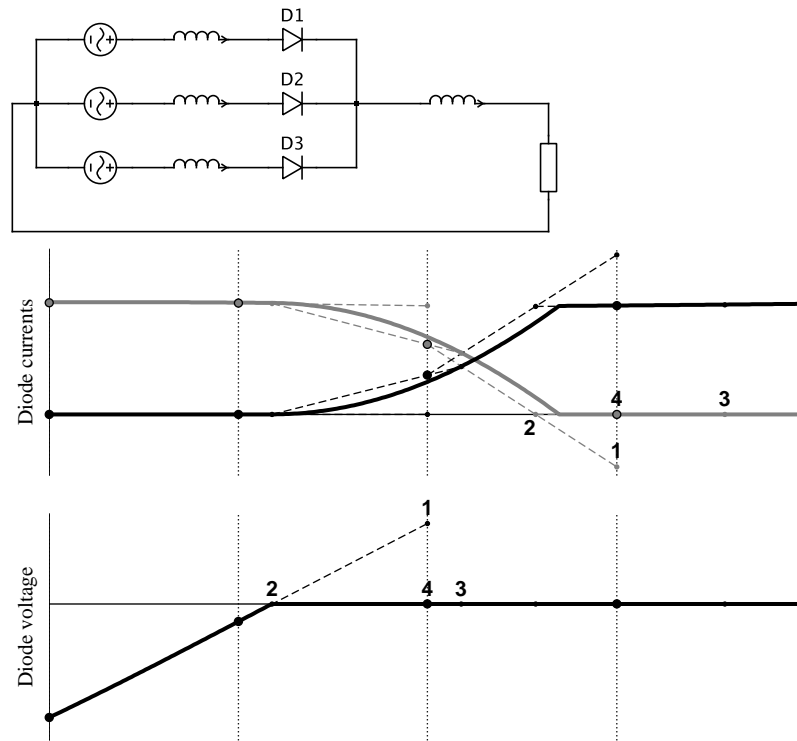
- 1** Check whether the solver has stepped over a non-sampled switching event in the last simulation step.
- 2** If so, determine the time of the event and calculate the model state *just after* the event using linear interpolation and handle the event, i.e. toggle one or more switches.
- 3** Perform *one full* forward step.
- 4** Linearly interpolate the model states back to the actual simulation time.

This algorithm is illustrated using the example of a half-wave rectifier shown below. The two graphs show the commutation of the dc current from diode D3 (shown in gray) to diode D1 (shown in black). The solid lines show the results from a simulation with a variable-step solver, large dots mark the steps of the fixed-step simulation, and small dots mark the internal interpolation steps.

Commutation starts when the voltage across D1 becomes positive. The fixed-step solver first steps well beyond the zero-crossing of the voltage (1). PLECS then internally steps back to the zero-crossing (2) and turns on D1. With the



new set of state-space equations, it performs an internal full step forward (3) and then interpolates back to the actual simulation time (4). Next, the solver steps beyond the zero-crossing of the current through D3 (1). Again, PLECS internally steps back to the zero-crossing (2) and turns off D3. With the new set of state-space equations, it performs an internal full step forward (3) and then interpolates back to the actual simulation time (4).



### Interpolation of non-sampled switching events

Note that *without* this interpolation scheme, D3 would have been turned off at point (1). This would have caused the current through the inductor in phase 3 to become *discontinuous*. Such a non-physical behavior can lead to gross simulation errors and should therefore be avoided.

## 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.

**Inherited** An *inherited sample time* is used for blocks that do not have a sample time of their own but may adopt the sample time from other blocks connected to them. This includes blocks such as Gain, Sum and Product.

**Constant** A *constant sample time* is used for blocks that are updated only once at the beginning of a simulation. The only block that explicitly uses a constant sample time is the Constant block. However, other blocks may *inherit* a constant sample time.

For most block types the sample time is automatically assigned. Discrete blocks and the C-Script block (see page 337) 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 table below 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, $T_p > 0$ $T_p$ $T_o$ : Sample offset, $0 \leq T_o < T_p$
Discrete-Variable	[-2, 0] -2
Inherited	[-1, 0] -1
Constant	[inf, 0] inf

### Sample Time Inheritance

For blocks with an inherited sample time, PLECS employs the following propagation scheme to determine an appropriate sample time:

- 1** Propagate the sample times *forward* along the block execution order (see “Block Sorting” on page 29). A block with an inherited sample time will be assigned a sample time based on the sample times of the blocks that are connected to the block’s *inputs*.
- 2** Propagate the sample times *backward* along the block execution order. A block with an inherited sample time will be assigned a sample time based on the sample times of the blocks that are connected to the block’s *outputs*.
- 3** Loop until there are no inherited sample times left or until no inherited sample time can be resolved.

Sample times are assigned according to the following rules:

- If any sample time is inherited, the block sample time also remains inherited.

- Else, if *all* sample times are constant, the block sample time is set to constant.
- Else, if *any* sample time is continuous, the block sample time is set to continuous.
- Else, if *all* sample times are fixed-step discrete or constant and the fastest sample time is a valid base sample time of the other non-constant sample times, the block sample time is set to the fastest sample time.
- Else, the block sample time is set to semi-continuous.

Any block sample time that cannot be resolved using this propagation scheme is set to continuous.

### 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  $3 \times 10^{-4}$  is not equal to  $3e-4$  (even though the difference is only about  $5.4 \times 10^{-20}$ ).

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

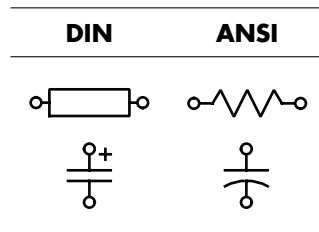
The language used by PLECS can be specified in the **Language** field. PLECS uses the language settings of your computer as default setting. Available languages are *English* and *Japanese*. To activate the new language settings PLECS must be restarted.

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.

When the **Circuit browser default** setting specifies the default filtering mode used for all circuit browsers (see section “Circuit Browser” on page 79).

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 247) 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.

---

**Note** To create a new component library in PLECS Blockset, you need to copy the PLECS Library block from the PLECS Extras library into a Simulink model or library. For details on creating custom libraries see also section “Libraries” (on page 53).

---



## Thermal

The setting **Thermal description search path** contains the root directories of the thermal library. See section “Thermal Library” (on page 126) 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 96). 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.

## Update

PLECS can be configured to check for updates every time it is started. If the computer running PLECS is located behind a firewall it may be necessary to configure proxy settings. These settings can be determined automatically or entered manually.

The **Server name** configures the fully qualified domain name or the IP address of the HTTP proxy server. Leave empty to disable proxy usage.

The **Server port** configures the TCP port of the HTTP proxy server.

The **Proxy user** configures the username to use for proxy authentication. Leave empty to disable proxy authentication.

The **Proxy password** configures the password for proxy authentication.

---

**Note** To check for updates PLECS sends its version to the PLEXIM update server. No further personal information is transmitted.

---

### Coder

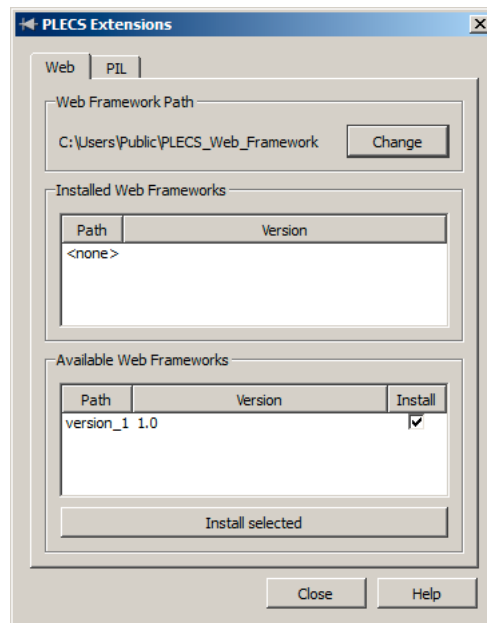
This tab only appears if you have a license for the PLECS Coder. For more information about the PLECS Coder, see “Code Generation” (on page 257).

The setting **Target support packages path** lets you specify a folder, which PLECS searches for target support packages for the PLECS Coder. A target support package enables the PLECS Coder to generate code that is specific for a particular target such as the PLECS RT Box.

The table **Installed targets** lists the support packages that are installed in the specified folder. PLECS automatically searches the folder when you start it. To refresh the list after you have installed a new target, click on the **Refresh** button.

## Installing Extensions

The functionality of PLECS can be extended using packages called *extensions*. For PLECS 4.2, two extensions are available: **WBS** (Web-Based Simulation) and **PIL** (Processor-In-the-Loop). To install and configure these extensions, the Extensions dialog can be used: in PLECS, click on the menu entry **PLECS Extensions...** from the **File** menu.



**WBS** allows the visitor of a web page to run PLECS simulations interactively in a web browser. The simulation models are provided by a PLECS simulation server. PLECS Standalone can be used as a simulation server on the local computer. No additional WBS license is needed as long as this feature is used only for development and test purposes; for the deployment of the web service a separate server license is required. The PLECS WBS framework contains the necessary files to set up the web service. Please see the accompanying documentation within the framework.

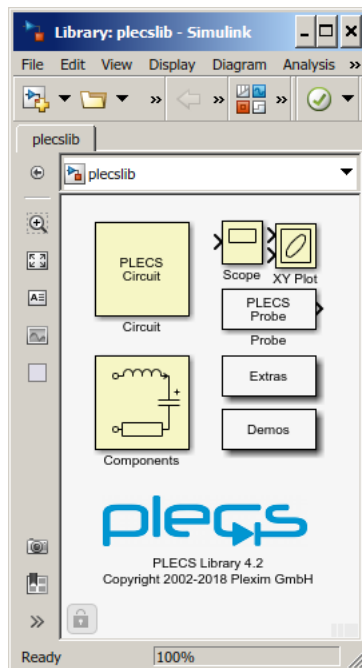
The **PIL** approach allows to run code on an embedded controller that is synchronized with the simulation in PLECS. The necessary framework files for generating suitable embedded applications can be installed by users that have a separate PIL license. For more information about **PIL**, see chapter

“Processor-in-the-loop” (on page 275) and the separate documentation available from the PLECS website.

The dialog shows the available frameworks shipped with the current PLECS release in the lower part and the already installed frameworks in the upper part. To extract a framework an installation path has to be specified once (at the top of the dialog). Select the desired version from the available frameworks and click on **Install selected**.

## Creating a New Circuit with 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 50).

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('s1',202); open_system(gcf,'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.

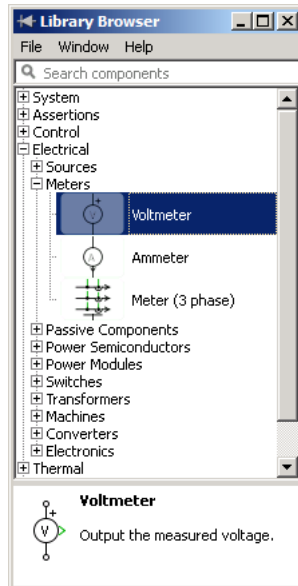
- 4** On the **Parameters** pane of the mask editor add a checkbox parameter with the prompt `Open schematic` and the variable name `openschematic`. As a dialog callback for the new parameter enter

```
if (strcmp(get_param(gcb,'openschematic'),'on'))
    set_param(gcb,'openschematic','off');
    plecs('s1',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.

---

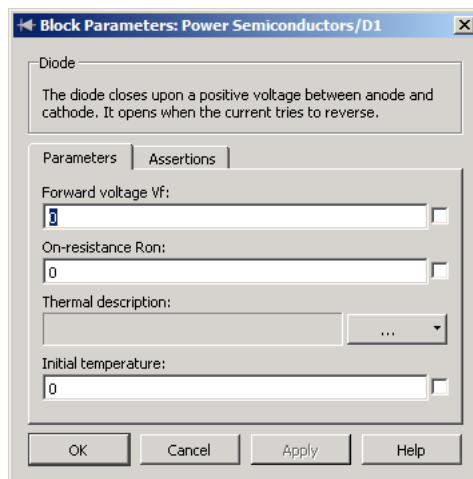
**Note** In PLECS Blockset you cannot place Simulink blocks in a PLECS schematic or PLECS components in a Simulink model since both programs do not share the same Graphical User Interface.

---

## Components

### Specifying Component Parameters

Every component has a dialog box to view and modify the component parameters. To open the parameter dialog, double-click on a component or select the component and choose **Parameters...** from the **Edit** menu or the component's context menu.



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 501) or the parameter **Width** of the Wire Multiplexer (see page 720). Such parameters must be literal values and are evaluated immediately.

Integer values may be specified as decimal, binary or hexadecimal values. Hexadecimal integers are prefixed by 0x, binary integers are prefixed by 0b. The expressions 0xfa and 0b11111010, for example, both evaluate to 250.



## 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 66).

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 47).

## 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 Parameters of Multiple Components

You can simultaneously change the parameters of multiple components of the same type. To do so, select the components, then double-click any of them or choose **Parameters...** from the **Edit** menu or the components’ context menu.

In the parameter dialog for multiple components, parameters, for which the selected components have different values, show a placeholder text `multiple values`. If you leave this placeholder as is, the components retain their individual values for this parameter when you apply any other changes that you have made.

## Changing Parameters During a Simulation

Parameters are evaluated once a new simulation is started. Their values remain constant throughout the simulation. Certain parameters can be changed during the simulation, their value is used as soon as the change is applied. Depending on the parameter type it may be necessary to reevaluate other parts of the model, which may take some extra computation time.

Parameters are changeable during the simulation if they do not change the structure of the model. If, for example, a parameter value is a vector the elements of the vector may be changed, whereas the size of the vector must remain the same. Parameters that influence the number of terminals of a component or the width of a signal cannot be changed during simulation.

### Changing Component Names

Component names are edited by double-clicking them in the schematic. To show or hide a component name, toggle **Show name** in the **Format** menu.

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 41 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 41 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 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 resolved upon loading of a circuit. Afterwards, any changes that you make to a referenced library component are automatically propagated to the referencing components when you start a simulation or (in PLECS Blockset) when you update the simulation model.

### 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 **Sub-system** submenu of the **Edit** menu or the component's context menu.

It is often desirable to break the links to all user-defined libraries in a model, for example when sending a model to another PLECS user who does not have all the libraries that the model depends on. This can be done by selecting **Break all library links...** from the **Edit** menu. Library links to the PLECS component library are not affected by this functionality.

## Connections

Connections define the relationship and interaction between components.

### Signal Connections and Physical Connections

Signals are drawn with green lines ending with an arrow head. They represent a directed flow of values from the signal output of one component to the signal 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.

Physical connections represent energy flow between two points and do not have an inherent direction. They are drawn in separate colors for the different physical domains: black for electrical, red for magnetic, blue for thermal and violet for mechanical. Physical connections can be created between physical terminals of the same domain.

### Creating Connections

To create a new connection, move the mouse pointer over an unconnected terminal, press the mouse button, and drag the mouse pointer to the desired destination. If you drag the mouse pointer near a matching terminal, the pointer shape changes to a double cross, and the two terminals will be connected when you release the mouse button. If you drag the mouse pointer over a component, the connection will be routed to the nearest matching terminal (if any) of that component.

### Creating Branches

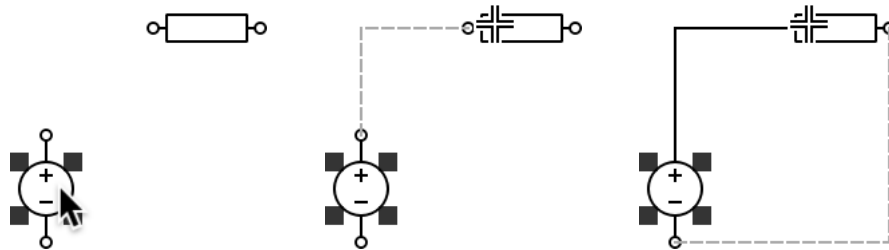
Branches are used to connect more than two terminals. To create a branch connection, place the mouse pointer on an existing connection or node where you want the branch to start. Press the right mouse button and drag the mouse pointer to the desired destination. Instead of the right mouse button you can also use the left mouse button while holding down the **Ctrl** key.

Alternatively, you can also create a branch by clicking on an unconnected terminal and dragging the mouse pointer to a matching connection or node.

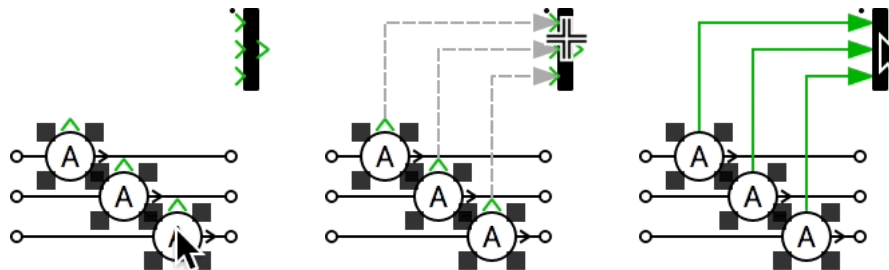
## Automatic Creation of Connections

If you select a component and hold the **Alt** key while hovering the mouse pointer over another component, the schematic editor suggests connections between matching terminals or terminal groups of the two components. A terminal group in this context is a contiguous set of terminals of the same kind along one edge of a component. To create the connection(s), press the mouse button. Only one pair of matching terminals or terminal groups is connected at one time. If there are multiple candidates, the connection with the shortest path is chosen.

This is illustrated in the figures below. First, the starting component, a voltage source, is selected. Next, the mouse pointer is moved to the destination component, a resistor, while holding down the **Alt** key. The editor suggests a connection between the closest two electrical terminals. Last, after a mouse click, the connection is created, and the editor suggests another connection between the remaining two terminals.



You can also let the schematic editor create connections from multiple starting components to a single destination component at once. This is useful e.g. to combine the signal outputs of multiple meters into one Signal Multiplexer (see page 582). First, select the many components, then move the mouse pointer to the destination while holding down the **Alt** key. Press the mouse button to create all connections at once.



## Editing Connections

After a connection has been created, you can change its path by moving individual segments. To move a connection segment, click it with the left mouse button, then drag it to the desired destination.

You can also move parallel segments of different connections simultaneously. To do so, select the connections, then click on any one of the parallel segments and drag it to the desired destination. The other segments will be shifted simultaneously while maintaining their relative distances.

## Annotations

You can annotate schematics with text labels. Create an annotation by double-clicking in an unoccupied area of a schematic and start typing. To finish editing, press the **Escape** key or click anywhere outside the annotation box.

You can move an annotation by selecting and dragging it with the mouse. To edit an existing annotation, double-click it. 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 61).

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 port blocks (e.g. Signal Inport (see page 580), Signal Outputport (see page 583) or Electrical Port (see page 390)) 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.

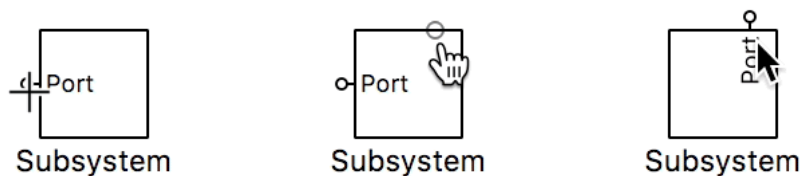


## Arranging Subsystem Terminals

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.

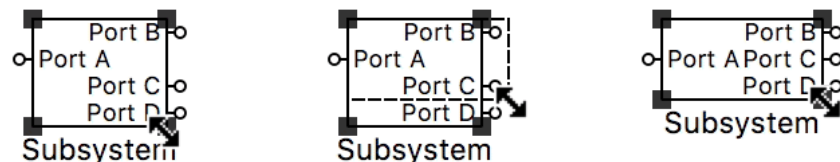


Notice how the shape of the cursor changes to crosshairs as you move it into the capture radius of the terminal. When you press and hold down the center mouse button, the cursor shape changes to a pointing hand.

## 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. 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.

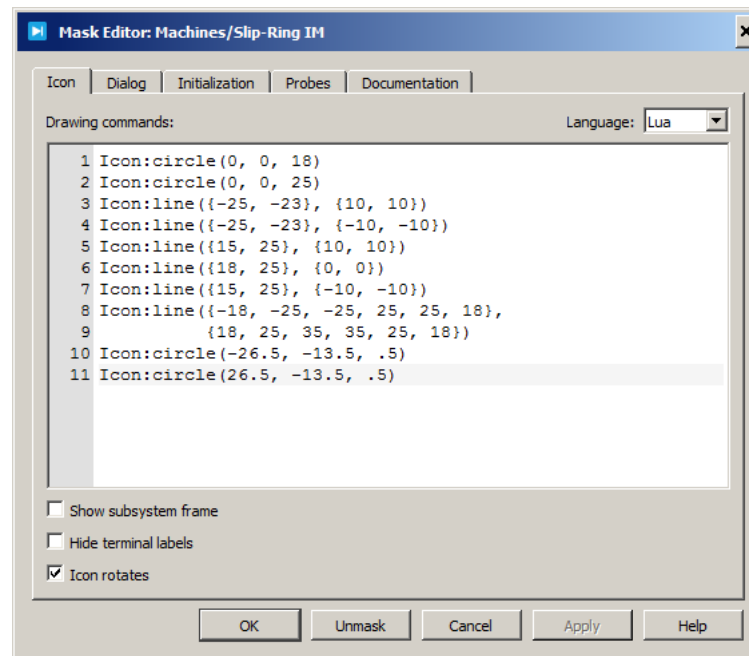
## Masking Subsystems

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 **Create mask...** from the **Subsystem** submenu of the **Edit** menu or the block's context menu. The mask editor appears. The mask editor consists of five 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 Editor Icon Tab

## Mask Icon Drawing Language

The **Language** selector lets you choose the programming language used for the drawing commands. Choose **Lua** to create dynamic icons that can change depending on user input. See “Getting Started with Lua” (on page 74) for a brief introduction to this language. Choose **Legacy** to select the syntax used by PLECS 4.1 and older.

## Mask Icon Drawing Commands

The drawing commands available in the Lua language are described below. If you enter more than one command, the graphic objects are drawn in the order in which the commands appear. In case an error occurs during evaluation of the commands, PLECS displays three question marks (? ? ?) in the mask icon; if you hover the mouse over the subsystem block, a tooltip will show the error message.

### Text

The commands

```
Icon:text('text')  
Icon:text(x, y, 'text')
```

display *text* in the center of the icon or centered around the coordinates *x*, *y*. The text does not rotate with the icon; it is always displayed from left to right. The second command can be followed by parameter/value pairs to specify additional properties listed in the table below.

### Line

The command

```
Icon:line(xvec, yvec)
```

draws the line specified by the vectors *xvec* and *yvec*. Both vectors must have the same length. Note that vectors are entered using curly braces, e.g. {1, 2, 3}. The vectors may contain non-finite values such as 0/0 or 1/0. When non-finite values are encountered, the line is interrupted and continued at the next point that has finite values.

**Text Properties**

Property	Description
FontSize	An integer specifying the font size of the text
TextFormat	Specify the string PlainText to display the text as is (the default) or RichText to enable HTML markup such as <b></b> or <sup></sup>.
Color	A vector { <i>r</i> , <i>g</i> , <i>b</i> } of three integers in the range from 0 to 255 specifying the text color in RBG format

**Patch**

The command

```
Icon:patch(xvec, yvec)
```

draws a solid polygon with vertices specified by the vectors *xvec* and *yvec*. Both vectors must have the same length. Note that vectors are entered using curly braces, e.g. {1, 2, 3}.

**Circle**

The command

```
Icon:circle(x, y, r)
```

draws a circle with the center coordinates *x*, *y* and the radius *r*.

**Ellipse**

The command

```
Icon:ellipse(x, y, rx, ry)
```

draws an ellipse with the center coordinates *x*, *y* and the radii *rx* and *ry*.

### **Arc**

The command

```
Icon:arc(x, y, rx, ry, start, span)
```

draws an elliptical arc with the center coordinates *x*, *y* and the radii *rx* and *ry* beginning at the *start* angle (in degrees) and extending *span* degrees counterclockwise. The 0 degree angle is at 3 o'clock. Clockwise arcs can be drawn using a negative *span* angle.

### **Color**

The command

```
Icon: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.

### **Image**

The commands

```
Icon:image(xvec, yvec, 'filename')  
Icon:image(xvec, yvec, 'filename', 'on')
```

read an image from the file *filename* and display it on the mask icon. The parameter *filename* must be either 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). Supported image formats are BMP, GIF, JPG and PNG.

The two-element vectors *xvec* and *yvec* specify the minimum and maximum coordinates of the image's extent. Note that vectors are entered using curly braces, e.g. {-10, 10}.

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

Only one image can be displayed on a mask icon; if you use multiple image commands, only the last one will be effective.

## Querying Parameter Values

The command

```
Dialog:get('variable')
```

returns the string value of the mask parameter associated with the variable *variable*. Note that the parameter values are not evaluated. Thus, if the user enters e.g. 2\*2, the return value will be '2\*2' and not '4'. The return value for a combobox parameter is a string containing the 1-based index of the chosen option or the expression entered by the user.

## Legacy Command Syntax

The legacy command syntax is listed in the following table. The meaning of the command arguments is analogous to the Lua syntax. Vector parameters are entered using square brackets with optional commas for separating the elements, e.g. [1, 2, 3] or [3 4 5]. Note that the legacy syntax does not support all commands and options. Single-line comments start with a percent sign (%).

### Legacy Command Syntax

Command	Syntax
<b>Text</b>	text('text') text(x, y, 'text')
<b>Line</b>	line(xvec, yvec)
<b>Patch</b>	patch(xvec, yvec)
<b>Circle</b>	circle(x, y, r)
<b>Color</b>	color(r, g, b)
<b>Image</b>	image(xvec, yvec, imread('filename')) image(xvec, yvec, imread('filename'), 'on')

## 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

adjusted when the block is resized. In an unrotated and unflipped block, the x-axis stretches from the left towards the right, and the y-axis stretches from the top towards the bottom.

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.

---

**Note** PLECS expects you to confine icon drawings to the boundaries of the subsystem frame. PLECS will *not* clip your drawings to the subsystem frame, but if you draw outside the frame, the drawings will not be erased properly e.g. when the subsystem is moved.

---

### 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 un-set and the name of the corresponding port block is visible.

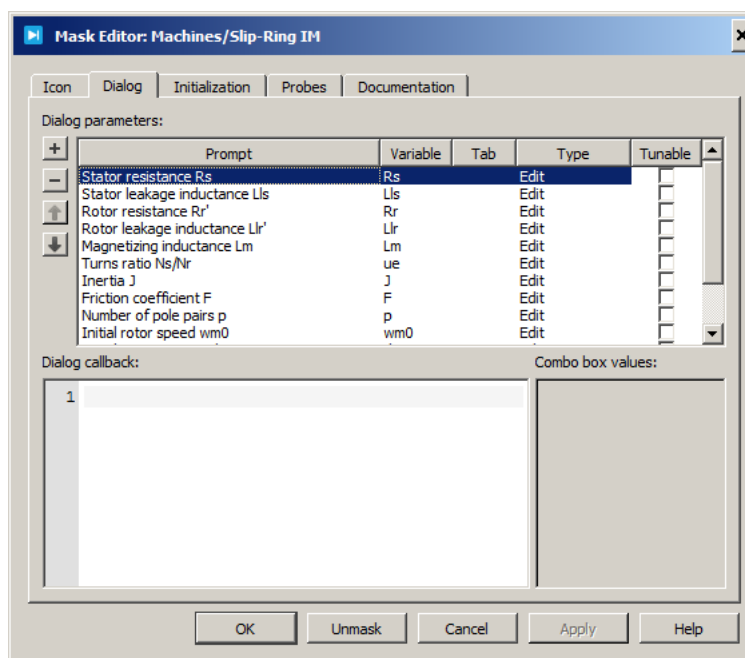
#### Icon rotates

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

### Mask Dialog

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





## Mask Editor Dialog Tab

### Prompts and Associated Variables

Mask parameters are defined by a **Prompt**, a **Variable** and a **Type**. The prompt provides information that helps the user identify the purpose of a parameter. The variable name specifies the variable that will be assigned the parameter value.

---

**Note** In PLECS Standalone, the maximum length of variable names is 63 characters. This is due to the way, in which a mask workspace is stored in PLECS and exchanged with Octave. It is advisable to observe this limit also in PLECS Blockset to ensure that a model can be exchanged with PLECS Standalone.

---

Mask parameters appear on the dialog box in the order they appear in the

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

### Parameter Types

Parameters of type **Edit** are shown as a text edit field. The entered text is interpreted as a MATLAB/Octave expression and is usually expected to evaluate to a numeric value. Parameters of type **String** are shown as a combination of a text edit field and a switch that determines whether the entered text should be interpreted as a literal string or as a variable or expression that *evaluates* to a string. 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 123) for more details.

### Tab Names

You can group parameters into separate tabs shown in the parameter dialog by assigning a tab name in the **Tab** column. All parameters that have the same tab name will appear on the same tab page in the parameter dialog.

### Tunable Parameters

By default, mask parameter values cannot be modified during a simulation. However, if you check the box in the **Tunable** column, the corresponding parameter will be made tunable so that you can in fact modify it interactively during a simulation by entering a new value in the parameter dialog. Whenever you change the parameter, PLECS will re-evaluate the parameter variable and the mask initialization commands and propagate the new variable values to the underlying components, which in turn must have tunable parameters. If a changed variable is used in a non-tunable component parameter, the change will have no effect until the simulation is restarted.

### Dialog Callback

The dialog callback is a Lua function that is executed whenever the user changes a mask parameter. You can use it to disable or hide a parameter or change its value depending on the value of another parameter.

## Querying Parameter Values

The command

```
Dialog:get('variable')
```

returns the string value of the mask parameter associated with the variable *variable*. Note that the parameter values are not evaluated. Thus, if the user enters e.g. 2\*2, the return value will be '2\*2' and not '4'. The return value for a combobox parameter is a string containing the 1-based index of the chosen option or the expression entered by the user.

## Setting Parameter Properties

The command

```
Dialog:set('variable', 'property', value, ...)
```

changes one or more properties of the mask parameter associated with the variable *variable*. The properties are listed in the following table.

### Parameter Properties

Property	Description
Value	A string specifying the parameter value
Enable	A Boolean (i.e. true or false) specifying the enable state of the parameter. A disabled parameter is greyed out in the dialog and cannot be modified.
Visible	A Boolean (i.e. true or false) specifying the visibility of the parameter in the dialog

---

**Note** Parameters that are disabled or invisible are not evaluated, and the parameter variables are assigned a NaN value (not a number).

---

## Hiding, Showing and Moving Terminals

The command

```
Block:showTerminal('name', flag)
```

shows or hides the terminal named *name* depending on the boolean value *flag*. The companion port of a hidden terminal acts in the same way as if the terminal was shown but unconnected.

The command

```
Block:moveTerminal('name', x, y)
```

moves the terminal named *name* to the *relative* coordinates *x, y* with respect to the *unrotated* and *unflipped* block. Note that the terminal rotation is not changed.

## Mask Workspace

### 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 access only variables that are defined in this mask workspace.

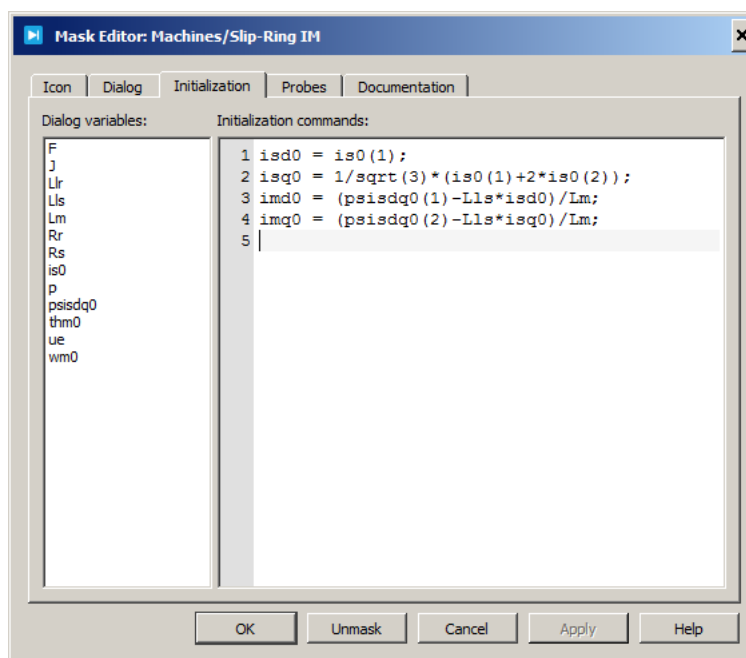
### Initialization Commands

Mask initialization commands are defined on the **Initialization** pane. They are evaluated in the mask workspace when a simulation is started. You can enter any valid MATLAB/Octave expression, consisting of MATLAB/Octave functions, operators, and variables defined in the mask workspace. Variables defined in the base workspace cannot be accessed. The dialog parameter variables are listed on the left hand side of the tab.

You can use mask initialization commands to check the user input, e.g. whether a variable value is within a certain range, or to define additional workspace variables that may be derived from mask parameters.

To show an error message in the Diagnostics window, e.g. that a certain mask parameter value is not valid, use the command

```
error('error message')
```



### Mask Editor Initialization Tab

To show a warning message in the Diagnostics window, use the command

```
plecs('warning', 'warning message')
```

Note that the native MATLAB/Octave command `warning` will only print the warning message to the MATLAB command window or the Octave console.

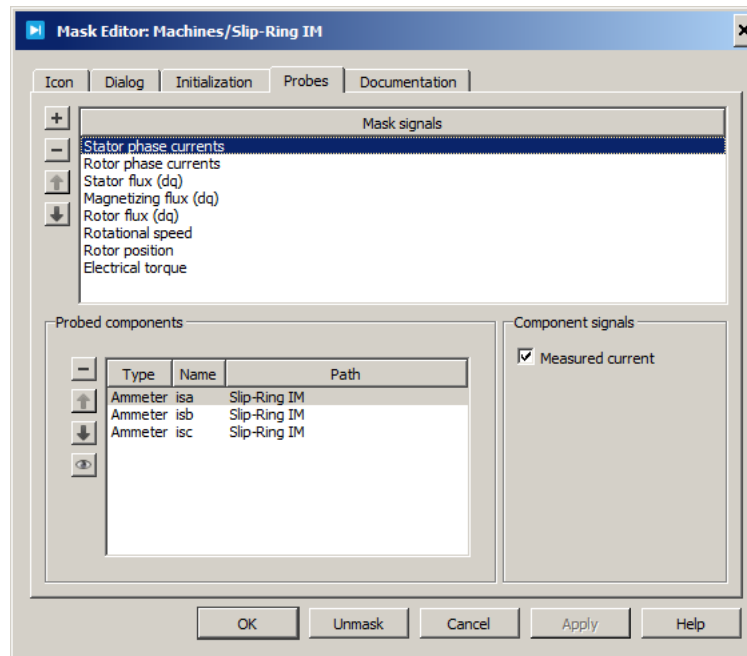
---

**Note** In PLECS Standalone, the maximum length of variable names is 63 characters. This is due to the way in which a mask workspace is stored in PLECS and exchanged with Octave. It is advisable to observe this limit also in PLECS Blockset to ensure that a model can be exchanged with PLECS Standalone.

---

## 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.

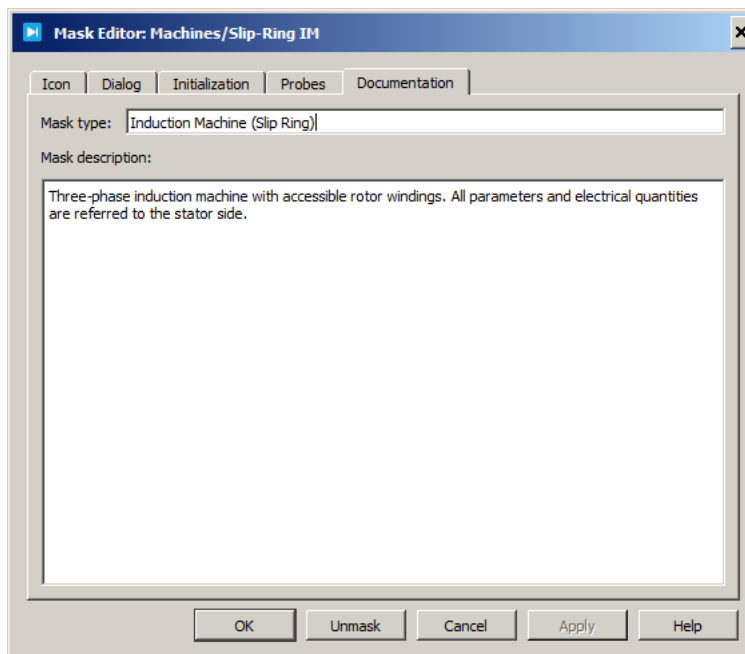


### Mask Editor Probes Tab

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 Editor Documentation Tab

#### 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 **Subsystem** submenu of the **Edit** menu or the block's context menu. You can later protect it again by choosing **Protect** from the same menus.

## Getting Started with Lua

Lua is a simple yet powerful open-source scripting language. This section introduces you to the basic concepts that you are likely to need in order to create dynamic subsystem masks. For a full reference please visit the Lua site at <http://www.lua.org>.

### Types and Variables

Lua is a dynamically typed language, which means that types are not associated with variables but only with values. To define a variable, you simply assign a value to it.

By default, Lua declares all variables as *global*. However, PLECS executes Lua code in a protected environment that forbids the creation or modification of global variables. Therefore, you must explicitly declare variables as *local* using the `local` keyword, e.g.

```
local a = "a string"
```

The most basic types in Lua are: `nil`, `boolean`, `number`, `string` and `table`. You can query the type of a value with the function `type` which returns the type name as a string.

**Nil** Nil is a type with the single value `nil`. It is used to represent the absence of a useful value.



**Booleans** The Boolean type has two values `false` and `true`. It can be used in conditional expressions. If you use other types in conditional expressions, beware that Lua considers only the Boolean `false` and `nil` as false and anything else as true. In particular, both the numerical 0 and the empty string `""` are considered true in conditional tests.

Lua supports the logical operators `and`, `or` and `not`.

**Numbers** Lua differentiates between (double-precision) floating point numbers and (64-bit) integer numbers. Numerals with a decimal point or an exponent are considered floats; otherwise, they are treated as integers.

**Strings** String literals in Lua can be delimited by single or double matching quotes:

```
local a = "a string"
local b = 'another string'
```

The difference between the two kinds is that inside one kind of quotes you can use the other kind of quote without needing to escape it. The escape character is the backslash (`\`), and the common C escape sequences such as `\n` for new-line are supported.

Long strings literals are delimited with matching double square brackets that enclose zero or more equal signs, e.g. `[...] ]` or `[==[...]==]`. They can span several lines and do not interpret escape sequences.

**Tables** Tables are Lua's generic data structuring mechanism. They are used to represent e.g. arrays, sets or records. A table is essentially an associative array that accepts as keys not only numbers or strings but any other value except `nil`. A table is constructed with curly braces and a sequence of key/value pairs, e.g.

```
a = { x = 10, y = 20 }
```

A numerical vector as used in the mask icon drawing commands is just a table with 1-based consecutive integer keys and numeric values. It can be constructed using the shorthand form of omitting the keys and just specifying the values:

```
a = { 1, 2, 3 }
```

Alternatively, a numerical vector can be constructed using the following PLECS specific syntax extension:

```
a = Vector{ 1, 2, 3 }
```

A vector constructed in this way can be added to or multiplied with a scalar value. This is especially helpful in icon drawing functions if the graphic object shall be displaced or scaled:

```
a_x = Vector{ 1, 2, 3 }*zoom_factor + x_offset  
a_y = Vector{ 4, 5, 6 }*zoom_factor + y_offset
```

### Comments

A simple comment starts anywhere with two consecutive hyphens (--) and runs until the end of the line. Long comments start with two hyphens followed by two square brackets (--[[ ]) and continue until the first occurrence of two closing square brackets (]]).

A common trick is to use --]] to end a long comment to quickly comment and uncomment a block of code:

```
--[[  
Icon:line({-10, 10}, {-10, 10})      -- commented out  
--]]
```

To un-comment the code block, prepend another hyphen to the first line:

```
---[[  
Icon:line({-10, 10}, {-10, 10})      -- will be executed  
--]]
```

In the first example, the --[[ starts a long comment that continues until (and including) the ]] in the third line. In the second example, the first two hyphens start an ordinary single-line comment, and so the second line is not commented out. The two hyphens at the beginning of the third line again start a single-line comment. Without these hyphens, the unpaired closing square brackets in the third line would cause a syntax error.

### Statements

Lua does not need an explicit separator between consecutive statements, but you can use a semicolon if you wish. Also, line breaks are treated like ordinary white space, so you can split a single statement into multiple lines without having to use a special continuation mark.

## Relational Operators

Lua supports the following relational operators:

```
< > <= >= == ~=
```

Relational operators always return a Boolean value. The == operator tests for equality and the ~= operator for inequality. They can be applied to any two values. If the values have different types, Lua considers them not equal. Otherwise, they are compared according to their type.

## Functions

A function is defined as follows

```
local function drawTriangle(x, y)
  Icon:line(Vector{0, 8.66, -8.66, 0}+x,
            Vector{-10, 5, 5, -10}+y)
end
```

As with variables, the `local` keyword is required because without it the function would be declared *globally*, which PLECS forbids. A function definition consists of the keyword `function`, a *name* (`drawTriangle`), a list of *parameters* (`x, y`), a *body*, i.e. a list of statements, and the terminator `end`. Parameters are local variables that are initialized with the values of the arguments passed in the function call.

The example above defines a function that draws a triangle with the center point  $x, y$ . The function can be called as follows

```
drawTriangle(-10, 0)
drawTriangle(10, 0)
```

A function can also return values using the `return` statement:

```
local function sum(values)
  local result = 0
  for i = 1, #values do
    result = result + values[i]
  end
  return result
end

local x = sum({1, 2, 3})    -- x will be 6
```

## Control Structures

Lua provides control structures for conditional execution and iteration.

**if then else** The `if` statement tests its condition and executes the `then` branch if the condition is true and otherwise the `else` part. The condition can result in any value, but as mentioned earlier, Lua treats all values other than `nil` and `false` as true. In particular, both the numerical 0 and the empty string (`'`) are treated as true.

```
if Dialog:get('choice') == '1' then
    Icon:text('+')
else
    Icon:text('-')
end
```

Multiple conditions can be tested with the `elseif` statement. It is similar to an `else` followed by an `if` but avoids the need for an extra `end`:

```
if Dialog:get('choice') == '1' then
    Icon:text('+')
elseif Dialog:get('choice') == '2' then
    Icon:text('-')
else
    Icon:text('+/-')
end
```

**for** The `for` statement has the following form:

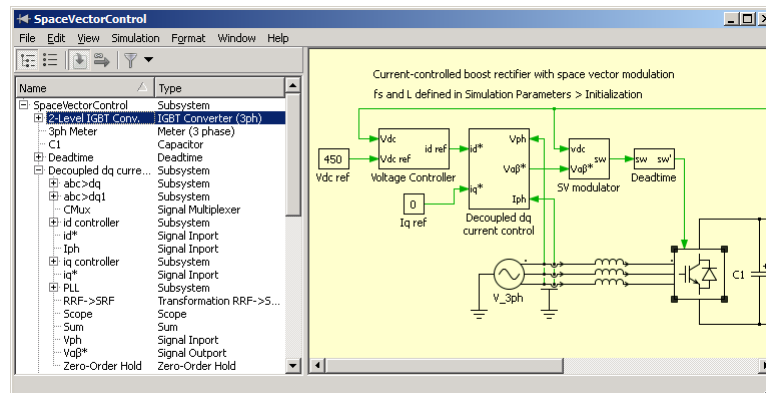
```
for x = -10, 10, 5 do
    Icon:line({x, x}, {-10, 10})
end
```

The loop variable, `x`, which is implicitly local to the loop, is successively assigned the values from -10 to 10 with an increment of 5, and for each value the loop body is executed. The result of the example will be five parallel vertical lines. The step value is optional; if it is omitted, Lua will assume a step value of 1.

Lua also provides other control structures (iterator-based `for`, `while`, `repeat`). However, these are not relevant for dynamic masks.

## Circuit Browser

The Circuit Browser enables you to navigate a circuit diagram hierarchically. To display the Circuit Browser, select **Show circuit browser** from the **View** menu of the schematic editor. Note that this option is only available from the main schematic window, i.e. the Circuit Browser will only be shown once for every model.



The editor window splits into two panes. The left pane shows the Circuit Browser, the right pane displays the current schematic. The Circuit Browser shows either a hierarchical or a flat list of all components and subsystems in the circuit.

Selecting a component in the Circuit Browser will also select the same component in the schematic and vice versa. By dragging the mouse or by holding down the shift key you can select multiple components. If the selected components are compatible with each other, their parameters can be edited simultaneously. The parameter dialog is displayed on double clicking the selected components or on clicking the option **Component parameters...** from the context menu.

The schematic is automatically updated to show the last selected component. The schematic of a specific subsystem can be displayed by double clicking it in the Circuit Browser.

The components can be sorted by clicking on one of the column headers. To quickly find e.g. all diodes in a circuit, switch to the flat view (see below) and sort it by clicking on **Type**. This will group all diodes together in the list.

## Viewing Options

In the toolbar of the Circuit Browser, several viewing options are available.

### View As Tree

The first two buttons in the toolbar switch between hierarchical and flat view. In the hierarchical (or tree) view, the first entry 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 subsystems. By double-clicking on the entry you can expand or collapse the list of these subsystems. This will also show the schematic of the subsystem.

### View As Flat List

The flat view lists all components of the entire circuit except the top-level circuit itself. This view has an additional column that shows the path of each component.

### Look Under Masks

By default, the Circuit Browser treats a masked subsystem like a component, i.e. it will not list the contents of the subschematic. You can change this behavior by toggling this button.

### Follow Library Links

By default, the Circuit Browser treats a library link like a component, i.e. it will not list the contents of the subschematic. You can change this behavior by toggling this button.

### Filter

The Filter button lets you choose, which components are shown in the Circuit Browser. When filtering is turned off, all components are shown. By clicking on the small arrow symbol, you can select between two filtering modes:

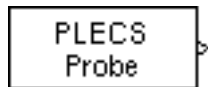
**Show only subsystems** When this mode is active, the Circuit Browser shows only subsystems.

**Show assertions** When this mode is active, the Circuit Browser shows only assertion blocks and components with attached assertions (see “Assertions” on page 85). To view the associated assertion parameters, double-click on the entry in the Circuit Browser.

**Custom filter** When this mode is active, a search bar is shown. The Circuit Browser shows only those components that match the search criterion or all components if the search string is empty.

The default filtering mode of the Circuit Browser can be specified in the PLECS preferences (see section “Configuring PLECS” on page 41).

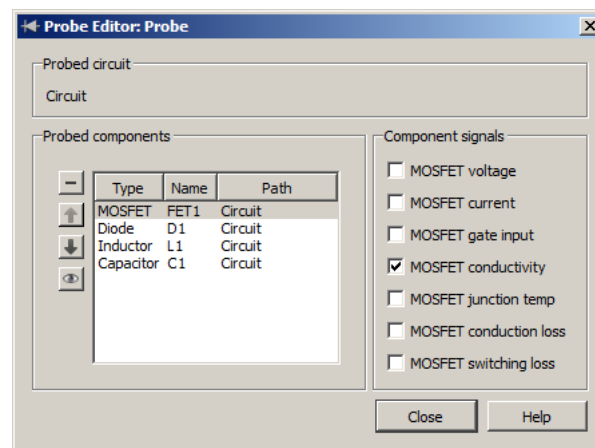
## PLECS Probe



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 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 using the **Up** ↑, **Down** ↓ and **Remove** – buttons. If you click on the **Show component**  button, the currently selected component will be shown in the schematic editor.

**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.

---

## Assertions

Assertions allow you to monitor arbitrary signals during a simulation and raise a warning or error message if they fail to meet a given condition. For instance, imagine you want to ensure that a certain component operates within a safe temperature range. Once the temperature leaves the defined operating range, you would like to receive a notification. In PLECS, this can be achieved using assertions.

Assertions are conditions that are assumed to hold during the entire simulation of a model. When a condition becomes invalid, i.e. when an assertion fails, PLECS executes a predefined action. The possible actions are:

- to **ignore** the failed assertion
- to add a **warning** message to the diagnostics window
- to add a **warning** message to the diagnostics window and additionally **pause** the simulation
- to add an **error** message to the diagnostics window and immediately **stop** the simulation

There is a global model parameter that allows you to override the actions defined locally in the individual assertions (see “Simulation Parameters” on page 103).

---

**Note** In PLECS Standalone, assertions are partially disabled during analyses (see “Analysis Tools” on page 161) and simulation scripts (see “Simulation Scripts” (on page 233)). In a Steady-State Analysis, assertions are only enabled during the final period shown at the end of the analysis. Assertions are not allowed to pause the Steady-State analysis, but it may be aborted if an assertion triggers an error message. In all other analyses, assertions are entirely disabled. Assertions are not allowed to pause a simulation script, but a script may be aborted if an assertion raises an error message.

---

PLECS provides two kinds of assertions: built-in assertion blocks and assertions that can be established on components based on their probe signals.

## Assertion Blocks

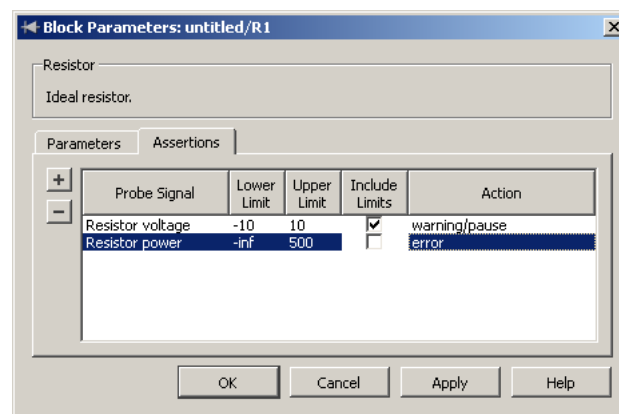
There is one basic Assertion block (see page 323) that interprets its input signal as a boolean value. While the value is non-zero, the assertion holds; whenever the input becomes zero, the assertion fails.

Because assertions are often used to ensure that a signal is within a certain range or below or above a certain limit, PLECS provides additional assertion blocks to directly do this. These blocks are:

- Assert Dynamic Lower Limit (see page 320)
- Assert Dynamic Range (see page 321)
- Assert Dynamic Upper Limit (see page 322)
- Assert Lower Limit (see page 324)
- Assert Range (see page 325)
- Assert Upper Limit (see page 326)

## Component Assertions

PLECS also provides the possibility to add assertions directly to components. This can be used to define a valid range for any probe signal of a component. To add assertions to a component, open its parameter dialog and click on the **Assertions** tab. If the parameter dialog of a component does not provide this tab, it is not possible to add assertions.



Use the “+” and “-” buttons to add or remove assertions. In the different columns, the parameters for the assertions can be provided: the probe signal

that is limited by the assertion, the lower and upper limits, whether the limits should be inclusive, and the action executed when the assertion fails (see above). If the limits are included (by setting a check mark in that column), the limits themselves are considered part of the valid range, otherwise the signal has to be strictly within the limits. The values `-inf` and `inf` may be used to disable the lower or upper limit, respectively.

## Locating Assertions

To quickly get an overview of all assertions that are defined in a model, open the Circuit Browser (see page 79) and set the filter option to **Show assertions**. The Circuit Browser will then list only assertion blocks and components, for which assertions have been defined.

## Controlling Access to Circuits and Subsystems

PLECS allows you to control user access to individual subsystems 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 subsystem, choose **Permissions...** from the **Subsystem** submenu of the **Edit** menu or 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 subsystem.
- The **Modify** privilege controls whether a user can modify the schematic of a circuit or subsystem. For a subsystem 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 Subsystems

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 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 237) 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)

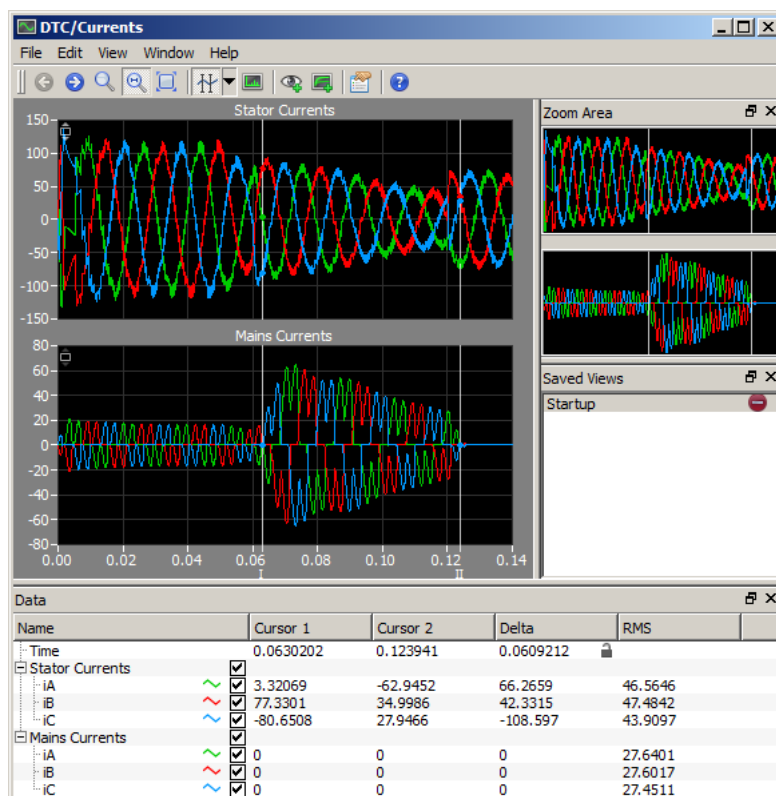
To export a schematic choose **Export...** from the **File** menu and select your desired output format. A second dialog lets you specify the export options for the specific format, e.g. the bitmap resolution.

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 macOS) 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 dialog 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.

## Scrolling

During a simulation, if the current x-axis range is smaller than the simulation time span, the scope will automatically scroll the x-axis so that the current simulation time step is always shown. The scrolling mode (paged or continuous) may be specified in the scope parameters.

Scrolling is suspended if you manually zoom or pan so that the current simulation time is greater than the right x-axis limit. This is indicated by a stack of small arrows at the right border of the plot. Scrolling is resumed if you click on these arrows or if you zoom or pan so that the current simulation time is less than the right x-axis limit.

## Y-Axis Auto-Scaling

The scope features a dynamic auto-scaling mode, in which the y-axis limits are always adjusted to the currently visible waveforms. The state of the auto-scaling mode is indicated by a small semi-transparent icon () in the top-right corner of a plot. Clicking on this icon will toggle the state. Auto-scaling will be enabled for all plots in the scope if you click the **Zoom to Fit** button. If you zoom in the y direction of a plot, auto-scaling will be disabled for that

plot. The initial auto-scale state at simulation start is specified for each plot individually in the scope parameters (see page 573).

## 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 43).

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 appropriately. 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 98).

## 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 235).

## 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 parameter description of the Scope block (see page 573). The plot background color can be changed in the PLECS preferences (see section “Configuring PLECS” on page 41).

## 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 spectral leakage 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, linear** displays the absolute value of each coefficient. **Absolute, logarithmic** displays the common logarithm of the absolute values, multiplied by 20. **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 91) for details.

## Calculation of the Fourier coefficients

The following approximation is made to calculate the Fourier coefficients of a signal with variable sampling intervals  $\Delta T_m$ :

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

where

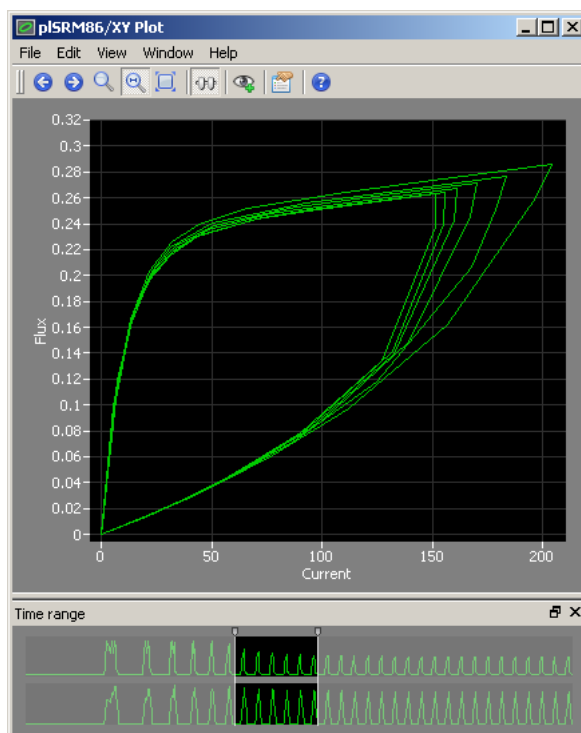
$$f_m(t) = a_m t + b_m \quad \text{for continuous signals}$$

$$f_m(t) = b_m \quad \text{for discrete signals}$$

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,  $\Delta T_m$ : 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. You can choose to draw trajectories by connecting consecutive points with a direct line, to draw a vector from the origin to the current point or a combination of both.



### 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 vectors are drawn, the right end of the time range determines the position of the vector head.

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 722).

### **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 91) for details.

## Simulation Parameters

### PLECS Blockset Parameters

This section describes the simulation parameters available in PLECS Blockset for Simulink. For the PLECS Standalone simulation parameters please refer to the next section (see page 107).

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 33). 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  $\Delta t$  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.

**Disc. method** This parameter determines the algorithm used to discretize the state-space equations of the electro-magnetic model.

**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 454) instead. The non-linear feedback paths in this model contain Integrator blocks (see page 464) which prevent the algebraic loops.

---

## Diagnostics

**Zero crossing detection disabled** In order to accurately determine the proper switching times of power semiconductors, PLECS highly depends on the solver's capability to locate zero crossings. If you switch off the zero crossing detection in the Simulink solver or use the less accurate "Adaptive" detection algorithm, PLECS will therefore issue a diagnostic message. This option allows you to specify the severity level (warning or error) of this message.

If you encounter problems due to many consecutive zero crossings, it is usually not advisable to modify the zero crossing detection settings. Consecutive zero crossings are often caused by insufficient simulation accuracy, typically in conjunction with a stiff model. In this case it may help to tighten the relative tolerance of the Simulink solver (from the default  $1e-3$  to  $1e-5$  or  $1e-6$ ) and to switch from the default solver ode45 to a stiff solver such as ode23tb and, where applicable, set the Simulink solver option **Solver reset method** to Robust.

**Number of consecutive gate signal changes** If you configure a Signal Inport block (see page 580) in a top-level schematic to be a *gate signal*, PLECS expects this signal to change only at discrete instants. If instead the signal changes in more than the specified number of consecutive simulation time steps, PLECS will issue an error message to indicate that there may be a problem in the gate signal generator. You can disable this diagnostic by entering 0.

**Division by zero** This option determines the diagnostic action to take if PLECS encounters a division by zero in a Product block (see page 530) or a Function block (see page 398). A division by zero yields  $\pm\infty$  or nan („not a

number”, if you divide 0/0). Using these values as inputs for other blocks may lead to unexpected model behavior. Possible choices are ignore, warning and error. In new models, the default is error. In models created with PLECS 3.6 or earlier, the default is warning.

**Algebraic loop with state machines** This option determines the diagnostic action to take if PLECS detects an algebraic loop that includes a State Machine (see page 600). This may lead to unexpected behavior because the State Machine will be executed multiple times for the same simulation step during the iterative solution of the algebraic loop. Possible choices are ignore, warning and error. The default is error.

**Negative switch loss** This option determines the diagnostic action to take if PLECS encounters negative loss values during the calculation of switch losses (see “Loss Calculation” on page 119). PLECS can issue an error or a warning message or can continue silently. In the latter two cases, the losses that are injected into the thermal model are cropped to zero.

**Assertion action** Use this option to override the action that is executed when an assertion fails (see Assertion block on page 323). The default is use local settings, which uses the actions specified in each individual assertion. Assertions with the individual setting ignore are always ignored, even if this option is different from use local settings. Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).

## Sample times

**Synchronize fixed-step sample times** This option specifies whether PLECS should attempt to find a common base sample rate for blocks that specify a discrete sample time.

**Use single base sample rate** This option specifies whether PLECS should attempt to find a *single* common base sample rate for *all* blocks that specify a discrete sample time.

These options can only be modified for a Continuous State-Space model; for a Discrete State-Space model they are checked by default. For details see section “Multirate Systems” (on page 38).

## Algebraic loops

**Method** Use this option to select the strategy adopted by the nonlinear equation solver. Currently, either a *line search* method or a *trust region*



method can be used.

**Tolerance** The relative error bound. The solver updates the block outputs iteratively until the maximum relative change from one iteration to the next and the maximum relative residual of the loop equations are both smaller than this value.

### State-space calculation

**Use extended precision** When this option is checked, PLECS uses higher-precision arithmetics for the internal calculation of the state-space matrices for a physical model. Check this option if PLECS reports that the system matrix is close to singular.

**Enable state-space splitting** When this option is checked, PLECS will attempt to split the state-space model for a physical domain into smaller independent models that can be calculated and updated individually. This can reduce the calculation effort at runtime, which is particularly advantageous for real-time simulations.

**Display state-space splitting** When this option is checked, PLECS will issue diagnostic messages that highlight the components that make up the individual state-space models after splitting. This is useful e.g. in order to connect Model Settings blocks in the appropriate places (see Electrical Model Settings on page 389, Rotational Model Settings on page 552 and Translational Model Settings on page 672).

### Simulink Coder

**Target** This option specifies the code generation target that is used when you generate code with the Simulink Coder. For details on the available targets see section “Code Generation Targets” (on page 270).

**Inline circuit parameters for RSim target** This option controls whether PLECS inlines parameter values or whether it should keep them tunable when it creates code for the RSim target. For details see section “Tunable Circuit Parameters in Rapid Simulations” (on page 272).

## 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 33). 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

**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 0.

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.

---

**Disc. method** This parameter determines the algorithm used to discretize the state-space equations of the electro-magnetic model.

**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.

## Sample times

**Synchronize fixed-step sample times** This option specifies whether PLECS should attempt to find a common base sample rate for blocks that specify a discrete sample time.

**Use single base sample rate** This option specifies whether PLECS should attempt to find a *single* common base sample rate for *all* blocks that specify a discrete sample time.

These options can only be modified for a variable-step solver; for a fixed-step solver they are checked by default. For details see section “Multirate Systems” (on page 38).

## State-space calculation

**Use extended precision** When this option is checked, PLECS uses higher-precision arithmetics for the internal calculation of the state-space matrices for a physical model. Check this option if PLECS reports that the system matrix is close to singular.

**Enable state-space splitting** When this option is checked, PLECS will attempt to split the state-space model for a physical domain into smaller independent models that can be calculated and updated individually. This can reduce the calculation effort at runtime, which is particularly advantageous for real-time simulations.

**Display state-space splitting** When this option is checked, PLECS will issue diagnostic messages that highlight the components that make up the individual state-space models after splitting. This is useful e.g. in order to connect Model Settings blocks in the appropriate places (see Electrical Model Settings on page 389, Rotational Model Settings on page 552 and Translational Model Settings on page 672).

## Assertions

**Assertion action** Use this option to override the action that is executed when an assertion fails (see Assertion block on page 323). The default is use `local settings`, which uses the actions specified in each individual assertion. Assertions with the individual setting `ignore` are always ignored, even if this option is different from `use local settings`. Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).

## Algebraic loops

**Method** Use this option to select the strategy adopted by the nonlinear equation solver. Currently, either a *line search* method or a *trust region* method can be used.

**Tolerance** The relative error bound. The solver updates the block outputs iteratively until the maximum relative change from one iteration to the next and the maximum relative residual of the loop equations are both smaller than this value.

## Diagnostics

**Division by zero** This option determines the diagnostic action to take if PLECS encounters a division by zero in a Product block (see page 530) or a Function block (see page 398). A division by zero yields  $\pm\infty$  or nan („not a number”, if you divide 0/0). Using these values as inputs for other blocks may lead to unexpected model behavior. Possible choices are ignore, warning and error. In new models, the default is error. In models created with PLECS 3.6 or earlier, the default is warning.

**Negative switch loss** This option determines the diagnostic action to take if PLECS encounters negative loss values during the calculation of switch losses (see “Loss Calculation” on page 119). PLECS can issue an error or a warning message or can continue silently. In the latter two cases, the losses that are injected into the thermal model are cropped to zero.

**Stiffness detection** This parameter only applies to the non-stiff, variable-step DOPRI solver. The DOPRI solver contains an algorithm to detect when a model becomes „stiff” during the simulation. Stiff models cannot be solved efficiently with non-stiff solvers, because they constantly need to adjust the step size at relatively small values to keep the solution from becoming numerically unstable.

If the DOPRI solver detects stiffness in model, it will raise a warning or error message depending on this parameter setting with the recommendation to use the stiff RADAU solver instead.

**Max. number of consecutive zero-crossings** This parameter only applies to variable-step solvers. For a model that contains discontinuities (also termed „zero-crossings”), a variable-step solver will reduce the step size so as to make a simulation step precisely at the time when a discontinuity occurs (see “Event Detection Loop” on page 32). If many discontinuities occur in subsequent steps, the simulation may come to an apparent halt without actually

stopping because the solver is forced to reduce the step size to an excessively small value.

This parameter specifies an upper limit for the number of discontinuities in consecutive simulation steps before PLECS stops the simulation with an error message that shows the responsible component(s). To disable this diagnostic, set this parameter to 0.

**Algebraic loop with state machines** This option determines the diagnostic action to take if PLECS detects an algebraic loop that includes a State Machine (see page 600). This may lead to unexpected behavior because the State Machine will be executed multiple times for the same simulation step during the iterative solution of the algebraic loop. Possible choices are ignore, warning and error. The default is error.

## 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 50).

---

**Note** The maximum length of variable names is 63 characters. This is due to the way, in which a workspace is stored in PLECS and exchanged with Octave.

---

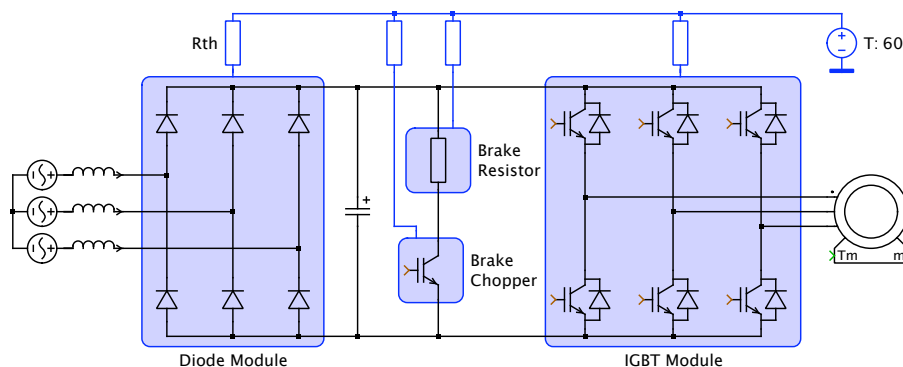


# Thermal Modeling

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 406) 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.

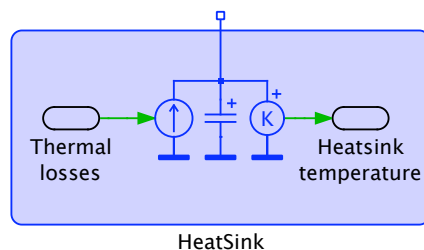


Heat conduction from one heat sink to another or to an ambient temperature is modeled with lumped thermal resistances and capacitances that are con-

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 349) or the Controlled Temperature block (see page 351).

## Thermal Loss Dissipation

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

### Semiconductor Losses

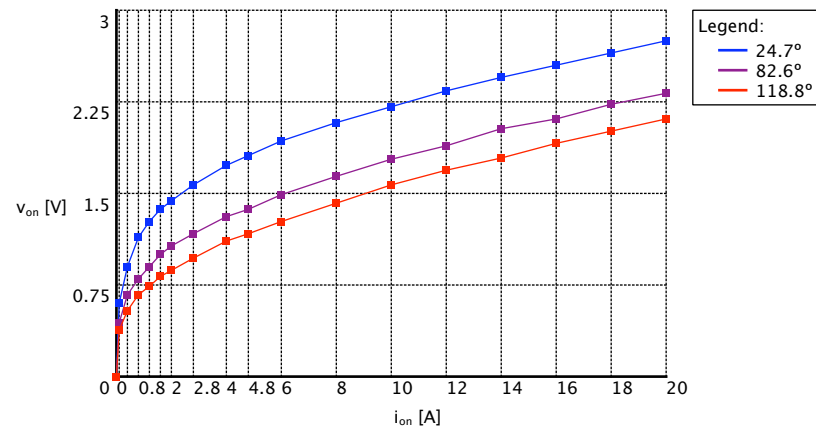
Power semiconductors dissipate losses due to their non-ideal nature. These losses can be classified as conduction losses and switching losses. For completeness the blocking losses due to leakage currents need to be mentioned, but they can usually be neglected.

Semiconductor losses are specified by referencing a thermal data sheet in the component parameter **Thermal description**. See section “Thermal Description Parameter” (on page 123) and “Thermal Library” (on page 126) 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:  $v = v_{on}(i, T)$ . You may also specify additional custom function arguments. This function is defined in the **Conduction loss** tab of the thermal description as a 2D look-up table or a functional expression (see “Thermal Editor” on page 128).

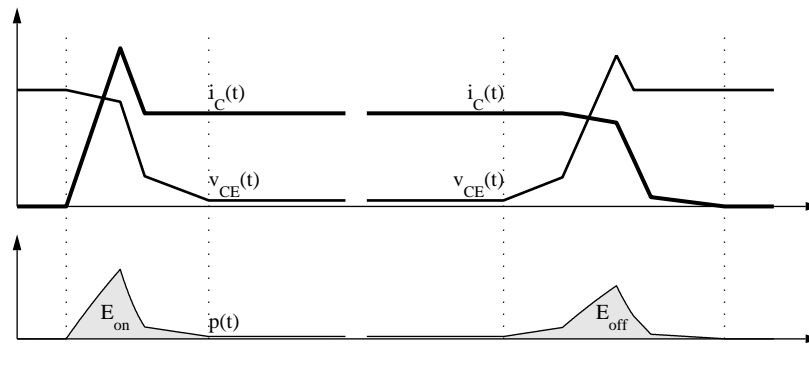


A setting of 0 V for a single temperature and current value means no conduction losses. If you do not specify a thermal description in the device parameters, the default will be used, i.e. the losses are calculated from the electrical device parameters.

**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

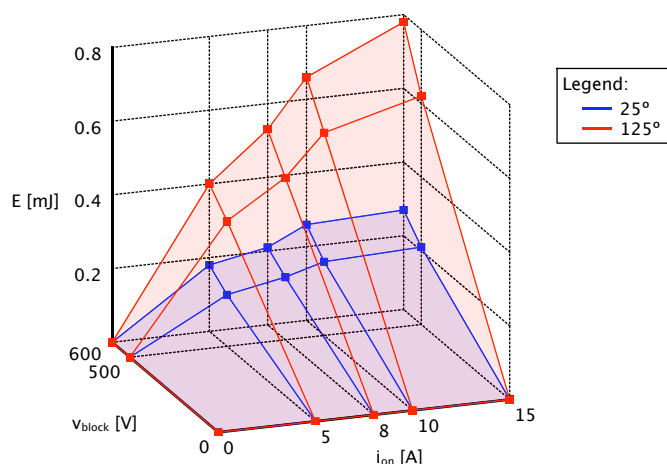
Switching losses occur because the transitions from on-state to off-state and vice versa do not occur instantaneously. During the transition interval both the current through and the voltage across the device are substantially larger than zero which leads to large instantaneous power losses. This is 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:  $E = E_{on}(v_{block}, i_{on}, T)$ ,  $E = E_{off}(v_{block}, i_{on}, T)$ . You may also specify additional custom function arguments. These functions are defined in the tabs **Turn-on loss** and **Turn-off loss** of the thermal editor as 3D look-up tables or functional expressions (see “Thermal Editor” on page 128).

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.

---

## Loss Calculation

As described above, the conduction and switching losses are defined by means of look-up tables. From these tables the actual losses are calculated during a simulation using linear interpolation if the input values (on-state current, pre- and post-switching current or voltage, junction temperature) lie within the specified index range. If an input value lies out of range, PLECS will *extrapolate* using the first or last pair of index values.

If the calculated loss value is negative, PLECS will issue a diagnostic message and/or crop the value to zero. You can select the diagnostic action to be taken with the diagnostic parameter **Negative switch loss** in the simulation parameters dialog (see “PLECS Blockset Parameters” on page 103 and “PLECS Standalone Parameters” on page 107).

### Supported devices

Semiconductor components that implement this loss model are

- the Diode (see page 361),
- the Thyristor (see page 639),
- the GTO (see page 401),
- the GTO with Diode (see page 403),
- the IGBT (see page 414),
- the IGBT with Diode (see page 434),
- the Reverse Blocking IGCT (see page 440),
- the Reverse Conducting IGCT (see page 442),
- the MOSFET (see page 493),
- the MOSFET with Diode (see page 496) and
- the TRIAC (see page 687).

In addition, the Set/Reset Switch (see page 575) 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:

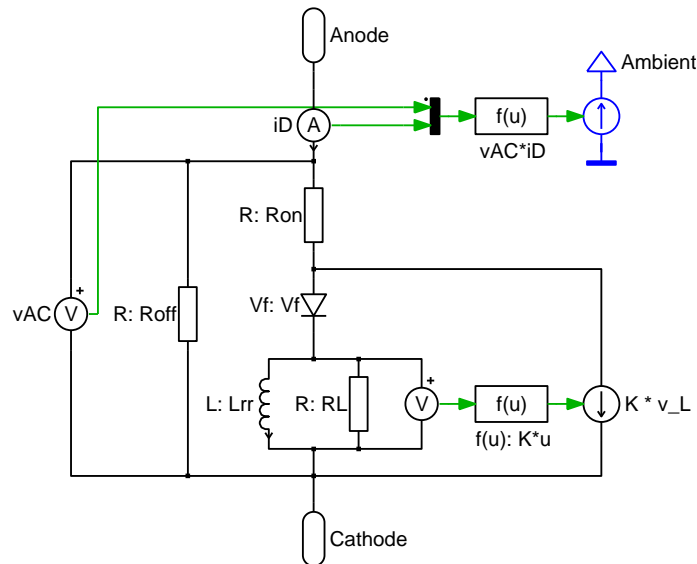
- the Resistor (see page 542),
- the Variable Resistor with Variable Series Inductor (see page 709),
- the Variable Resistor with Constant Series Inductor (see page 706),
- the Variable Resistor with Variable Parallel Capacitor (see page 707) and
- the Variable Resistor with Constant Parallel Capacitor (see page 705).

## Heat Sinks and Subsystems

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 316). 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 363). 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.

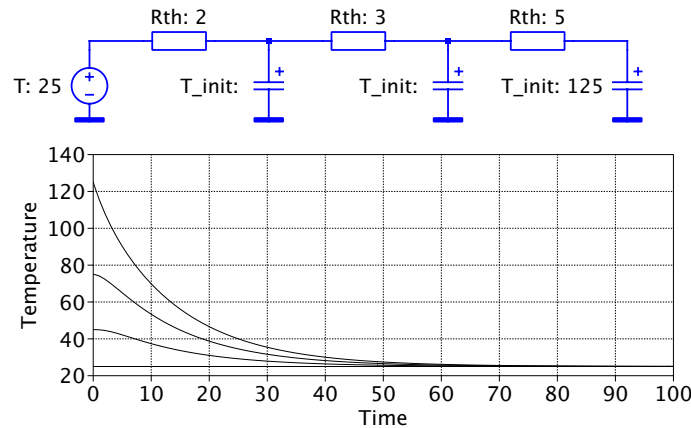
The Diode with Reverse Recovery therefore uses a Controlled Heat Flow block (see page 350) to inject the electrical power loss into the thermal model via the Ambient Temperature block. The power loss is calculated by multiplying the device voltage and the device current.

## Temperature Initialization

The state variables of a thermal model are the temperatures of thermal capacitances, and like other state variables they need to be initialized with a starting value. For this purpose the Thermal Capacitor (see page 631) and other components that implicitly contain thermal capacitances have a parameter **Initial temperature**, that allows you to specify this starting value.

However, you can also let PLECS calculate the initial value for you based on other temperatures in the thermal system. To do so, simply leave the parameter blank or enter nan (a floating point constant standing for “Not A Number”). At the beginning of a simulation, PLECS will perform a “DC analysis”, treating thermal capacitances with known initial values like constant temperature sources and calculating the unknown initial values such that the system would be in steady state.

As an example, consider the following thermal system consisting of a constant temperature source and three thermal R/C pairs. If you leave the **Initial temperature** parameter of the three capacitances blank, all three will “inherit” the starting temperature from the source. On the other hand, if you leave only the parameters of the first two capacitances blank and specify an initial value of 125 for the third one, PLECS will initialize the first capacitance with  $T_0 = 25 + (125 - 25) \cdot \frac{2}{2+3+5} = 45$  and the second one with  $T_0 = 25 + (125 - 25) \cdot \frac{2+3}{2+3+5} = 75$ . Of course, as soon as the simulation starts, the temperatures in all three capacitances will eventually drop to the temperature of the source.





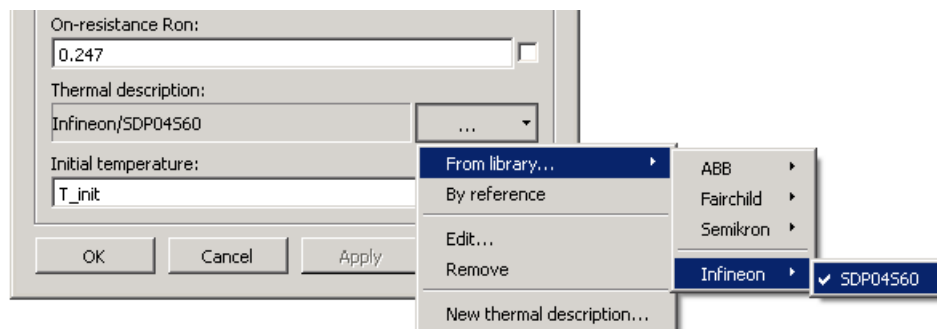
## Thermal Description Parameter

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 126) 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 237)).

# 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 128)). 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 41)). 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 237)).

---

## Browsing the Thermal Library

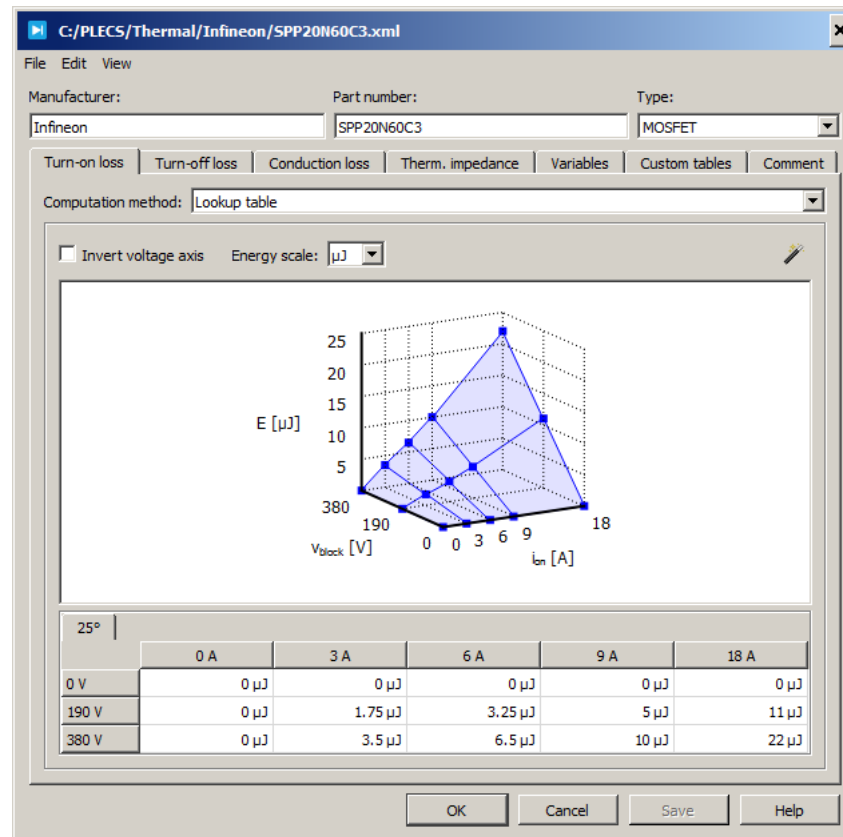
PLECS allows for browsing the thermal library with the **Thermal library browser**. It is invoked from the **Window** menu.

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 **Window** menu) or by assigning a data sheet to a semiconductor in the **Thermal description** parameter and then selecting the menu entry **Edit...**

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 126) for details of the structure of the thermal library.



The Thermal Editor window has the following input elements:

**Manufacturer, Part number** These text fields are for documentation purposes only.

**Type** This selector serves as a filter for the **Thermal description** menu entry on the **Thermal** tab of a semiconductor parameter dialog. It must be set according to the semiconductor type it is intended to be used with.

**Turn-on loss, Turn-off loss, Conduction loss** On these tabs you define the switching and conduction losses of the device. See “Editing Switching Losses” (on page 129) and “Editing Conduction Losses” (on page 130).

**Therm. impedance** On this tab you define the thermal impedance between the junction and the case of the device. See “Editing the Thermal Equivalent Circuit” (on page 131).

**Variables** On this tab you define custom parameters (such as gate resistance or stray inductance) that may be used to define device losses. Here you can also specify hard limits both for your custom parameters and for intrinsic variables, i.e. the blocking voltage, the device current and the junction temperature.

**Custom tables** On this tab you define custom lookup tables that may be used to define device losses.

**Comment** This tab provides you with a text field that you may use for documentation purposes.

## Editing Switching Losses

Switching losses are defined on the **Turn-on loss** and **Turn-off loss** tabs. The **Computation method** popup specifies whether the loss function is defined as a 3D lookup table, a functional expression or a combination of both.

If you select **Lookup table**, the pane below will show a 3D lookup table with the blocking voltage, the device current and the junction temperature as input variables. For more information regarding lookup tables see “Editing Lookup Tables” (on page 134).

If you select **Formula**, the pane below will show a text field that allows you to enter a functional expression. A formula may consist of numerical constants including pi, arithmetic operators (+ - \* / ^), mathematical functions (abs, acos, asin, atan, atan2, cos, cosh, exp, log, log10, max, min, mod, pow, sgn, sin, sinh, sqrt, tan, and tanh), brackets and the function arguments. The default function arguments are the blocking voltage  $v$ , the device current  $i$  and

the junction temperature  $T$ . You may define additional function arguments on the **Variables** tab (see “Adding Custom Variables” on page 132). You may also reference custom lookup tables using the function lookup (see “Adding Custom Lookup Tables” on page 133).

If you select **Lookup table and formula**, the pane below will show both lookup table and formula field. With this method, an energy  $E$  is first computed from the lookup table and may then be used in the formula to calculate the final loss energy value. For instance, in order to quickly increase the switching loss by 20%, you could enter  $1.2 * E$  into the formula field.

### Editing Conduction Losses

Conduction losses are defined by means of the on-state voltage drop on the **Conduction loss** tab. The **Computation method** popup specifies whether the voltage drop is defined as a 2D lookup table, a functional expression or a combination of both.

If you select **Lookup table**, the pane below will show a 2D lookup table with the device current and the junction temperature as input variables. For more information regarding lookup tables see “Editing Lookup Tables” (on page 134). On using the import wizard for constructing lookup table data from vendor plots, see “Import data from plot images” (on page 136).

If you select, **Formula**, the pane below will show a text field that allows you to enter a functional expression. The default function arguments are the device current  $i$  and the junction temperature  $T$ . You may define additional function arguments on the **Variables** tab (see “Adding Custom Variables” on page 132). You may also reference custom lookup tables using the function lookup (see “Adding Custom Lookup Tables” on page 133).

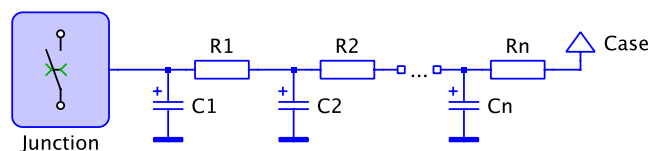
If you select **Lookup table and formula**, the pane below will show both lookup table and formula field. With this method, a voltage  $v$  is first computed from the lookup table and may then be used in the formula to calculate the final voltage drop value. For instance, in order to quickly increase the voltage drop by 20%, you could enter  $1.2 * v$  into the formula field.



## Editing the Thermal Equivalent Circuit

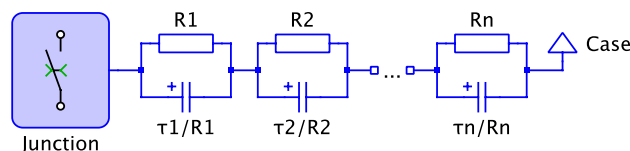
The thermal equivalent circuit of a component describes its physical structure in terms of thermal transitions from the junction to the case. Each transition consists of a thermal resistor and a thermal capacitor. They can be edited on the **Therm. impedance** tab of the thermal editor. The thermal equivalent 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  $\tau_i$  (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**.

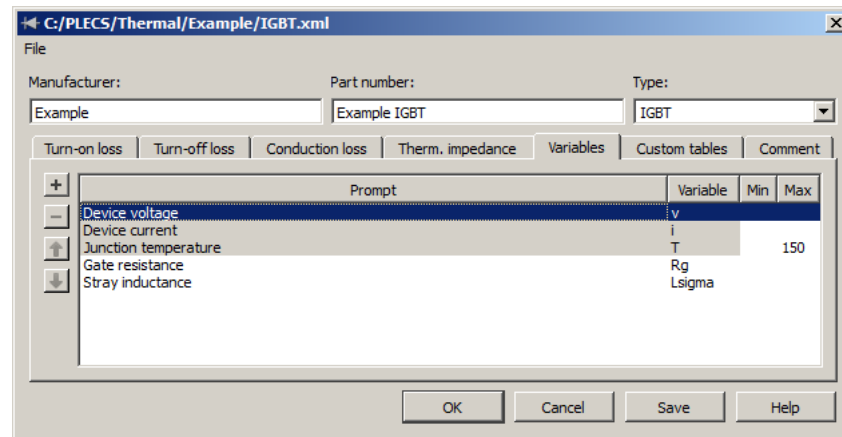


### Foster network

**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.

## Adding Custom Variables

Custom variables, such as gate resistance or stray inductance, that you wish to use in the definition of device losses may be defined on the **Variables** tab.



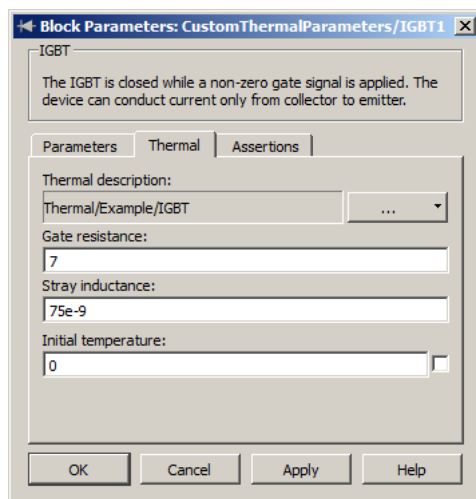
Use the **Add +**, **Remove -**, **Up ↑** and **Down ↓** buttons on the left to add or remove custom variables or to reorder them. Note that the first three lines in the list are reserved for the intrinsic variables and may not be removed or reordered.

When you add a custom variable, you may specify a **Prompt**, which should provide a brief description of the purpose variable, and a **Variable**, which must be a unique identifier. This identifier may then be used in the formula expressions that define the device losses.

In the **Min** and **Max** columns you may enter minimum and maximum allowed values for both intrinsic and custom variables. During a simulation, PLECS will monitor the actual values of the variables and raise a diagnostic message if a variable value exceeds a specified limit. The default action is to show an error and stop the simulation but this may be changed in the simulation parameters dialog using the diagnostic parameter **Loss variable limit exceeded** (see “PLECS Blockset Parameters” on page 103 and “PLECS Standalone Parameters” on page 107).

## Specifying custom variable values

When you select a thermal description with custom variables on the **Thermal** tab of a semiconductor parameter dialog, the dialog will show additional parameter fields for the custom variables using the prompts mentioned above. An example dialog is shown below.



Instead of static values that remain constant during a simulation you may also specify the label of a Signal Goto block (see page 579) for a custom variable. The label is a string consisting of a prefix for the scope (g: for global, s: for schematic and m: for masked subsystem) and the tag name of the Goto block. For example, if your model contains a Goto block with global scope and the tag name Rg, you would enter 'g:Rg' (including the quotation marks) in order to reference this signal in a custom variable of a thermal description. This can be used e.g. to simulate the effect of a gate drive that can dynamically change the effective gate resistance.

## Adding Custom Lookup Tables

Custom lookup tables are defined on the **Custom tables** tab. To add a new table, click on **New...** and specify a unique name and the number of dimensions of the new table. Use the **Duplicate...**, **Rename...** and **Remove** buttons to duplicate, rename or remove an existing custom table.

Custom tables can be used in function expressions for device losses using the lookup function, which is called with a string specifying custom table name and one to three numeric arguments depending on the number of dimensions of the table.

For example, consider that you have defined a custom variable  $R_g$  for the gate resistance and a custom table Gate Resistance Eon Scaler that describes how the turn-on losses scale in terms of the gate resistance. You could then use the **Lookup table and formula** method on the **Turn-on loss** tab, specify the nominal losses in the turn-on-loss lookup table and enter the following function expression:

```
E*lookup('Gate Resistance Eon Scaler', Rg)
```

### Editing Lookup Tables

When editing an intrinsic lookup table on one of the three loss tabs or a custom lookup table, you can add and remove new interpolation points for a table dimension with the **Edit** menu or the context menu in the table. To enter multiple values at once, separate them by semicolons or spaces.

To rotate and tilt a three-dimensional table view, click on an empty space with the left mouse button and drag the mouse while keeping the mouse button pressed.

### Lookup method

When calculating function values from a lookup table, PLECS uses linear interpolation if an input value lies within the index range for the corresponding table dimension. If the input value lies outside the index range, PLECS will extrapolate using the first or last pair of index values.

### Copy, Paste and Scaling

Thermal data can be copied and pasted within the tables of the thermal editor, and to or from other programs, like e.g. Microsoft Excel. This can be done using the context menu or by pressing **Ctrl-C/Ctrl-V** (or **cmd-C/cmd-V** on Mac OS X). To specify the target location for the data, you have to select a part of the table that has the same number of rows and columns as the copied

data. When copying from another program, only the first correctly formatted number in each table cell will be copied, any additional information (e.g. units) will be discarded.

Values selected in a table can be scaled by a given factor by right-clicking and choosing **Scale selected values...** from the context menu. To convert a value from 0.23 J to 0.23 mJ, e.g., you can scale it with a factor of 0.001. To only change the unit but not the actual value, i.e. to change 0.23 J to 230 mJ, use the **Energy scale** drop box at the top right.

## Importing Data from Graphical Datasheets

PLECS provides an **Import Wizard** that facilitates the import of data from graphs that are typically used on real datasheets. The wizard is opened by clicking on the magic wand icon (🪄) that appears in the top right corner of any page that allows you to enter tabular data, i.e. the loss and custom tables and the thermal impedance.

The screenshot shows the PLECS Import Wizard interface. The main window displays a graph of energy loss (E<sub>tot</sub>, E<sub>m</sub> [J]) versus current (I<sub>c</sub> [A]). The X-axis ranges from 0 to 1600 A, and the Y-axis ranges from 0.0 to 4.0 J. The graph shows a red line with data points and two black lines representing E<sub>tot</sub> and E<sub>m</sub>. Below the graph is a table with columns for current (400 A, 600 A, 800 A, 1000 A) and a row for energy loss (0 V). The table contains values: 0.55666 J, 0.89024 J, 1.3181 J, 1.8167 J. To the right is a settings panel for the graph, including X-axis [A] (X-min: 0, X-max: 1600, Snap: 100, Scale: linear) and Y-axis [J] (Y-min: 0, Y-max: 4, Scale: linear). The settings panel also includes Plot image options (Opacity and Orientation: Mirror axes).


When you open the import wizard for the first time on a particular page, you are requested to provide a graph image. To import an image, drag a image file to the empty wizard area or click into the area to open a file browser that lets you choose an image file. Image files must have a bitmap file format (PNG, GIF, BMP, JPG or XPM). To import graphs from a PDF file, take a snapshot of


the desired graph, then select **Paste** from the **Edit** menu of the editor window to paste the snapshot into the wizard.

After the image has been imported, a green coordinate system is drawn on top of it. Your first task should be to align the green axes with the coordinate system in the image. You can move an axis or change its length by dragging the axis itself or its end point with the mouse.

Ensure that the axes have the proper dimensions. For turn-on and turn-off losses, the x-axis is expected to be in Amperes and the y-axis, in Joules; for conduction losses, the x-axis is expected to be in Amperes and the y-axis, in Volts. If the dimensions in the image are swapped, you can flip the image by clicking on the **Mirror axes** button in the image configuration dialog (see below).

## Configuring the Graph Import

After you have aligned the green coordinate system, you need to enter the axis limits into the configuration dialog that has opened automatically when the image was imported. If you have closed the dialog, you can open it again by clicking on the  button in the wizard toolbar or on one of the green axis limit labels.

In addition to the minimum and maximum settings, the x-axis has a **Snap** property that is initialized automatically from the axis limits. You can override the snap value by entering a number here; entering 0 disables snapping. To restore the automatically calculated value, click on the  button.

Both x-axis and y-axis have a **Scale** property that lets you choose between a linear and logarithmic scale. By default, double-logarithmic scaling is used for thermal impedances only; for all other imports the scaling defaults to linear.

The **Opacity** slider lets you change the opacity of the graph image from fully transparent (or invisible) to fully opaque. The **Mirror axes** button will mirror the graph image *diagonally* so that the two axes are exchanged. This is useful e.g. when importing conduction losses where the graph typically shows Volts on the x-axis and Amperes on the y-axis.

## Adding and Moving Points

Points are added by double-clicking anywhere in the coordinate system. If snapping is enabled, the x-value is adjusted to the nearest snap value. To move points, drag them with the left mouse button. If snapping is enabled,

you can temporarily disable it by pressing and holding the **Shift** key when you click on a point.




To add new curves (e.g. for another temperature value) use the corresponding entry from the **Edit** menu or from the context menu of the table at the bottom of the editor. If there is more than one curve, a double-click will add new points to all curves: The point at the mouse location is added to the *current* curve, i.e. the curve that you added or interacted with most recently. For all other curves, points are added based on their neighboring points' coordinates.

Also, if there is more than one curve, the movement of points is restricted to the y-direction. The reason for this is that all curves in the look-up table share the same x-values, so changing the x-value of a point in one curve will affect all other curves as well. You can temporarily override this restriction by pressing and holding the **Shift** key when you click on a point.

### Zooming and Panning

You can zoom into the graph for more precise placement of points and axes. Zooming is controlled via the **View** menu and the corresponding buttons in the toolbar. You can also zoom in and out by holding the **Ctrl** key (**Cmd** key on macOS) while rolling the mouse wheel. When you have zoomed into the graph, you can pan the image using the sliders or by holding the **Ctrl** or **Cmd** key while pressing and dragging the left mouse button. Pressing the spacebar will zoom the graph to fit the window.

### Adding and Managing Graph Images

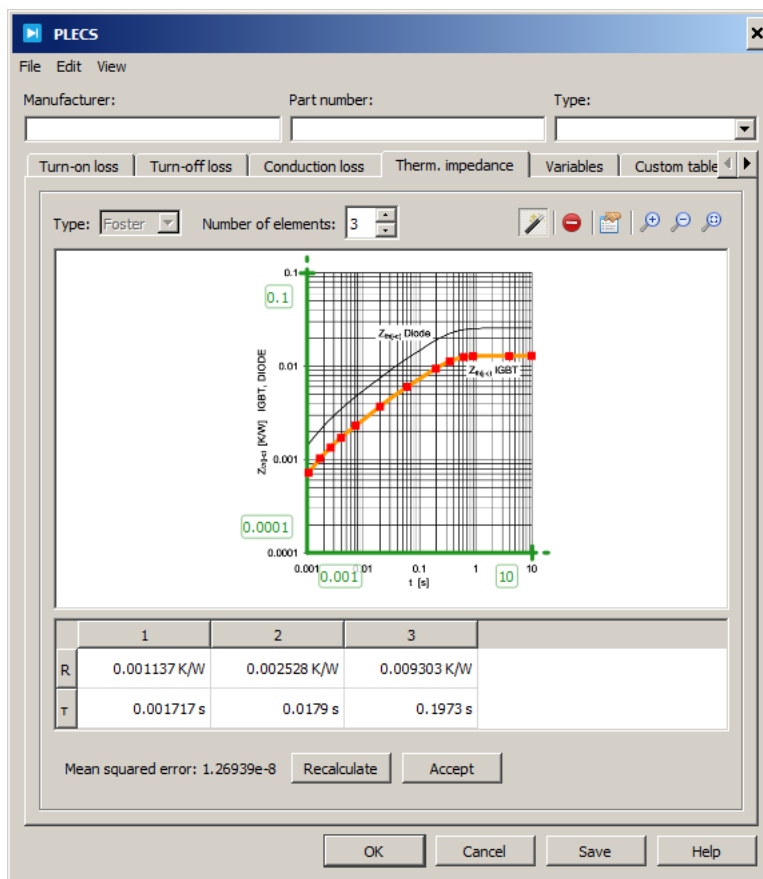
Sometimes, the curves for one look-up table come from different graphs. To add a new graph image to the wizard, click the  button in the toolbar. To change the visibility of a particular curve on the current graph, use the check box in the corresponding row header in the table at the bottom. Press the  button in the toolbar to rename the current graph; graph names are used purely for documentation purposes. To remove a graph (but not the curves), press the  button.

### Fitting Thermal Impedances

When a vendor datasheet provides the thermal impedance as a heating curve rather than Foster or Cauer network coefficients, you can use the import wizard to fit Foster coefficients to a given heating curve. First, import the graph



of the heating curve as described above and place a number of points on the heating curve, then choose the desired number of Foster elements. As a general rule, you must place at least two points per Foster element. As soon as these requirements are met, PLECS will calculate a set of Foster coefficients and display the result as an orange curve on top of the graph.



PLECS uses a non-deterministic optimization algorithm to minimize the error between the calculated curve and the points that you have placed. This algorithm may not converge at all or converge at a local instead of the global minimum. If the current fit is not satisfactory, you can calculate a new one by pressing the **Recalculate** button. Once the results are acceptable, press the **Accept** button to close the wizard and transfer the calculated Foster values.

---

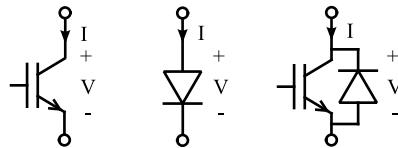
**Note** The fitting algorithm can handle only *single pulse* curves. Vendor datasheets sometimes also show heating curves for repeated pulses with different duty cycles; these curves cannot be used to calculate Foster coefficients with PLECS.

If the fitting algorithm repeatedly does not find a satisfactory solution, you may need to increase the number of Foster elements. Typically, three to five Foster elements should yield good results.

---

## 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**

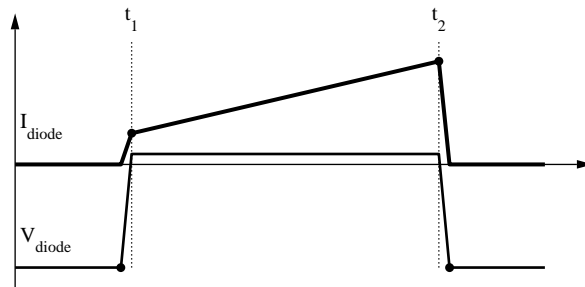
### Single Semiconductor Switch Losses

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.

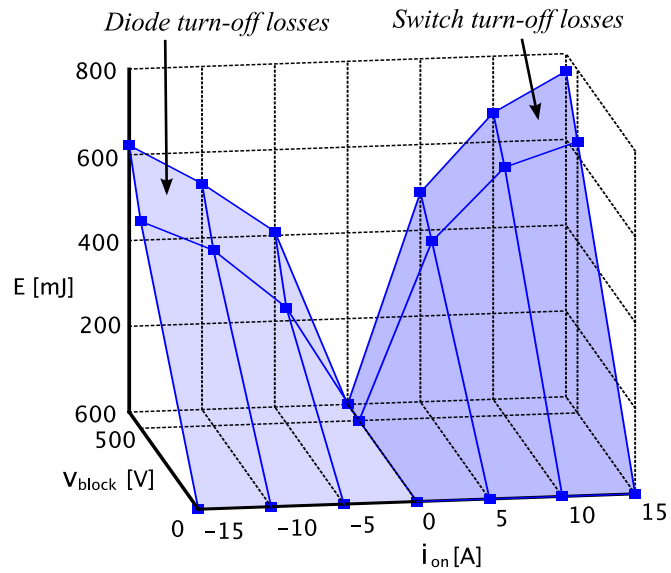


**Diode voltage and current during switching**

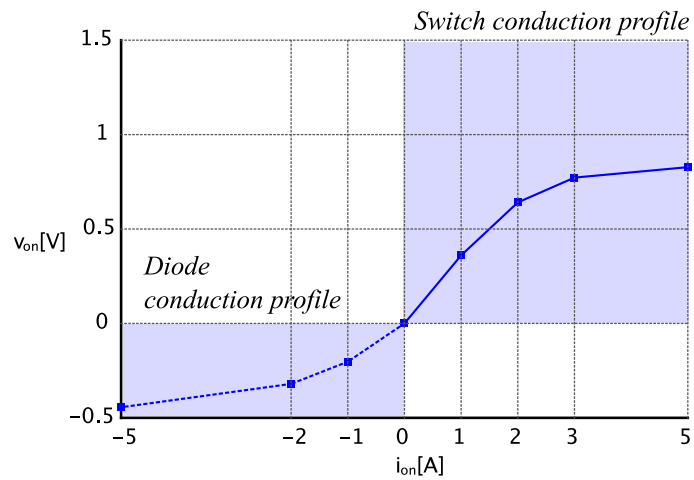
### Losses of Semiconductor Switch with Diode

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			
					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.

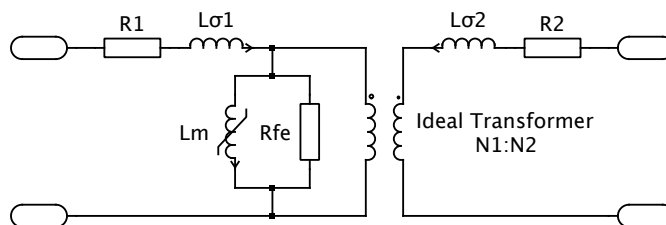
## Equivalent circuits for magnetic components

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  $\Phi$  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:

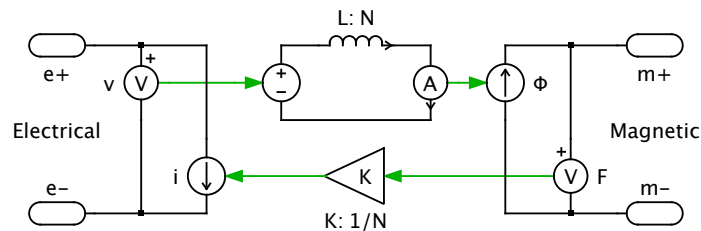
$$v = N \frac{d\Phi}{dt}$$

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

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,  $\mu_r$  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.

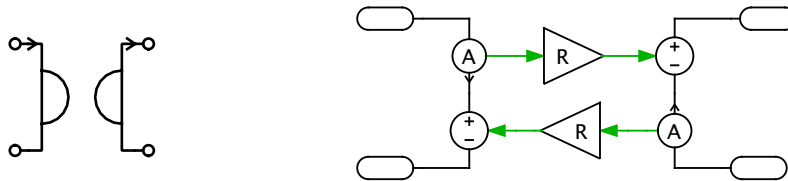
Windings form the interface between the electrical and the magnetic domain. A winding of  $N$  turns is described with the equations below. The left-hand

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.



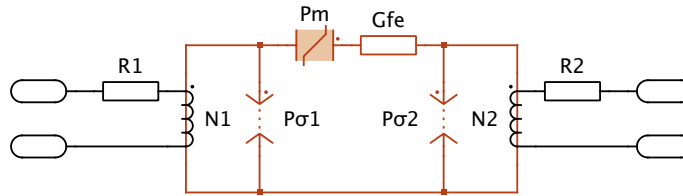
### Gyrator symbol and implementation

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,  $P_m$  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 Kirchoff'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

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

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  $B_{\text{sat}}$ ,  $a$  and  $\mu_{\text{sat}}$ . These coefficients can be found e.g. using a least-squares fitting procedure. Calculating the derivative 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. Buntentbach, "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. Buntentbach, "Analogies 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.

# Mechanical Modeling

One-dimensional mechanics describe the mechanical interaction between bodies that have exactly one degree of freedom. A translational body (or Mass) can *move along* a single axis, and a rotational body (or Inertia) can *rotate around* a single axis. With this limitation one-dimensional mechanical systems can be modeled similarly to electrical systems using simple analogies that are listed in the following table.

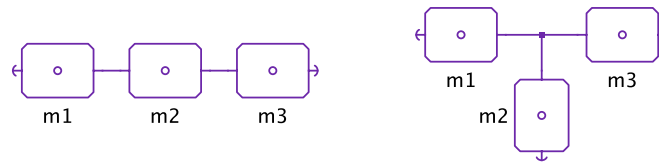
## Electrical and Mechanical Analogies

Electrical	Translational	Rotational
Voltage	Speed	Angular speed
Current	Force	Torque
Capacitor	Body (mass)	Body (moment of inertia)
Inductor	Spring	Spring
Resistor	Damper	Damper
Transformer	Lever	Gear
Switch	Clutch	Clutch

## Flanges and Connections

The two mechanical subdomains use separate connectors: a translational flange  $\tau$  and a rotational flange  $\tau$ . You can draw connections between flanges of the same type. By creating branch connections you can connect more than two flanges. Flanges that are connected to each other have the same displacement (i.e. position or angle), and the connection will exert whatever force is necessary in order to maintain this relationship.

Body components (i.e. the translational Mass and the rotational Inertia) have two rigidly connected flanges so that the two systems shown below are equivalent:



**Equivalent connections of three translational bodies**

## Force/Torque Flows and Sign Conventions

As the above table of electrical and mechanical analogies suggests, forces or torques acting on components are modeled as *flows* from one flange to another. The direction of a positive flow is indicated either with a dot next to a flange or with an arrow in the component icon.

Force and torque flows must be *balanced*, i.e. the sum of all flows towards a component must generally be zero, but there are two exceptions to this rule:

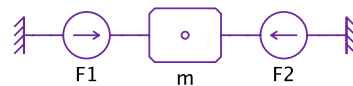
- Reference components only have a single flange so balancing is not possible for a single instance. Reference components in fact represent connectors of a single, global reference frame, and it is the net flow towards this reference frame that must be zero.
- Body components have an implicit internal connection to the global reference frame. A positive net flow towards a body causes the body to accelerate in the positive direction.



In this context it is important to note that the positive direction does not necessarily correlate with the graphical orientation of the components. For instance, the schematic shown below models the equation

$$F_1 + F_2 = m \cdot a$$

i.e. both forces accelerate the body in the positive direction, even though in the schematic the two forces might appear to oppose each other.



**Mass and two forces**

## Positions and Angles

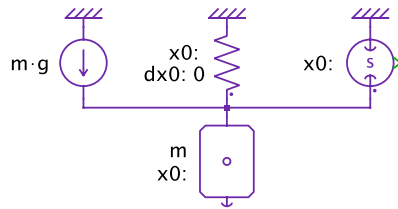
In contrast to other modeling environments, PLECS does not generally use flange displacements as state variables in the component equations in order to avoid having to solve Index-2 problems. Instead, absolute or relative displacements are only calculated when required e.g. in a hard-stop component or if you explicitly measure them using a sensor. The displacements are then calculated by integrating the corresponding absolute or relative speed.

### Initial Conditions

As with all integrators, displacement meters must be provided with proper initial values. PLECS allows you to specify these initial values directly in the components that require them or indirectly via neighboring components. For this purpose, most components have an initial displacement parameter that defaults to an empty string, which means "don't care" or "don't know".

At simulation start, PLECS will automatically calculate required but unknown initial values from the values that you have provided. An error will be flagged if you do not supply enough data to determine required initial values. On the other hand, an error will also be flagged, if you provide *too much* and *inconsistent* data.

The example shown below models a body with mass  $m$  that is subject to a gravitational force  $m \cdot g$  and suspended from a spring. The spring is initially unstretched ( $dx_0 = 0$ ) but its equilibrium displacement  $x_0$  is not specified.



### Spring and mass

If the model is run as is, PLECS will flag an error because it does not have enough data to calculate this equilibrium length and the initial value of the position sensor. To fix this, you can specify any *one* of the following three parameters:

- 1 the initial value  $x_0$  of the position sensor
- 2 the initial position  $x_0$  of the body
- 3 the equilibrium displacement  $x_0$  of the spring

Note that you may specify more than one of the above values, but if you do so, the settings must be consistent.

### Angle Wrapping

When calculating angles by integrating angular speed, care must be taken to avoid numerical problems during longer simulations. For this reason, PLECS automatically wraps the integral in the interval between  $-\pi$  and  $+\pi$  when you measure an *absolute* angle with a position sensor that has one flange internally connected to the rotational reference frame. Note that *relative* angles – measured with a position sensor that has two accessible flanges – are *not* wrapped because you can wind a torsion spring by more than one turn.

## Ideal Clutches

Analogous to its ideal electrical switches, PLECS features ideal mechanical clutches that engage and disengage instantaneously. While engaged they make an ideal rigid connection between their flanges and while disengaged they transmit zero force (or torque).

## Inelastic Collisions

PLECS permits you to connect an ideal clutch between two bodies and engage the clutch while they move (or rotate) at different speeds. PLECS models such an event as a perfectly inelastic collision and calculates the common speed after the collision based on the conservation law of angular momentum so that e.g.

$$\omega^+ = \frac{J_1\omega_1^- + J_2\omega_2^-}{J_1 + J_2}$$

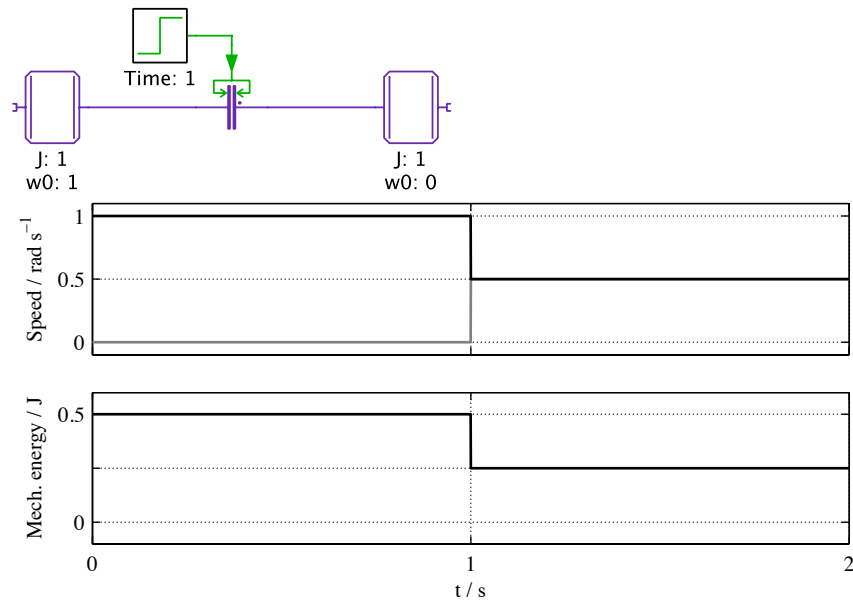
where  $J_1$  and  $J_2$  are the moments of inertia of the two bodies,  $\omega_1^-$  and  $\omega_2^-$  are the two angular speeds prior to the collision and  $\omega^+$  is the common angular speed after the collision.

It is important to note that kinetic energy is lost during an inelastic collision even though the clutch is ideal and lossless. Assuming for simplicity that  $J_1 = J_2 = J$  so that  $\omega^+ = \frac{1}{2}(\omega_1^- + \omega_2^-)$ , the kinetic energy of the system before and after the collision is for example

$$\begin{aligned} E^- &= \frac{1}{2}J(\omega_1^{-2} + \omega_2^{-2}) \\ E^+ &= \frac{1}{2}(2J)\omega^{+2} \\ &= \frac{1}{4}J(\omega_1^- + \omega_2^-)^2 \\ \Rightarrow E^- - E^+ &= \frac{1}{4}J(\omega_1^- - \omega_2^-)^2 \end{aligned}$$

This is demonstrated using the simple example shown below consisting of two bodies with the same inertia  $J = 1 \frac{\text{kg}\cdot\text{m}^2}{\text{rad}^2}$ . One initially rotates with  $\omega_1^- = 1 \frac{\text{rad}}{\text{s}}$  while the other is stationary  $\omega_2^- = 0$ . There is no friction or external torque acting on the bodies. When the clutch engages at  $t = 1\text{s}$ , the two bodies immediately rotate with the same speed  $\omega^+ = .5 \frac{\text{rad}}{\text{s}}$  and the total kinetic energy of the system reduces instantaneously from  $.5J$  to  $.25J$ .

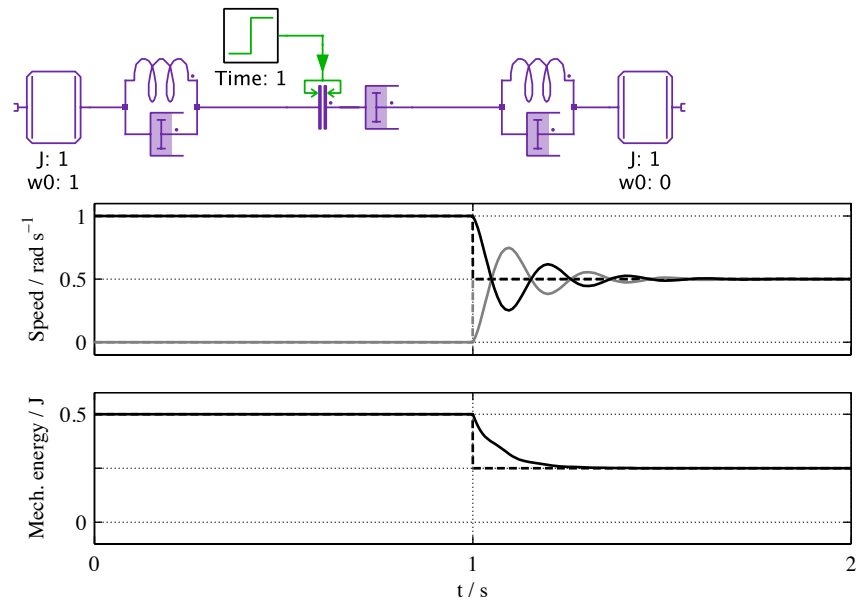
It is interesting to compare this response with that of a more detailed model, in which the clutch is modeled with a finite damping coefficient when engaged. Additionally the two shafts connecting the bodies with the clutch are



### Inelastic collision with ideal clutch

assumed to have a certain elasticity and damping coefficient. The corresponding schematic and plots are shown below; for comparison the response from the idealized model is superimposed with dashed lines.

The damping coefficients and spring constants have been exaggerated so that there is visible swinging. Note however, that after the transients have settled, the two bodies rotate at the same common speed as in the idealized model. Likewise, the final mechanical energy stored in the system is the same as in the idealized model.



**Inelastic collision with non-ideal clutch and elastic shafts**



# 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  $t_{\text{start}}$  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  $rtol$ :

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

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).



## Non-periodic Case

If the operating point of the system is defined as non-periodic (DC), a variant of the algorithm described above is performed. As in the periodic case, Newton iterations are executed to find the steady-state. Here, the algorithm searches for the roots of the function

$$f(\mathbf{x}) = \dot{\mathbf{x}}$$

i.e. the time derivative of the vector of state variables  $\mathbf{x}$ . Since no simulation has to be performed to compute  $f$ , the full Jacobian  $\mathbf{J}$  is calculated in each iteration. The convergence criterion remains the same as for the periodic case.

## Limitations

### Hidden state variables

In 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.

---

In PLECS Standalone, the operating point can be defined as “non-periodic” (DC). In this case, no sweep is executed, but the Bode plot is computed directly from the state space matrices at the steady-state.

## 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

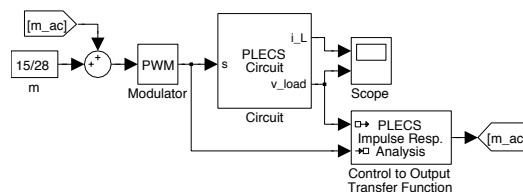
The impulse response analysis is performed in three steps:

- 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.

## Multitone Analysis

The Multitone Analysis is similar to an AC Analysis. Again the response of the system to a small perturbation signal is analysed. However, instead of multiple sinusoidal signals of different frequencies, only one multitone signal is applied. It is composed of several sinusoidal signals and therefore contains all investigated frequencies at once.

The multitone signal is computed as

$$u(t) = \sqrt{\frac{2}{N}} \sum_{k=1}^N \sin\left(2\pi k f_b t + \frac{\pi(k-1)^2}{N}\right),$$

where  $N$  is the number of tones and  $f_b$  the base frequency. In PLECS, the user can control the amplitude of the perturbation signal by a factor that is multiplied to  $u(t)$ .

### Algorithm

The simulation is divided into two phases. It is assumed that the system reaches its steady-state in the first phase of duration  $T_i$ . In the second phase of duration  $T_b = 1/f_b$ , the response of the system is recorded for the Fourier analysis.

The Multitone Analysis performs these steps:

- 1** Perform an unperturbed simulation of length  $T_i + T_b$ . Record the system response during  $T_b$  in  $y_0$ .
- 2** Perform a simulation of the same length and perturbed by  $u$ . Record the system response during  $T_b$  in  $y$ .
- 3** Compute the Fourier transforms  $U$  of  $u$  and  $Y$  of  $y - y_0$ .
- 4** Compute the transfer function as  $G = Y/U$ .

## Remarks

The Multitone Analysis is faster than the AC Analysis because it only needs to compute the response to one signal instead of a set of signals for each frequency. On the other hand, the analysed frequencies are restricted to multiples of the base frequency. Since the Multitone Analysis does not use the Steady-State Analysis, it still works in cases where the Steady-State Analysis fails, provided  $T_i$  is large enough.

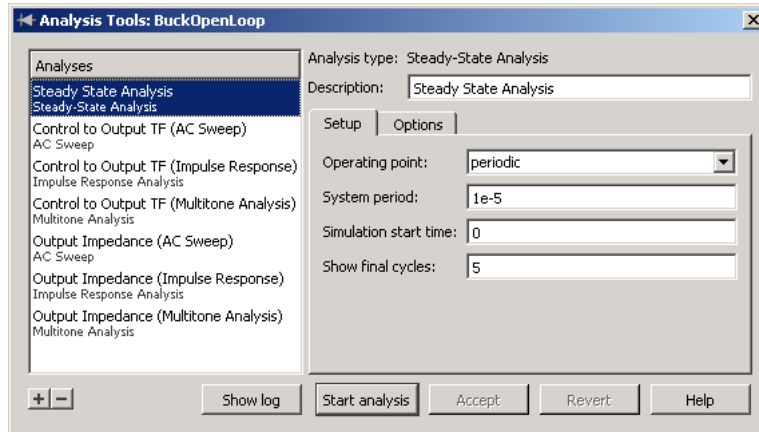
Note that the lengths of the  $y_0$  and  $y$  vectors are different in general. To compute the difference  $y - y_0$ , the missing values are linearly interpolated.

## References

- S. Boyd, "Multitone signals with low crest factor", IEEE Transactions of Circuits and Systems, Vol. CAS-33, No. 10, 1986.
- C. Fernández, P. Zumel, A. Fernández-Herrero, M. Sanz, A. Lázaro, A. Barro, "Frequency response of switching DC/DC converters from a single simulation in the time domain", Applied Power Electronics Conference and Exposition (APEC), 2011 Twenty-Sixth Annual IEEE , March 2011.

## 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

### Operating point

This parameter defines whether the operating point of the system is periodic or non-periodic (DC). If it is periodic, the system period can be speci-

fied using the next parameter.

**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. A setting of 0 is equivalent to defining the system as non-periodic. When set to auto, which is the default, PLECS will try to determine the system period automatically.

**Simulation start time**

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

**Show final cycles / timespan**

The number of steady-state cycles for which a transient simulation is run at the end of an analysis. Or, if the simulation is non-periodic, the duration of the final transient simulation. 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 589) and a Small Signal Response (see page 590) block in order to define the points at which the perturbation is injected and the response is measured. The Small Signal Gain (see page 588) 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.

### Operating point

This parameter defines whether the operating point of the system is periodic or non-periodic (DC). If it is periodic, the system period can be specified using the next parameter.

### 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. A system period of 0 is equivalent to defining the system as non-periodic. When set to auto, which is the default, PLECS will try to determine the system period automatically.

### 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  $t_{\text{start}}$  to be used in the transient simulation runs. Simulations run from  $t_{\text{start}}$  to  $t_{\text{start}} + T$ , 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 168).

## Impulse Response Analysis

In order to perform an impulse response analysis, you need to insert a Small Signal Perturbation (see page 589) and a Small Signal Response (see page 590) 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 170). 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 to obtain smooth curves.

## Multitone Analysis

In order to perform a multitone analysis, you need to insert a Small Signal Perturbation (see page 589) and a Small Signal Response (see page 590) 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.

**Initial simulation period**

The duration of an initial simulation performed before the response is measured. It is assumed that during this period, the system reaches its

steady state. The total simulation duration will be the sum of this parameter and one period of the base frequency signal.

**Frequency range**

A vector containing the lowest and highest frequency of the multitone perturbation signal. The highest frequency is rounded up towards the next integer multiple of the lowest frequency.

In a multitone analysis, the frequencies are linearly spaced (see “Multitone Analysis” (on page 166)). Since the Bode plot has a logarithmic scale, PLECS thins out the higher frequency values to accelerate the analysis.

If the lowest and highest frequencies are far apart, i.e. separated by several orders of magnitude, the multitone analysis may become slow. One reason is that the simulation times become long and the simulation steps small, since they depend on the lowest and highest frequencies, respectively. You may try to speed up the analysis by specifying *intermediate* frequency values, i.e. by entering a frequency vector with more than two elements. Each intermediate value must be greater than ten times the preceding value. If the frequency vector contains  $n$  elements,  $n - 1$  separate multitone analyses are performed and the Bode plot is composed of the respective results. Since the frequency range of each individual multitone analysis is smaller than the overall range, the total time needed may become shorter.

**Amplitude**

The amplitude of the perturbation signal. Note that the actual perturbation signal may have a slightly different amplitude due to its composition from the different tones.

**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.

**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 239) 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*.

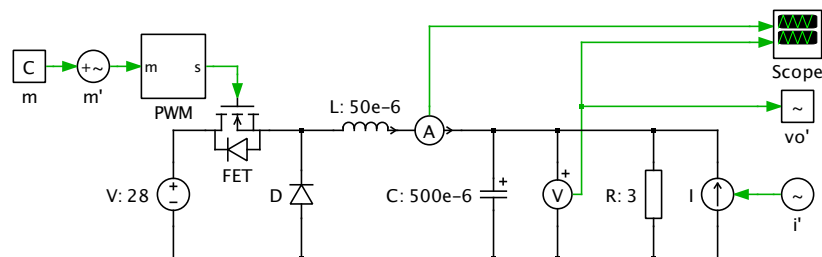
```
t = plecs('get', circuit, 'Topology');
```

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 247) 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  $v_o'$ .

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**.

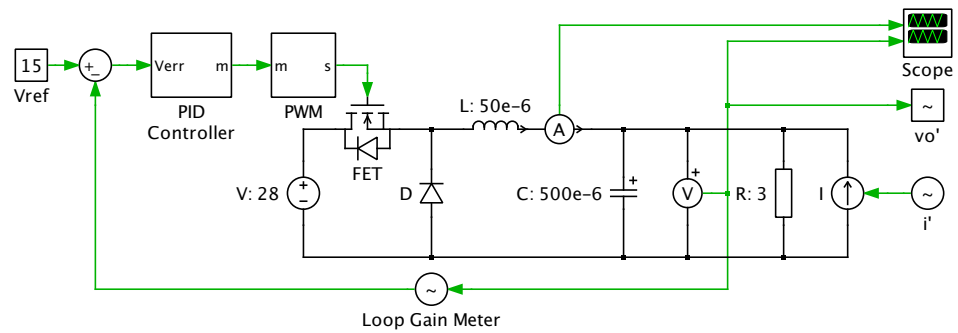
## Output Impedance

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 PLECS Blockset

In 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 739) 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 <b>System period length/Max simulation time span</b> field.
TStart	Simulation start time. Set by the <b>Simulation start time</b> field.

**Steady-State Analysis Parameters (contd.)**

<b>Parameter</b>	<b>Description</b>
Tolerance	Relative error tolerance used in the convergence criterion. Set by the <b>Termination tolerance</b> field.
MaxIter	Maximum number of iterations allowed. Set by the <b>Max number of iterations</b> field.
Display	Specifies the level of detail of the diagnostic messages displayed in the command window (iteration, final, off). Set by the <b>Display</b> 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 <b>Hidden model states</b> 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 <b>Steady-state variable</b> field.
NCycles	Number of steady-state cycles that should be simulated at the end of an analysis. Set by the <b>Show steady-state cycles</b> 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 728) 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 (AC Sweep) block (see page 734) 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**

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

**AC Analysis Parameters (contd.)**

<b>Parameter</b>	<b>Description</b>
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 <b>Hidden model states</b> drop-down list.
OutputName	Name of a MATLAB variable used to store the transfer function at the end of an analysis. Set by the <b>Output variable</b> field.
BodePlot	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the <b>Plot Bode diagram</b> 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 732) 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 <b>System period length</b> field.
TStart	Simulation start time. Set by the <b>Simulation start time</b> field.
FreqRange	Range of the perturbation frequencies. Set by the <b>Frequency sweep range</b> field.

**Impulse Response Analysis Parameters (contd.)**

<b>Parameter</b>	<b>Description</b>
FreqScale	Specifies whether the sweep frequencies should be distributed on a linear or logarithmic scale. Set by the <b>Frequency sweep scale</b> field.
NPoints	Number of data points generated. Set by the <b>Number of points</b> field.
Perturbation	Perturbation amplitude of the discrete impulse. Set by the <b>Perturbation</b> field.
Compensation	Specifies whether and how the effect of the sampling should be compensated (none, discrete pulse, external reference). Set by the <b>Compensation for discrete pulse</b> drop-down list.
Tolerance	Relative error tolerance used in the convergence criterion of the initial steady-state analysis. Set by the <b>Termination tolerance</b> field.
MaxIter	Maximum number of iterations allowed during the initial steady-state analysis. Set by the <b>Max number of iterations</b> field.
Display	Specifies the level of detail of the diagnostic messages displayed in the command window (iteration, final, off). Set by the <b>Display</b> 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 <b>Hidden model states</b> drop-down list.
OutputName	Name of a MATLAB variable used to store the transfer function at the end of an analysis. Set by the <b>Output variable</b> field.

**Impulse Response Analysis Parameters (contd.)**

<b>Parameter</b>	<b>Description</b>
BodePlot	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the <b>Plot Bode diagram</b> 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.

**Multitone / Loop Gain Analysis**

To perform a multitone analysis, copy the Multitone Analysis block (see page 737) 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 (Multitone) block (see page 735) 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

```
plmultitone(block);
```

where *block* is the Simulink handle or the full block path of the Multitone Analysis 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 Multitone Analysis and Loop Gain Analysis blocks. The **Parameter** column shows the parameter names to be used with the `plcsweep` 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.

### Multitone Analysis Parameters

Parameter	Description
FreqRange	Range of the perturbation frequencies. Set by the <b>Frequency sweep range</b> field.
Amplitude	Amplitude of the perturbation signal. Set by the <b>Amplitude</b> field.
TStart	Simulation start time. Set by the <b>Simulation start time</b> field.
Display	Specifies the level of detail of the diagnostic messages displayed in the command window (iteration, final, off). Set by the <b>Display</b> drop-down list.
HideScopes	Hide all Simulink scope windows during an analysis in order to save time.
OutputName	Name of a MATLAB variable used to store the transfer function at the end of an analysis. Set by the <b>Output variable</b> field.
BodePlot	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the <b>Plot Bode diagram</b> drop-down list.

### 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);
```

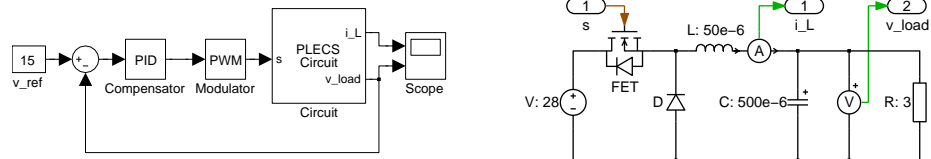
sets the vector of switch positions for the subsequent analysis to *switchpos*.

```
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 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.

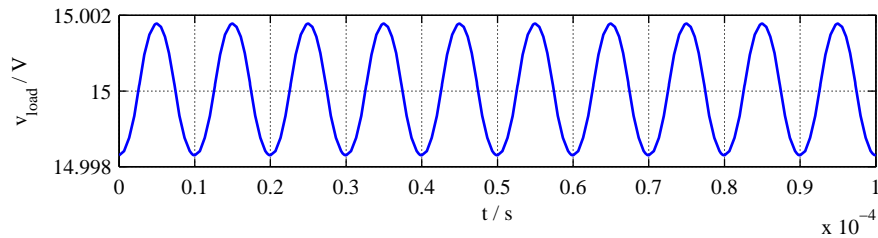


The examples used in this section follow the design example in [Erickson], chapter 9. They have been implemented in the demo models `p1BuckSweep`, `p1BuckImpulseResponse` and `p1BuckLoop`.

## 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  $V_{\text{out}}/V_{\text{src}} = 15\text{V}/28\text{V}$ . 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.



### Steady-state output voltage

### 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 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.

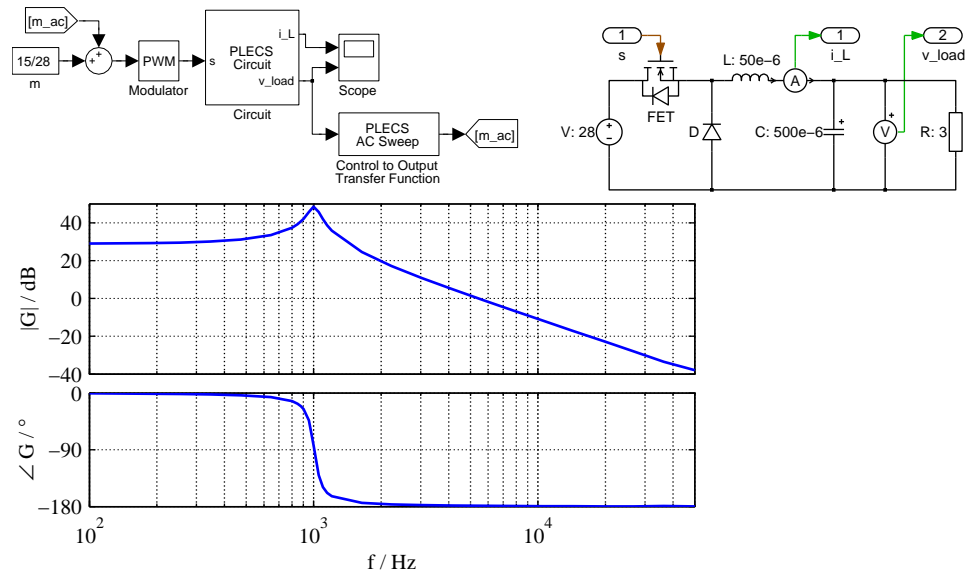
The initial amplitude of the perturbation is set to  $1e-3$  which is approx.  $2/1000$  of the duty cycle. We want to sweep a frequency range between 100Hz and 50kHz with a few extra points between 800Hz and 1200Hz. This is achieved by setting the parameter to `[100 800:50:1200 50000]`. As expected, 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 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  $1e-2$ .

### 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





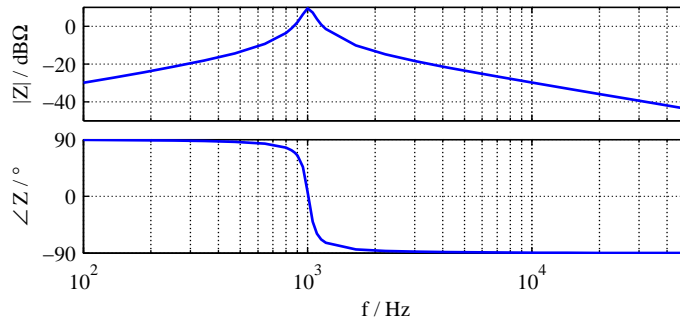
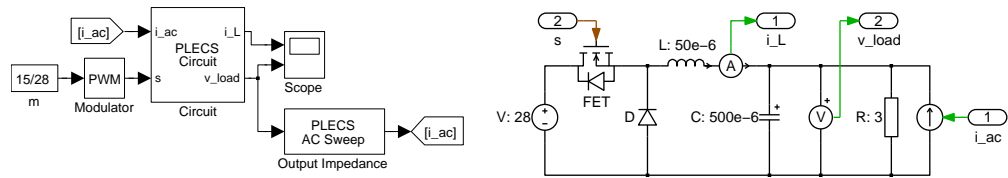
### Open-loop control-to-output transfer function

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.

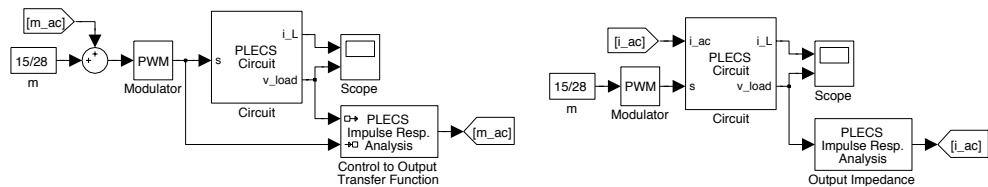
### Loop Gain Analysis

**Compensator settings** The compensator should attain a crossover frequency of  $f_c = 5$  kHz. At this frequency the open-loop control-to-output transfer function has a phase of nearly  $-180^\circ$ . It should be lifted by  $52^\circ$  to get a peak overshoot of 16%. This is achieved using a PD compensator with a zero at  $f_z = 1.7$  kHz, a pole at  $f_p = 14.5$  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  $f_z = 500$  Hz is added.

The compensator is implemented as shown above. The compensator output is limited to 0.1...0.9. In order to prevent windup problems during the steady-state analysis the integrator is limited to the same range.



### Open-loop output impedance



### Using the Impulse Response Analysis block

**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.

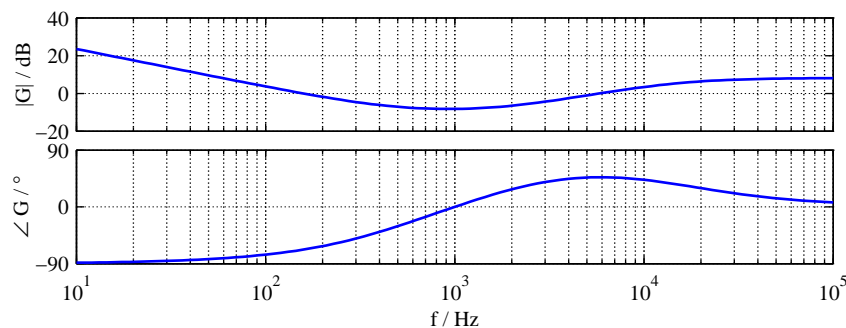
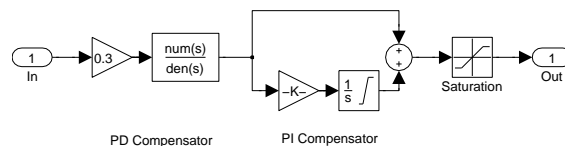
## State-Space Averaging

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 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  $\bar{\mathbf{A}}$ ,  $\bar{\mathbf{B}}$ ,  $\bar{\mathbf{C}}$



### PID compensator and transfer function

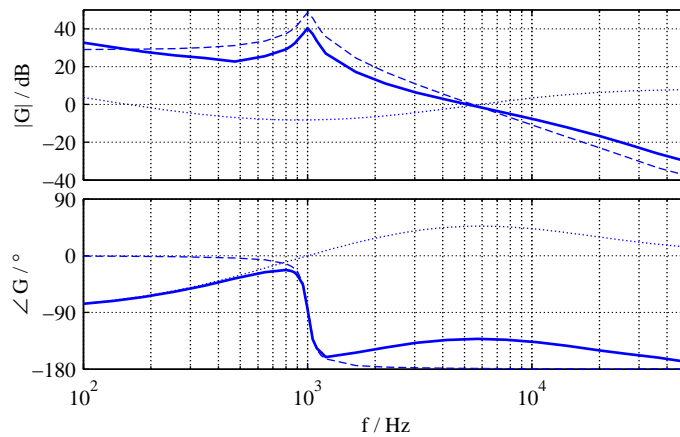
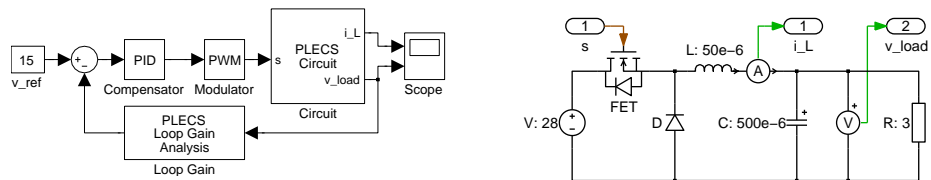
and  $\bar{D}$  are defined as

$$\begin{aligned}\bar{A} &= \bar{m}A_1 + (1 - \bar{m})A_2 \\ \bar{B} &= \bar{m}B_1 + (1 - \bar{m})B_2 \\ \bar{C} &= \bar{m}C_1 + (1 - \bar{m})C_2 \\ \bar{D} &= \bar{m}D_1 + (1 - \bar{m})D_2\end{aligned}$$

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  $A_1$ ,  $A_2$  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('plBuckSweep');
```



**Closed-loop gain**

```

names = plecs('get', 'plBuckSweep/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:

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

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'

```

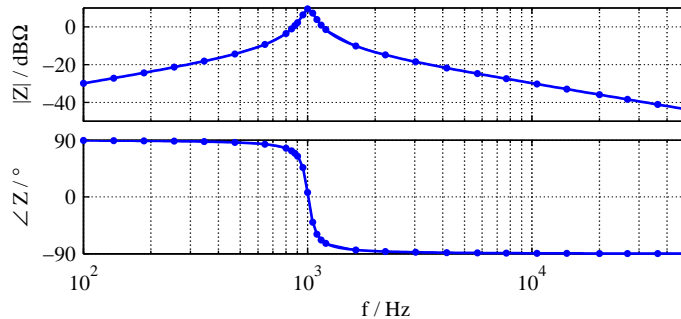
```
names.Outputs
```

```
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.

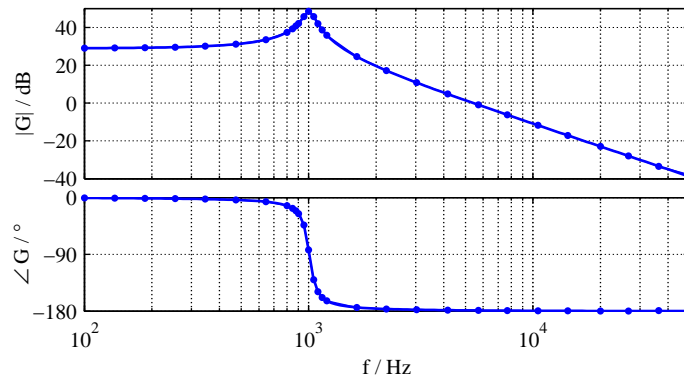
This leads to the following program code:

```
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 non-linear 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 31).

### 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 `InputSignal(i, j)`, `OutputSignal(i, j)` 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 32.)

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) = InputSignal(0, 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 32.) 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 (InputSignal(0, 0) >= 1.)
    OutputSignal(0, 0) = 1.;
  else
    OutputSignal(0, 0) = 0.;
}

ZCSignal(0) = InputSignal(0, 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, PLECS Blockset can control the time steps taken by the Simulink solvers only indirectly by using an internal zero-crossing signal. Therefore, 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
}
```

To access the sample times during execution of the C-Script, use the macros `SampleTimePeriod(i)` and `SampleTimeOffset(i)`. In the case of inherited sample times, the actual resolved values are returned, not [-1, 0] (see “Sample Times” on page 36).

## 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, 3d-arrays or strings.

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. In the case of strings, 1 and the length of the string measured in C characters (`char`) is returned, respectively. Note that because the strings are UTF-8 encoded, the length returned by `ParamDim(i, j)` may be larger than the number of unicode characters in the string.

To access the actual parameter values, use the macro `ParamRealData(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 eIdx = rowIdx + numRows*(colIdx + numCols*pageIdx);
double value = ParamRealData(0, eIdx);
```

To access string parameters, use the macro `ParamStringData(i)`. For example, to use the second parameter as an error message, you may write:

```
SetErrorMessage(ParamStringData(1));
```

## 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

```
OutputSignal(0, 0) = 2.*InputSignal(0, 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

```
OutputSignal(0, 0) = DiscState(0);
```

### Update Function Code

```
DiscState(0) = InputSignal(0, 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

```
OutputSignal(0, 0) = ContState(0);
```

### Derivative Function Code

```
ContDeriv(0) = InputSignal(0, 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.  $2\pi$  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;

OutputSignal(0, 0) = ContState(0);

```

In every major time step, if the integrator state has gone beyond the upper or lower boundary,  $2\pi$  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.  $2\pi$ . 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) = InputSignal(0, 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 198).

**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 (InputSignal(0, 0) > 1.)
        mode = UPPER_LIMIT;
    else if (InputSignal(0, 0) < -1.)
        mode = LOWER_LIMIT;
    else
        mode = NO_LIMIT;
}

switch (mode)
{
    case NO_LIMIT:
        OutputSignal(0, 0) = InputSignal(0, 0);
        break;
    case UPPER_LIMIT:
        OutputSignal(0, 0) = 1.;
        break;
    case LOWER_LIMIT:
        OutputSignal(0, 0) = -1.;
        break;
}

ZCSignal(0) = InputSignal(0, 0) + 1.;
ZCSignal(1) = InputSignal(0, 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 (InputSignal(0, 0) == 0)
{
    OutputSignal(0, 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))
{
    OutputSignal(0, 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 = InputSignal(0, 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
<code>NumInputTerminals</code>	int	R	Returns the number of input terminals.
<code>NumOutputTerminals</code>	int	R	Returns the number of output terminals.
<code>NumInputSignals(int i)</code>	int	R	Returns the number of elements (i.e. the width) of the signal connected to the <i>i</i> th input terminal.
<code>NumOutputSignals(int i)</code>	int	R	Returns the number of elements (i.e. the width) of the signal connected to the <i>i</i> th output terminal.



**C-Script Data Access Macros (contd.)**

<b>Macro</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
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).
NumSampleTime	int	R	Returns the number of sample times.
SampleTimePeriod(int i)	int	R	Returns the period of the <i>i</i> th sample time.
SampleTimeOffset(int i)	int	R	Returns the offset of the <i>i</i> th sample time.
IsMajorStep	int	R	Returns 1 during major time steps, else 0.
IsSampleHit(int i)	int	R	Returns 1 if the <i>i</i> th 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.
InputSignal(int i, j)	double	R	Returns the value of the <i>j</i> th element of the <i>i</i> th input signal.
OutputSignal(int i, j)	double	R/W	Provides access to the value of the <i>j</i> th element of the <i>i</i> th 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 <i>i</i> th continuous state. Continuous state variables may <i>not</i> be changed during minor time steps.

**C-Script Data Access Macros (contd.)**

<b>Macro</b>	<b>Type</b>	<b>Access</b>	<b>Description</b>
ContDeriv(int i)	double	R/W	Provides access to the derivative of the <i>i</i> th continuous state.
DiscState(int i)	double	R/W	Provides access to the value of the <i>i</i> th 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 <i>i</i> th zero-crossing signal.
ParamNumDims(int i)	int	R	Returns the number of dimensions of the <i>i</i> th user parameter.
ParamDim(int i, j)	int	R	Returns the <i>j</i> th dimension of the <i>i</i> th user parameter.
ParamRealData(int i, j)	double	R	Returns the value of the <i>j</i> th element of the <i>i</i> th user parameter. The index <i>j</i> 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. If the parameter is a string, this macro will produce a runtime error or an access violation if runtime checks are disabled.
ParamStringData(int i)	char*	R	Returns a pointer to a UTF-8 encoded, null-terminated C string that represents the <i>i</i> th user parameter. If the parameter is not a string, this macro will produce a runtime error or returns NULL if runtime checks are disabled.
SetErrorMessage(char *msg)	void	W	Use this macro to report errors that occur in your code. The simulation will be terminated after the current simulation step. In general, this macro should be followed by a return statement. The pointer <i>msg</i> must point to static memory.

**C-Script Data Access Macros (contd.)**

Macro	Type	Access	Description
SetWarningMessage(char *msg)	void	W	Use this macro to report warnings. The warning status is reset as soon as the current C-Script function returns, so you do not need to reset it manually. 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 = &OutputSignal(0, 0); // not recommended
output[1] = 1; // fails
*(output + 1) = 1; // fails
OutputSignal(0, 1) = 1; // ok
```

---

**Deprecated Macros**

The macros NumInputs, NumOutputs, Input(int i) and Output(int i) are deprecated but are still supported for C-Scripts that only have a single input and output terminal.



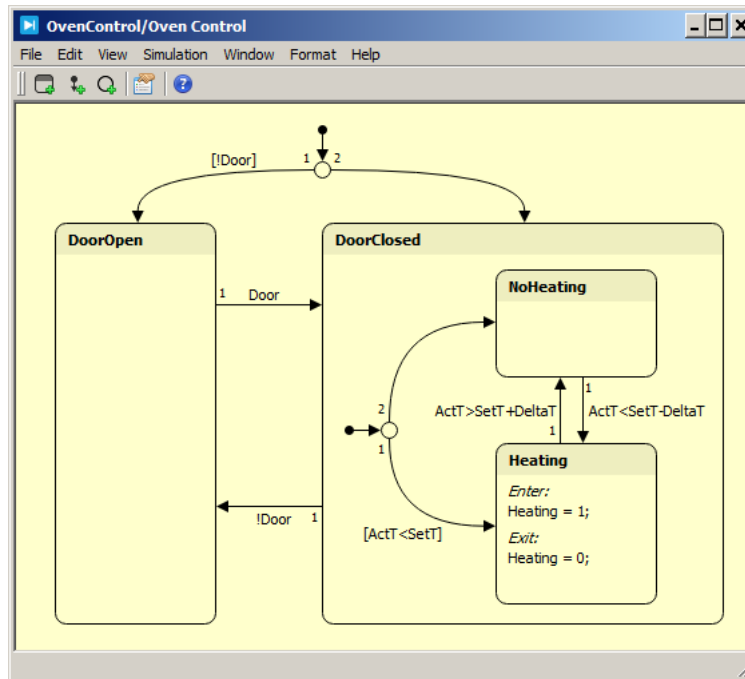
# State Machines

State machines are a formalism for event driven systems that move from one discrete state to another in response to discrete events. PLECS lets you graphically create and edit state machines using common concepts such as boxes for states and curved arrows for transitions, and simulate them together with a surrounding system. You can feed continuous or discrete signals into a state machine e.g. to react to external events and output discrete signals from a state machine e.g. as control signals. Actions are specified in the C programming language and can be associated with states and transitions. Thanks to their built-in timer events, state machines are equally useful for implementing supervisory controls and complex modulators.

This chapter is subdivided into three sections. The first section describes how you interact with the graphical editor to create and modify state machines. The second section describes the semantics of a state diagram and how they influence the execution of the state machine. The third section contains examples that highlight different features of the state machine.

## Working with State Machines

To create a new state machine, copy the State Machine block from the library browser into your model. A double-click on the block opens a new window with the state machine editor.



**State machine editor window**

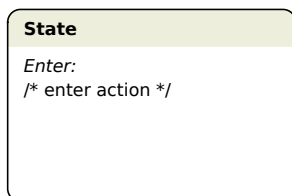
---

**Note** The State Machine uses a distinct editor. You cannot copy block diagram components to a state chart or state machine elements to a block diagram schematic.


---

## Working with States

A state is represented by a box with rounded corners. The name of the state is displayed at the top of the box in a title bar with a gray background.



### An empty state

To create a new state, click on the  button in the tool bar. Move the mouse anywhere on the chart and click the left mouse button to place the new state. To cancel the operation, press the **Escape** key, click the right mouse button, or click anywhere outside the editor window.

To duplicate an existing state, hold down the **Ctrl** key (**cmd** key on macOS), then click on the title bar of the state and drag the mouse to a new location in the same or a different editor window.

To change the size of a state, move the mouse over one of the four corners so that the pointer shape becomes a diagonal arrow. Hold down the mouse button, drag the mouse until the dashed box has the desired size and release the mouse button.

To change the name of a state, double-click on the name and edit it on the chart. Valid state names must start with a letter and can contain only letters and numbers. Whitespace, dashes or underscores are not allowed.

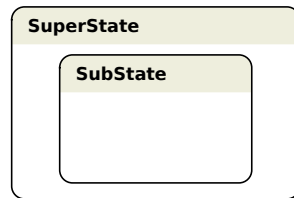
To edit the actions associated with the state, double-click on a free area within the state. This will open a tabbed code editor, in which you can edit the **Enter**, **During** and **Exit** actions for the state. If there is enough space, the action code is also displayed on the state. By double-clicking on the code, you can edit it directly on the chart.

### Hierarchical States

A state can contain other states. The containing state is called a super-state or *compound state*, the contained state, a sub-state. A state that does not con-

tain other states is called a *leaf state*. Compound states do not have a **During** action.

To create hierarchical states, resize a state so that it is large enough to fully surround another state, then move the other state into the first state.



#### A super-state containing a sub-state

---

**Note** Overlapping states are forbidden and will produce an error message at simulation start.

---

## Working with Transitions

Transitions are represented by curved arrows from one state to another (or also the same) state. To create a new transition, move the mouse over one of the edges so that the pointer shape changes to crosshairs. Hold down the mouse button, drag the mouse to another edge until the pointer shape changes to double crosshairs and release the mouse button. If you release the mouse button before the pointer shape has changed to double crosshairs, the operation is cancelled.

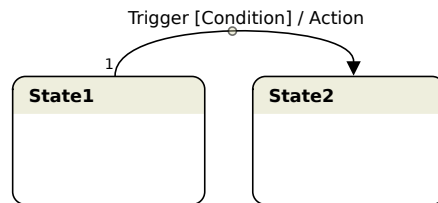
To change the start point or end point of an existing transition, move the mouse pointer over either end until the pointer shape changes to an open hand, then hold down the mouse button and drag the mouse to the new location until the pointer shape changes to a double crosshair. If you release the mouse button before the pointer shape has changed to double crosshairs, the operation is cancelled and the transition remains unchanged.

Depending on its shape, a transition may have one or more control handles that become visible when you hover the mouse over the transition. To change



the shape of the transition, hold down the mouse button over a control handle and drag it to the desired location.


A double-click on a transition opens the Transition Editor. It allows you to edit the **Priority**, **Trigger**, **Condition** and **Action** of the transition. The priority is also displayed at the origin of the transition arrow, and a double-click lets you edit it directly on the chart. Trigger, condition and action combined form the transition label in the form *Trigger [Condition] / Action*, which is shown next to the transition arrow. To change the location of the label along the transition, hold down the mouse button over the transition and drag it to the desired location. By double-clicking on the label you can edit it directly on the chart.




### A transition with labels and control handle

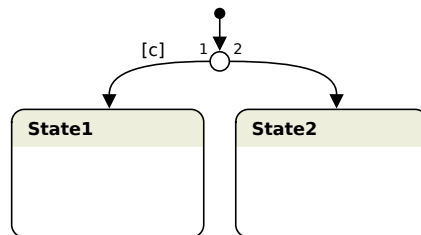
## Default Transitions

A default transition is represented by a curved arrow originating from a black dot. It is required on the top level of a state machine to define which state shall become active at the beginning of a simulation. If a compound state is the direct target of any transitions, it also requires an internal default transition to define which of its sub-states shall become active when an incoming transition is activated.

To create a new default transition, click on the  button in the tool bar. Move the mouse anywhere on the chart and click the left mouse button to place the origin of the default transition (marked with a dot). Then, move the mouse to draw the transition to an edge of the desired target state and click the left mouse button again. To cancel the operation, press the **Escape** key, click the right mouse button, or click anywhere on the chart that is not the edge of a state.

## Working with Junctions

Junctions are branching points that join or fork transitions. They are represented by circles. Junctions are useful e.g. if the state that shall become active at the beginning of a simulation depends on one or more conditions. To create a new junction, click on the  button in the tool bar.




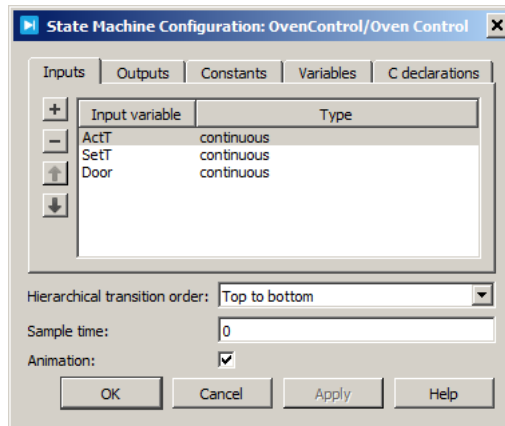
**A junction forking a default transition**

## Working with Annotations

Annotations are text blocks that you can place freely on a chart for documentation purposes. They have no influence on the execution of the state machine. To create a new annotation, double-click on an empty space in the chart and start typing. To move an annotation, hold down the mouse button over the annotation and drag it to the desired location. To edit an annotation, double-click on it. Choose **Text alignment** from the **Format** menu to change the text alignment of the annotation.

## State Machine Configuration

To open the configuration editor, click on the  button in the tool bar.



**State machine configuration editor**

### Input Signals

On the **Inputs** tab you define input signals that are fed from the surrounding system into the state machine. Input signals are specified by an **Input variable** and a **Type**.

The **Input variable** specifies the name with which you refer to the input signal in actions or expressions. It is also displayed next to the corresponding input terminal of the State Machine block. The variable name must be unique and a valid C identifier, i.e. it must start with a letter and consist of only letters, numbers and underscores.

The **Type** specifies whether the signal is a continuous signal or a trigger signal. For continuous and trigger signals, the actual signal value is assigned to the input variable in every time step. For trigger signals, additionally an *event* with the same name will be created when the input signal changes in the prescribed way. A rising trigger event is created in the instant when the input signal changes from zero to non-zero, a falling trigger event is created in the instant when the input signal changes from non-zero to zero. For trigger signals, a trigger symbol is also displayed next to the corresponding input terminal of the State Machine block.

---

**Note** Signal changes from one non-zero value to another non-zero value, e.g. from a negative value to a positive value, *do not* cause an event to be created.

---

Input signals appear on the State Machine block in the order in which they appear in the list. Use the four buttons to the left of the list to add or remove inputs or to change their order.

### Output Signals

On the **Outputs** tab you define output signals that are fed from the state machine to the surrounding system. Output signals are defined by an **Output variable** that specifies the name with which you refer to the output signal in actions. It is also displayed next to the corresponding output terminal on the State Machine block. The variable name must be unique and a valid C identifier, i.e. it must start with a letter and consist of only letters, numbers and underscores.

Output signals appear on the State Machine block in the order in which they appear in the list. Use the four buttons to the left of the list to add or remove outputs or to change their order.

### Constants and Variables

On these tabs you define global variables that you can use in actions or expressions. They must have a unique and valid C identifier, i.e. they must start with a letter and consist of only letters, numbers and underscores.

**Constants** remain constant during a simulation. The value is determined by the **Value** expression, which can be any valid MATLAB or Octave expression that evaluates to a scalar number.

**Variables** can be modified by actions. They are in fact additional discrete state variables in addition to the active state of the state machine. Their initial value is determined by the **Initial value** expression, which can be any valid MATLAB or Octave expression that evaluates to a scalar number.

## C Declarations

The **C declarations** tab is used for global declarations and definitions. It is also the place to include standard library headers and to define macros and static helper functions that you want to use in actions and expressions.

In addition to the user defined variables and functions, the following pre-defined macros can be used in actions:

### Predefined Action Macros

Macro	Type	Access	Description
currentTime	double	R	Returns the current simulation time.
SetErrorMessage(char *msg)	void	W	Use this macro to report errors that occur in your code. The simulation will be terminated after the current simulation step. In general, this macro should be followed by a return statement. The pointer msg must point to static memory.
SetWarningMessage(char *msg)	void	W	Use this macro to report warnings. The warning status is reset after the execution of the current simulation step, so you do not need to reset it manually. The pointer msg must point to static memory.

---

**Note** User defined variable or function names must not start with fsm\_ or FSM\_ because these prefixes are reserved for internal symbols in the generated code.

---

### Hierarchical Transition Order

This option is only relevant for hierarchical state machines (see “Hierarchical States” on page 217 and “Execution of Hierarchical State Machines” on page 228). When both a super-state and its sub-states have outgoing transitions

that are eligible to be taken at the same time, this option determines whether transitions from the super-state take precedence over transitions from the sub-states or vice versa.

### Sample Time

This parameter determines how the state machine is executed. 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 36).

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

With a Continuous sample time, the state machine is executed at every simulation step. With a Discrete-Periodic sample time, the state machine is executed only at the regularly spaced simulation steps prescribed by the sample time values. With an Inherited sample time, the actual sample time depends on the blocks that are connected to the State Machine block.

### Animation

This option is useful for debugging your state machine. When the option is checked, the simulation is paused whenever a transition fires, and the transition path including the source and target state is highlighted. The simulation can be continued by selecting **Continue** from the **Simulation** menu or by pressing the **Space** key.

## State Machine Execution

A state machine is executed at each simulation step that the solver makes if it has a continuous sample time or at the specified time steps if it has a discrete sample time (see also “Sample Time” on page 224). During each execution, the following steps are performed:

- 1** The triggers and conditions of all transitions leaving the currently active state are evaluated.
- 2** If a transition “fires”, i.e. if both trigger and condition are true, the State Machine executes the **Exit** action of the current state followed by the transition action and the **Enter** action of the target state.
- 3** If *no* transition fires, the **During** action of the current state is executed.

During each execution, at most one state change can occur, i.e. at most one transition can be taken. Note that a transition may include one or more junctions (see “Compound Transitions” below).

### Transition Evaluation

A transition from an active state can be taken when the specified *trigger* event occurs, provided that the *condition* is true at the same time. Both trigger and condition are optional. If a transition does not specify a trigger, any event (including the mere execution of the state machine) will qualify so that the transition can be taken if the condition is true. If a transition does not specify a condition, it can be taken when the trigger event occurs.

### Transition Priorities

It is possible that multiple transitions from the current state are eligible to be taken at the same time. In this case, the transition with the *lower* priority number is given precedence.

### Compound Transitions

A compound transition is a complete path from one leaf state to another (or the same) leaf state consisting of two or more transitions joined by one or more junctions. A compound transition can only be taken when *all* trigger events specified by the individual transitions occur and *all* conditions specified

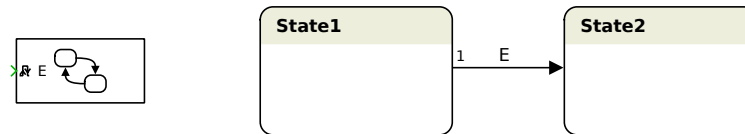
by the individual transitions are true. If a compound transition is taken, the actions of *all* individual transitions are executed in the order of the transitions on the path.

## Trigger Types

PLECS distinguishes between explicit, implicit and time-based triggers.

### Explicit Trigger

An explicit trigger is an input signal that is configured as a trigger signal. It is active whenever the input signal changes in the specified way. In the example shown below, the event **E** is active whenever the signal connected to the input terminal **E** changes from zero to a non-zero value or from a non-zero value to zero. If **State1** is active at this time, the transition fires.



### State machine with explicit trigger

---

**Note** It is expected that the input signal for an explicit trigger changes only at discrete instants. The signal source is responsible for registering an appropriate zero-crossing function to enable a variable-step solver to make a simulation step at the instant at which the signal reaches or leaves zero.

---

### Implicit Trigger

An implicit trigger is a relational expression. It is active when the expression *becomes* true, i.e. if it evaluated to false in the previous time step and evaluates to true in the current time step. If the state machine uses a continuous sample time, an implicit trigger will also register a zero-crossing function to



enable the solver to make a simulation step at precisely the instant at which the expression becomes true.

Notice the fundamental difference between an implicit trigger  $x > 0$  and a condition  $[x > 0]$  illustrated in the example below.



### Implicit trigger (left) versus condition (right)

In the left chart, if **State1** is active, the transition will be taken only when  $x$  becomes greater than 0. However, if  $x$  is already greater than 0 when **State1** becomes active, nothing will happen until  $x$  becomes less than or equal to 0 and afterwards greater than 0 again. If  $x$  is a continuous signal and the state machine uses a continuous sample time, the state machine will be executed precisely at the instant at which  $x$  crosses 0.

In the right chart, if **State1** is active, the transition will be taken at any execution of the state machine if  $x$  happens to be greater than 0. If  $x$  is already greater than 0 when **State1** becomes active, the transition will be taken during the next execution of the state machine. However, the execution of the state machine does not necessarily coincide with the instant at which  $x$  crosses 0.

### Time-Based Trigger

A time-based trigger is an expression of the form  $AFTER(delay)$ , where  $delay$  is an expression that evaluates to a number. If the state machine uses a continuous sample time, the trigger event will be created exactly  $delay$  seconds after the source state of the transition was entered. If the state machine uses a discrete sample time, the trigger event will be created at the first execution time following the delay period.

### Trigger Lifetime

A trigger event is only valid in the simulation step in which it is created. If no transition responds to the event in this time step, the event is ignored; it is *not* deferred to a subsequent execution of the state machine.

## Execution of Hierarchical State Machines

In a hierarchical state machine it is not immediately obvious which transitions are eligible to be taken, and which actions are executed when a transition is taken.

### Transition Evaluation

Only leaf states, i.e. states that do not contain other states, can be the *active* state. But compound states that directly or indirectly contain the active state are also *implicitly active*. When the state machine searches for a transition to be taken, it not only evaluates the transitions leaving the active leaf state but also those that leave the implicitly active compound states.

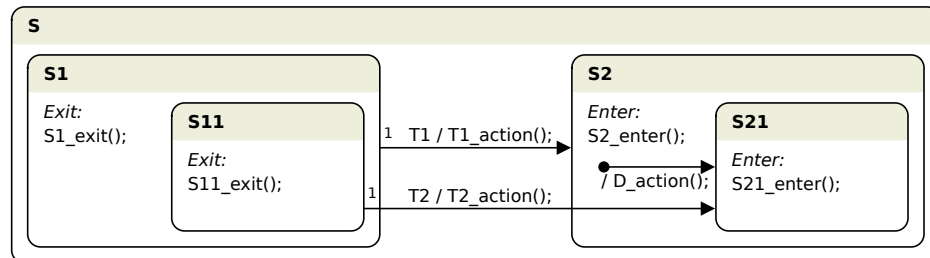
In the case where both a transition from a super-state and a transition from a sub-state are eligible to be taken, the option **Hierarchical Transition Order** in the **State Machine Configuration** dialog determines whether the transition leaving the super-state is given precedence over the transition leaving the sub-state (top to bottom) or vice versa (bottom to top). Note that in this context leaf states are considered to be at the bottom of a state hierarchy.

### Execution Sequence

The two states that are directly connected by a transition are designated the *main source* and the *main target* of the transition. Notice that source state and target state may both be compound states. The lowest compound state that contains both source state and target state is designated the *lowest common ancestor* (LCA) state. When the transition is taken, the following actions are executed in this order:

- 1** The **Exit** actions of all states from the active leaf state (which may be the main source state or a sub-state of it) up to (but not including) the LCA state are executed from bottom to top.
- 2** The transition action is executed.
- 3** The **Enter** actions of the state hierarchy inside the LCA state to the main target state are executed from top to bottom.
- 4** If the main target is a compound state, the action of its local default transition is executed followed by the **Enter** action of the default transition's target state. If necessary, this process is repeated recursively until a leaf state is reached.

This process is illustrated in the example below. Consider that **S11** is the currently active state, so both **S1** and **S** are also implicitly active.



### Execution sequence in a hierarchical state machine

Transition **T1** has **S1** and **S2** as its main source and target. Their LCA state is **S**. **S2** is a compound state, and therefore after it has been entered, its internal default transition causes **S21** to be entered. Consequently, taking **T1** causes the following sequence of actions to be executed: **S11\_exit()**; **S1\_exit()**; **T1\_action()**; **S2\_enter()**; **D\_action()**; **S21\_enter()**;

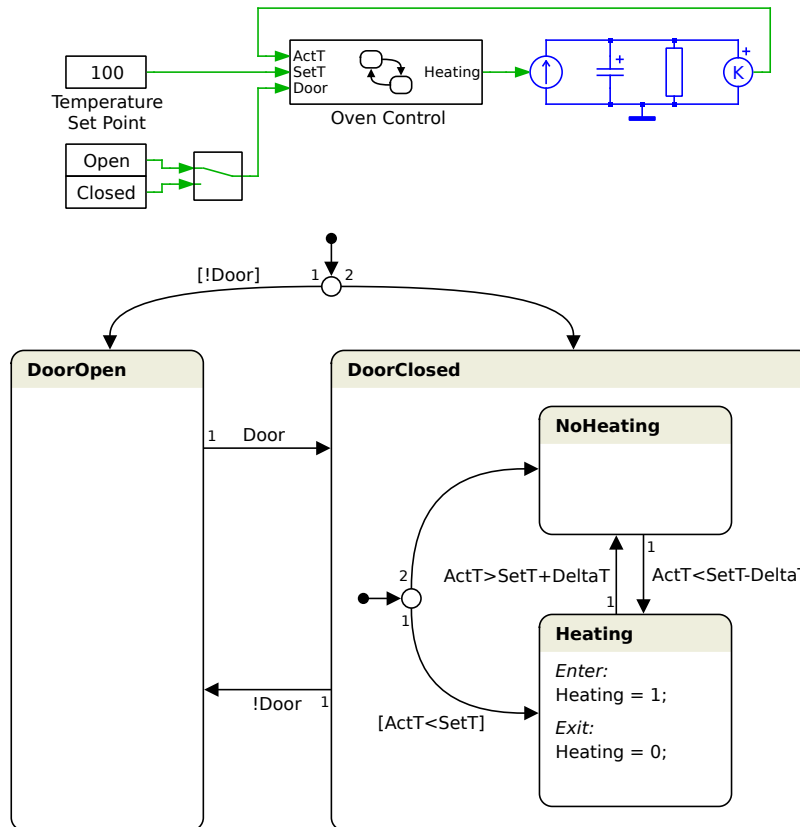
Transition **T2** has **S11** and **S21** as its main source and target. Their LCA state is again **S**. Taking **T2** causes the following sequence of actions to be executed: **S11\_exit()**; **S1\_exit()**; **T2\_action()**; **S2\_enter()**; **S21\_enter()**;

## State Machine Examples

### Oven Control

This example demonstrates a simple oven control. While the oven is in operation, a hysteresis type control shall keep the oven temperature within a certain tolerance band around a set point by switching the heating on and off. As a safety measure, we also want to ensure that the heating is always switched off when the oven door is open.

The oven is modeled with a heat source, a thermal capacitance and resistance and a thermometer. The oven control is implemented with a state machine. Its inputs are the actual and the desired temperature and the status of the oven door, which is defined as 0 when the door is open and 1 when the door is closed. The state machine output is the command for the oven heating. The



**Example of a simple oven control**

state machine uses a constant variable  $\Delta T$  that determines the limits of the tolerance band around the desired temperature.

The top-level states **DoorOpen** and **DoorClosed** reflect the actual state of the oven door. The default transition, which is taken at the first execution of the state machine, is branched with a Junction. If the door is initially open (i.e.  $Door == 0$ ), **DoorOpen** state will be entered, else the **DoorClosed** state. Notice that the unconditional “else” branch leaving the Junction has a lower precedence than the conditional branch. Afterwards, the state machine will transition from **DoorOpen** to **DoorClosed** when Door becomes non-zero and vice versa when the input variable Door becomes zero.

**DoorClosed** is a compound state. Therefore, when it is the target of a transi-

tion, its internal default transition will be executed, which is also branched with a Junction. If the actual temperature is below the set point when the default transition executes, the **Heating** state will be entered, else the **NoHeating** state. Afterwards, the state machine will transition from **Heating** to **NoHeating** when the actual temperature becomes greater than the upper limit and from **NoHeating** to **Heating**, when the actual temperature becomes less than the lower limit.

The output variable Heating is set to 1 when the **Heating** state is entered, and to 0 when the state is left. Notice that it does not matter whether the state is left because the transition  $ActT > SetT + \Delta T$  is taken when the temperature exceeds the upper limit, or because the transition **!Door** is taken when the oven door is opened. In either case, the **Exit** action of **Heating** is executed and the heating is turned off.

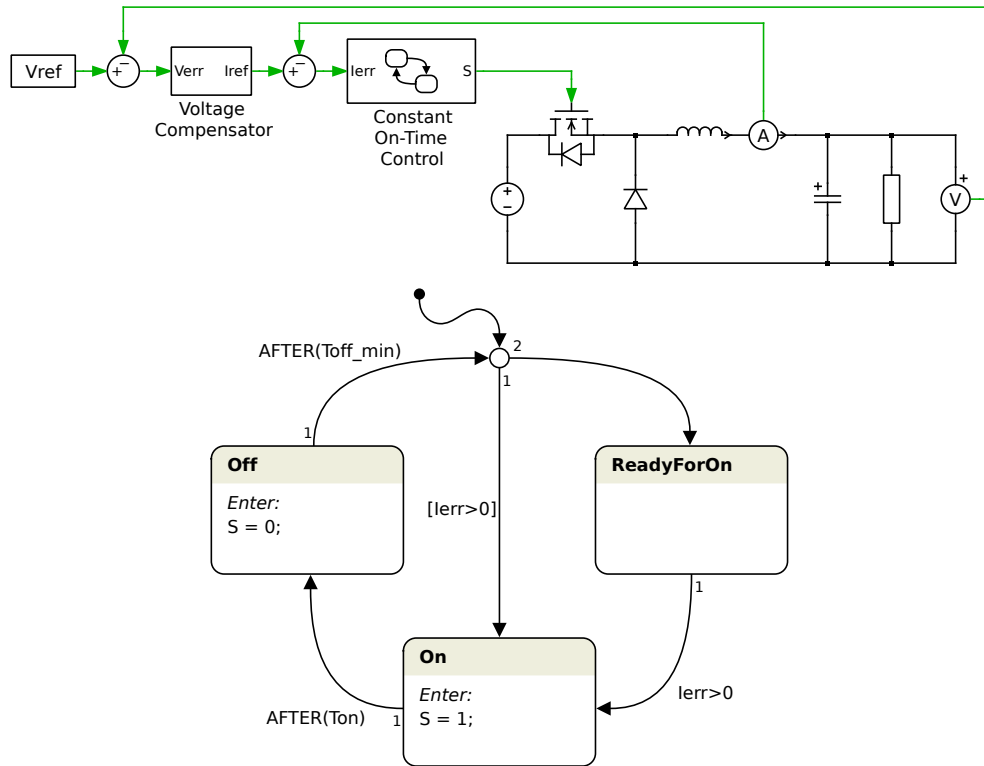
For this state machine to work properly, the **Hierarchical Transition Order** must be set to top to bottom (which is the default). This is important because a door opening event might coincide with the event that the actual temperature exceeds the upper or lower limit. The higher-level door opening event must take precedence over the lower-level temperature events so that the state machine will unconditionally transition to the **DoorOpen** state.

## Constant On-Time Control

This example demonstrates a constant on-time control for a buck converter. The outer control loop computes a current reference from the difference between the actual output voltage and the voltage reference. The inner loop uses a constant on-time control scheme as follows: When the actual current becomes less than the reference current, the FET shall be switched on for a fixed duration. After the FET has been switched off, it must remain switched off for a minimal duration before it can switch on again.

This scheme is implemented with a state machine that has the current error  $I_{err}$  as its input and the FET switching signal  $S$  as its output. The constant on-time and the minimum off-time are configured as constant variables  $T_{on}$  and  $T_{off\_min}$ . The state machine has two states **On** and **Off** that are used to switch the FET on and off, and a waiting state **ReadyForOn**.

When state **On** is entered, its **Enter** action will set the output signal  $S$  to 1. Exactly  $T_{on}$  seconds after the entry of state **On**, the state machine is executed again and the timer event  $AFTER(T_{on})$  becomes active. This causes the state machine to transition to state **Off** and the output signal  $S$  is set to 0. Exactly  $T_{off\_min}$  seconds after this event, the state machine is executed again,



**Example of a buck converter with constant on-time control**

and the timer event `AFTER(Toff_min)` becomes active. The transition for this timer event is branched with a Junction. If the input signal `Ierr` is *already* greater than 0 at this instant, the state machine transitions directly to state **On** again. Else, if `Ierr` is *not* greater than 0, the state machine transitions to the waiting state **ReadyForOn** and remains there until `Ierr` becomes greater than 0.

The default transition targets the same Junction so that the state machine will start in the **On** state or the **ReadyForOn** state depending on whether `Ierr` is greater than 0 at simulation start.

# 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 239).

## Command Line Interface in PLECS Blockset

PLECS Blockset offers a Command Line Interface (CLI) to access component and circuit parameters from scripts or 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.

## Handling of Errors and Warnings in Initialization Commands

During the execution of mask initialization commands, error messages issued with the MATLAB command `error` are caught by PLECS and shown in the Diagnostics window. To show a warning message in the Diagnostics window, use the command

```
plecs('warning', 'warning message')
```

Note that the native MATLAB command `warning` will only print the warning message to the MATLAB command window.



## Handling of 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 and unique, *traceName* is used as the name for the new trace, otherwise a default name is assigned. In both cases the method returns the name given to the trace.

```
plecs('scope', 'scopePath', 'RemoveTrace', 'traceName')
```

removes the trace named *traceName* from the scope indicated by the *scopePath*.

```
plecs('scope', 'scopePath', 'ClearTraces')
```

clears all traces in the scope indicated by the *scopePath*.

```
plecs('scope', 'scopePath', 'SaveTraces', 'fileName')
```

Saves the trace data of the scope at *scopePath* to the file *fileName* for later reference. If *fileName* is not an absolute path, it is interpreted relative to the model file that contains the scope.

```
plecs('scope', 'scopePath', 'LoadTraces', 'fileName')
```

Loads the trace data from the file *fileName* into the scope at *scopePath*. If *fileName* is not an absolute path, it is interpreted relative to the model file that contains the scope.

## Retrieving Scope Cursor Data

```
plecs('scope', 'scopePath', 'GetCursorData', [t1 t2])
plecs('scope', 'scopePath', 'GetCursorData', [t1 t2], ...
      'analysis1', 'analysis2', ...)
```

returns a struct with the signal values and analysis results (as specified) for the cursor positions `t1` and `t2`. Valid analysis names are `delta`, `min`, `max`, `absmax`, `mean`, `rms` and `thd`. For more information about scope cursors see section “Cursors” (on page 94).

The return value is a struct with the two fields `time` and `cursorData`. The field `time` is the vector `[t1, t2]`. The field `cursorData` is a nested cell array where the outer index corresponds to the number of plots in the scope and the inner index corresponds to the number of signals in a plot. Each cell is a struct with the fields `cursor1` and `cursor2` with the signal values and additional fields for the analyses that you have specified. If the scope has multiple traces, the field values are vectors with one element for each trace.

So, if the return value is stored in `data`, to access the signal value at cursor 2 for the third trace of the second signal in the first plot, you write

```
data.cursorData{1}{2}.cursor2(3)
```

### Exporting Scope Data

```
plecs('scope', 'scopePath', 'ExportCSV', 'fileName')  
plecs('scope', 'scopePath', 'ExportCSV', 'fileName', [t1 t2])
```

saves all scope data or only the specified time range as comma separated values to the text file `fileName`. If `fileName` is not an absolute path, it is interpreted relative to the model file that contains the scope.

### Exporting Scope Bitmaps

```
plecs('scope', 'scopePath', 'ExportBitmap', 'fileName')  
plecs('scope', 'scopePath', 'ExportBitmap', 'fileName', ...)
```

saves a bitmap of the scope to the file `fileName`. If `fileName` is not an absolute path, it is interpreted relative to the model file that contains the scope. The file format is determined automatically from the file extension. The `fileName` argument may be followed by one or more name/value pairs to override settings defined in the scope as described in table “Scope Settings for Bitmap Export” (on page 244).

## 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')
```

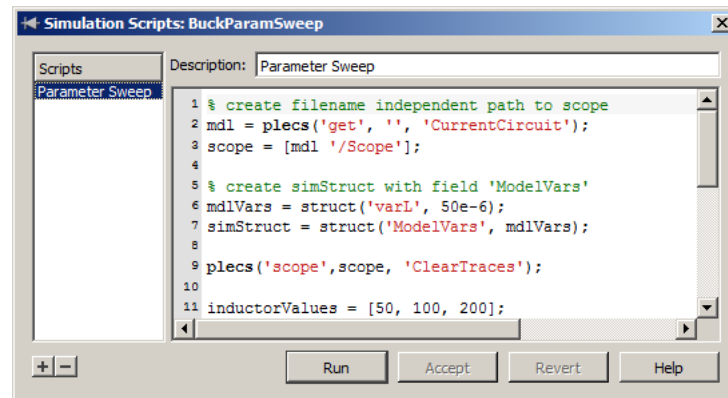
exports the model mdl to the model file exported.mdl which can be opened with the PLECS Viewer. The contents of all PLECS schematics are hidden.

```
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/>.

Any console output generated by Octave will appear in the **Octave Console** window, which you can open by choosing **Show Console** from the **Window** menu.

## 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.

### Clearing the Octave Console

The command

```
plecs('clc')
```

clears the console window. Note that the native Octave commands `clc` and `home` do not have any effect on the console window.

### 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 'CurrentComponent' can be used to query the path of the current component as defined above. The component path has to be an empty string:

```
plecs('get', '', 'CurrentComponent')
```

The special parameter 'CurrentCircuit' can be used to query the name of the model that is currently executed. It cannot be queried interactively from the console. 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.

A leading dot (.) in the component path is substituted with the current component or model as described in “Path Substitution” (on page 245).

## Handling of Errors and Warnings in Initialization Commands

During the execution of model or mask initialization commands, error messages issued with the Octave command error are caught by PLECS and shown in the Diagnostics window. To show a warning message in the Diagnostics window, use the command

```
plecs('warning', 'warning message')
```

Note that the native Octave command warning will only print the warning message to the Octave console.

## Handling of 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 and unique, *traceName* is used as the name for the new trace, otherwise a default name is assigned. In both cases the method returns the name given to the trace.

```
plecs('scope', 'scopePath', 'RemoveTrace', 'traceName')
```

removes the trace named *traceName* from the scope indicated by the *scopePath*.

```
plecs('scope', 'scopePath', 'ClearTraces')
```

clears all traces in the scope indicated by the *scopePath*.

```
plecs('scope', 'scopePath', 'SaveTraces', 'fileName')
```

saves the trace data of the scope at *scopePath* to the file *fileName* for later reference. If *fileName* is not an absolute path, it is interpreted relative to the model file that contains the scope.

```
plecs('scope', 'scopePath', 'LoadTraces', 'fileName')
```

loads the trace data from the file *fileName* into the scope at *scopePath*. If *fileName* is not an absolute path, it is interpreted relative to the model file that contains the scope.

The *scopePath* is the path to the scope within the model including the model name, e.g. 'DTC/Mechanical'. To access Bode plots from the analysis tools, use the model name followed by '/Analyses/' followed by the name of the analysis, e.g. 'BuckOpenLoop/Analyses/Control to Output TF (AC Sweep)'. A leading dot (.) in the scope path is substituted with the current component or model as described in "Path Substitution" below.

### Retrieving Scope Cursor Data

```
plecs('scope', 'scopePath', 'GetCursorData', [t1 t2])  
plecs('scope', 'scopePath', 'GetCursorData', [t1 t2], ...  
      'analysis1', 'analysis2', ...)
```



returns a struct with the signal values and analysis results (as specified) for the cursor positions `t1` and `t2`. Valid analysis names are `delta`, `min`, `max`, `absmax`, `mean`, `rms` and `thd`. For more information about scope cursors see section “Cursors” (on page 94).

The return value is a struct with the two fields `time` and `cursorData`. The field `time` is the vector `[t1, ts]`. The field `cursorData` is a nested cell array where the outer index corresponds to the number of plots in the scope and the inner index corresponds to the number of signals in a plot. Each cell is a struct with the fields `cursor1` and `cursor2` with the signal values and additional fields for the analyses that you have specified. If the scope has multiple traces, the field values are vectors with one element for each trace.

So, if the return value is stored in `data`, to access the signal value at cursor 2 for the third trace of the second signal in the first plot, you write

```
data.cursorData{1}{2}.cursor2(3)
```

### Exporting Scope Data

```
plecs('scope', 'scopePath', 'ExportCSV', 'fileName')
plecs('scope', 'scopePath', 'ExportCSV', 'fileName', [t1 t2])
```

saves all scope data or only the specified time range as comma separated values to the text file `fileName`. If `fileName` is not an absolute path, it is interpreted relative to the model file that contains the scope.

### Exporting Scope Bitmaps

```
plecs('scope', 'scopePath', 'ExportBitmap', 'fileName')
plecs('scope', 'scopePath', 'ExportBitmap', 'fileName', ...)
```

saves a bitmap of the scope to the file `fileName`. If `fileName` is not an absolute path, it is interpreted relative to the model file that contains the scope. The file format is determined automatically from the file extension. The `fileName` argument may be followed by one or more name/value pairs to override settings defined in the scope as follows:

**Scope Settings for Bitmap Export**

<b>Name</b>	<b>Description</b>
Size	A two-element integer vector specifying the width and height of the bitmap in pixels.
Resolution	An integer specifying the resolution of the bitmap in pixels per inch.
TimeRange	A two-element real vector specifying the time range of the data to be shown.
XLim	A two-element real vector specifying the limits of the x-axis. For a normal scope this is equivalent to TimeRange.
YLim	A cell array containing two-element real vectors specifying the limits of the y-axis of the plot(s). For a scope with a single plot a vector may be given directly.
XLabel	A string specifying the x-axis label.
YLabel	A cell array containing strings specifying the y-axis labels of the plot(s). For a scope with a single plot a string may be given directly.
Title	A cell array containing strings specifying the title of the plot(s). For a scope with a single plot a string may be given directly.
LegendPosition	A string specifying the legend position. Possible values are none, topleft, topmiddle, topright, bottomleft, bottommiddle and bottomright.
Font	A string specifying the font name to be used for the labels and the legend.
LabelFontSize	An integer specifying the label font size in points.
LegendFontSize	An integer specifying the legend font size in points.

## Path Substitution

If a component or scope path is a simple dot (.) or starts with a dot followed by a slash (./) the dot is substituted with the *current component*.

When a command is executed interactively from the console, the current component is the last component that was clicked on in the schematic editor; if the last click was not on a component, the current component is undefined.

During the evaluation of block parameters or mask initialization commands the current component is the component that is currently evaluated; during the evaluation of the model initialization commands it is the model itself.

## 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 251).

If any outputs 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 outputs. 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 251).

For a Steady-State Analysis, if any outputs 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 outputs are captured after a steady-state operating point has been obtained.

For an AC Sweep, an Impulse Response Analysis or a Multitone 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. If the command is called without a return value, a scope window will open and display the Bode diagram of the transfer function.

## 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 BuckParam-Sweep.

```
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 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 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 scope path uses a dot to reference the current model (see “Path Substitution” on page 245). 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*.

## Handling of 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', 'RemoveTrace', 'traceName')
```

removes the trace named *traceName* from the scope indicated by the *scopePath*.

```
plecs.scope('scopePath', 'ClearTraces')
```

clears all traces in the scope indicated by the *scopePath*.

```
plecs.scope('scopePath', 'SaveTraces', 'fileName')
```

Saves the trace data of the scope at *scopePath* to the file *fileName* for later reference. If *fileName* is not an absolute path, it is interpreted relative to the model file that contains the scope.

```
plecs.scope('scopePath', 'LoadTraces', 'fileName')
```

Loads the trace data from the file *fileName* into the scope at *scopePath*. If *fileName* is not an absolute path, it is interpreted relative to the model file that contains the scope.

The *scopePath* is the path to the scope within the model including the model name, e.g. 'DTC/Mechanical'. To access Bode plots from the analysis tools, use the model name followed by '/Analyses/' followed by the name of the analysis, e.g. 'BuckOpenLoop/Analyses/Control to Output TF (AC Sweep)'.

## 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 251).

If any outputs 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 outputs. 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 251).

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 parameter 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. Values can be numerical scalars, vectors, matrices or 3d arrays or strings.

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.

**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.

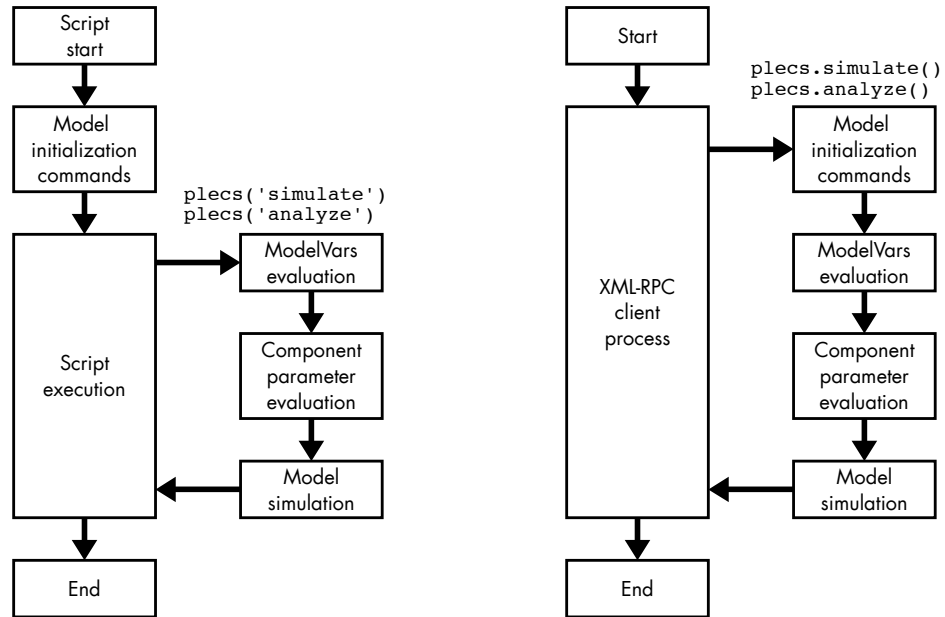
### Solver Options in Scripted Simulations

Parameter	Description
<code>Solver</code>	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 107) for more details.
<code>StartTime</code>	The start time specifies the initial value of the simulation time variable <i>t</i> at the beginning of a simulation, in seconds.
<code>StopTime</code>	The simulation ends when the simulation time has advanced to the specified stop time.

**Solver Options in Scripted Simulations (contd.)**

<b>Parameter</b>	<b>Description</b>
Timeout	A non-negative number that specifies the maximum number of seconds (wall-clock time) that a simulation or analysis is allowed to run. After this period the simulation or analysis is stopped with a timeout error. A value of 0 disables the timeout.
MaxStep	See parameter <b>Max Step Size</b> in section “Standalone Parameters” (on page 107). This parameter is only evaluated for variable step solvers.
InitStep	See parameter <b>Initial Step Size</b> in section “Standalone Parameters” (on page 107). 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 <b>Tolerances</b> in section “Standalone Parameters” (on page 107).
RelTol	See the description for <b>Tolerances</b> in section “Standalone Parameters” (on page 107).
Refine	See parameter <b>Refine factor</b> in section “Standalone Parameters” (on page 107).

**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



**Execution order for Simulation Scripts (left) and XML-RPC (right)**

### 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.

**Analysis Options in Scripted Analyses (contd.)**

<b>Parameter</b>	<b>Description</b>
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.
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.

**Analysis Options in Scripted Analyses (contd.)**

<b>Parameter</b>	<b>Description</b>
Perturbation	The full block path (excluding the model name) 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 (excluding the model name) 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.
ShowResults	Specifies whether to show a Bode plot after a small-signal analysis. This parameter is evaluated only for a small-signal analysis.



# Code Generation

As a separately licensed feature, PLECS can generate C code from a simulation model to facilitate real-time simulations. Code generation is subject to certain limitations, which are described in the first section of this chapter. The next two sections describe how the code generation capabilities are used within PLECS Standalone and PLECS Blockset, respectively.

## Code Generation for Physical Systems

As described earlier in this manual, PLECS models physical systems using piecewise linear state-space equations that can be described using multiple sets of state-space matrices. For details see “Physical Model Equations” (on page 28).

During normal simulations, PLECS calculates new sets of state-space matrices on the fly as the individual switching components change their states. This is not possible in the generated C code because the algorithms for calculating the matrices are proprietary and because the calculation would simply be too time consuming under real-time constraints.

When generating code for a physical model, PLECS therefore embeds the matrices for *all* combinations of switch states that it expects to encounter during the execution of a simulation run. In general this means that for a system with  $n$  switch elements  $2^n$  sets of state-space matrices are calculated and embedded into the generated C code. The actual number can be reduced by eliminating impossible combinations. For instance, the Triple Switch (see page 691) blocks internally consists of 3 switch elements but it still only accounts for 3 instead of  $2^3$  combinations because one and only one switch will conduct at any time. Even so, systems with many switches will lead to large source and executable files and long compile times.

## Reducing the Code Size

You can reduce the size of the generated code and the required compile time by specifying the combinations for which code should be generated. The combinations are specified using special Model Settings blocks that are connected to individual physical models, see Electrical Model Settings on page 389, Rotational Model Settings on page 552 and Translational Model Settings on page 672.

If at run-time a combination is encountered, for which no code has been generated, the execution is aborted with an error message.

## Maximum Number of Switches

The number of switch elements is limited to 16 or 32 per physical domain depending on the integer word size. This is due to the fact that the switch states of a physical domain are stored internally in a single unsigned integer variable. Note that some components, such as the Triple Switch (see page 691), internally consist of more than one switch element.

## Handling Naturally Commutated Devices

Switched physical models are difficult to handle in real-time simulations if the natural switching instants can occur *between* two time steps. This is the case for naturally commutated components where switching events are triggered by internal quantities of the physical model. Examples of such components are the diode (which turns on when the voltage becomes positive and turns off when the current becomes negative) or the mechanical friction components (which start slipping when the torque resp. force exceeds a certain level and become stuck when the speed becomes zero).

During normal simulations, PLECS handles such non-sampled switching events using an interpolation scheme (see “Interpolation of Non-Sampled Switching Events” on page 34). This is not practical under real-time constraints because the computation time required for the interpolation is several times larger than that of an ordinary simulation step. In real-time simulations, PLECS will therefore defer the switching to the immediately following time step. Note that this reduces the accuracy compared to a normal simulation.



## Switching Algorithm

A further difficulty with naturally commutated devices is that their conduction state is usually influenced by the conduction states of other switches. During normal simulations, PLECS solves this problem by iteratively toggling the conduction states of naturally commutated switches within one simulation step until the boundary conditions of all switches are satisfied (see the description of the Switch Manager in section “Physical Model Equations” on page 28).

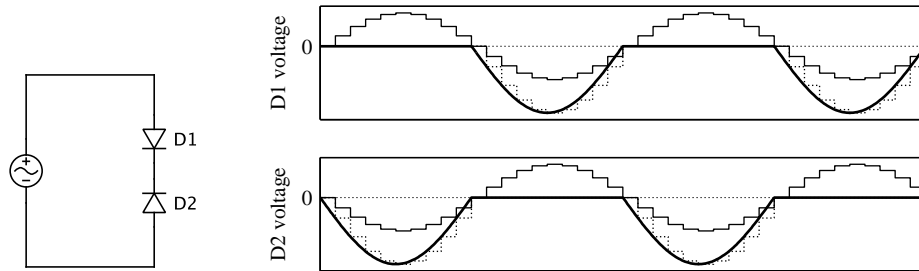
When generating code for a physical model, PLECS lets you choose between two switching algorithms, **Iterative** and **Direct Look-up**. You can specify the algorithm individually for each physical model using special Model Settings blocks that are connected to individual physical models, see Electrical Model Settings on page 389, Rotational Model Settings on page 552 and Translational Model Settings on page 672. The default switching algorithm is **Direct Look-up**.

**Iterative** PLECS generates code that implements an iterative switching method as described above. As a consequence of the iteration, simulation steps, in which a switching occurs, require more computation time than simulation steps without switching events. This is usually undesirable in real-time simulations because the longest execution time determines the feasible sample rate.

**Direct Look-up** Alternatively, PLECS can generate code that determines the proper switch conduction states directly as functions of the current physical model states and inputs and the gate signals of externally controlled switches. In order to generate these direct look-up functions, PLECS must analyze all possible transitions between all possible combinations of conduction states. This increases the computation time of the code generation process but yields nearly uniform execution times of simulation steps with or without switching events.

In order to reduce the number of possible combinations of switch conduction states and thus the code generation time and the code size, PLECS introduces the condition that naturally commutated devices (e.g. diodes or IGBTs) can only conduct if their current is non-zero. As a consequence, a diode may block even though the voltage is forward-biased if there is another blocking switch connected in series that prevents the current from flowing through the diode. This can produce unexpected voltage waveforms even though otherwise the model behaves correctly.

Consider the simple circuit shown below. Two diodes are connected in series but opposing each other so that no current can flow regardless of the polarity of the source voltage:



### Simulation of a circuit with two opposing diodes

In the two graphs, the bold lines show the results from a normal simulation. When the source voltage is positive, D1 conducts and D2 blocks, and hence the voltage across D1 is zero and the voltage across D2 equals the negative source voltage. As the source voltage becomes negative, D1 blocks and D2 conducts, and accordingly the voltage across D1 equals the source voltage and the voltage across D2 is zero.

The dotted stairstep lines show the results from generated code using the iterative switching method. As can be seen, the diodes behave in the same way as in the normal simulation. In the generated code using the direct look-up method shown with continuous stairstep lines, however, both diodes block at all times because no current can ever flow through either of them. Accordingly, the source voltage always divides evenly across the two diodes.

## Data Types

During normal offline simulations, PLECS uses double precision floating-point numbers as the sole data type for all signal connections. However, when you generate code, you can use other data types to increase the efficiency of the generated code.

The following table lists the data types supported by the PLECS Coder.

### Data Types

Name	Description
bool	Boolean
uint8_t	Unsigned 8-bit integer
int8_t	Signed 8-bit integer
uint16_t	Unsigned 16-bit integer
int16_t	Signed 16-bit integer
uint32_t	Unsigned 32-bit integer
int32_t	Signed 32-bit integer
float	Single-precision floating point
double	Double-precision floating point

Certain blocks have an implicit output data type other than floating point:

- Comparator (see page 344), bool
- Edge Detection (see page 384), bool
- Hit Crossing (see page 407), bool
- Logical Operator (see page 474), bool
- Monoflop (see page 491), bool
- Relational Operator (see page 540), bool
- Sign (see page 586), int32\_t

Other blocks let you specify the data type of their output signals:

- Constant (see page 347)

- Data Type (see page 356)
- Enable (see page 392)
- Gain (see page 399)
- Offset (see page 511)
- Product (see page 530)
- Pulse Generator (see page 533)
- Relay (see page 541)
- Signal Input (see page 580)
- Step (see page 602)
- Sum (see page 606)
- Rounding (see page 558)

A third group of blocks inherits the data type from the input signals.

- Delay (see page 360)
- Initial Condition (see page 463)
- Memory (see page 485)
- Minimum / Maximum (see page 487)
- Signal Switch (see page 585)
- Zero Order Hold (see page 725)

The other blocks use the default floating point data type. In the PLECS Standalone Coder, the default floating point data type can be specified in the **Coder Options** dialog (see “Generating Code” on page 264). In the PLECS Blockset Coder, the default floating point data type is double.

## Unsupported Components

PLECS currently does not support code generation for the following components:

- Algebraic Constraint (see page 315)
- Brushless DC Machine (see page 330)
- Electrical Algebraic Component (see page 385)
- Non-Excited Synchronous Machine (see page 507)
- Rotational Algebraic Component (see page 543)
- Switched Reluctance Machine (see page 608)
- Synchronous Reluctance Machine (see page 627)
- Translational Algebraic Component (see page 663)
- Variable Capacitor (see page 695)
- Variable Inductor (see page 698)
- Variable Resistor (see page 704)
- Variable Resistor with Constant Parallel Capacitor (see page 705)
- Variable Resistor with Constant Series Inductor (see page 706)
- Variable Resistor with Variable Parallel Capacitor (see page 707)
- Variable Resistor with Variable Series Inductor (see page 709)

PLECS also does not support code generation for models that contain algebraic loops.

## Code Generation with PLECS Standalone

PLECS Standalone produces C code in an embedded format, generating entry point functions that must be called from a main application. The following functions are generated:

**void *model\_initialize*(double time)**

This function should be called once at the beginning of a simulation to initialize the internal data structures and the start value of the global clock for components that depend on the absolute time.

**void *model\_step*()**

This function should be called at every simulation step to advance the model by one step.

**void *model\_terminate*()**

This function should be called at the end of a simulation to release resources that were acquired at the beginning of or during a simulation.

The prefix *model* is replaced by a model-specific string.

If a runtime error occurs during the execution of any of the three functions above, the variable `const char * model_errorStatus` points to a string with the error message. It is initialized with NULL.

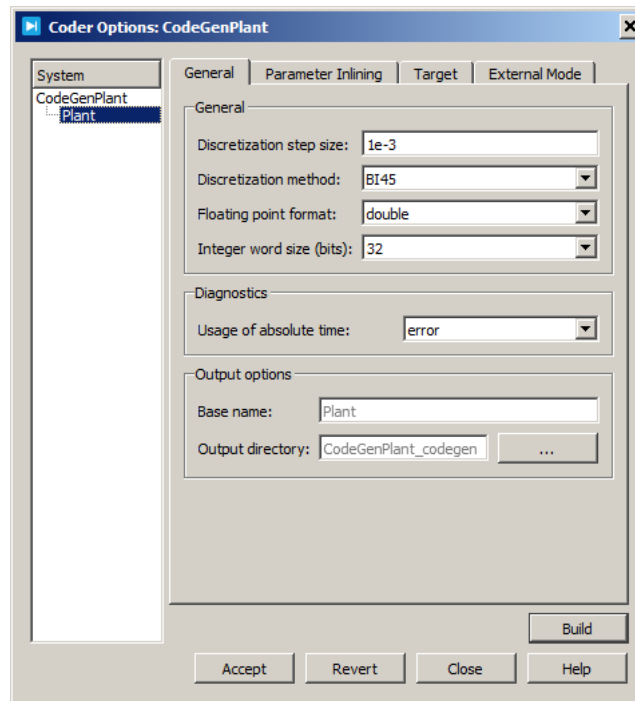
### Generating Code

To generate code, choose **Coder options...** from the **Coder** menu. This menu only appears if you have a license for the PLECS Coder.

The left hand side of the dialog window shows a tree view of the model and the code-generation subsystem that it contains. Intermediate subsystems that have not been enabled for code generation appear as disabled entries that cannot be selected.

To enable the model itself for code generation, choose the **Fixed-Step Discrete** solver in the simulation parameters of PLECS Standalone (or choose the **Discrete state-space** circuit model in the simulation parameters of PLECS Blockset).

To enable a subsystem for code generation, select the subsystem, then choose **Execution settings...** from the **Subsystem** submenu of the **Edit** menu or the context menu. In the Subsystem Settings dialog check the option **Enable code generation**. This option is only shown if you have a license for the PLECS Coder. Checking this option implicitly also checks the option **Treat as**



**atomic unit** and unchecks the option **Minimize occurrence of algebraic loops**. For more information on these options see “Virtual and Atomic Subsystems” (on page 603).

The right hand side of the dialog window shows a tabbed dialog with the code generation options for the system that is selected in the tree view. To generate code for the currently selected system, click on the **Build** button at the bottom of the right hand side.

Code generation can also be initiated in an Octave simulation script or via the XML-RPC interface using the commands

```
plecs('codegen', 'path', optStruct, 'outputDir')
```

or

```
plecs.codegen('path', optStruct, 'outputDir')
```

respectively, where *path* is a model name or a subsystem path. The parameters *optStruct* and *outputDir* are optional.

If *optStruct* is provided, it is expected to be a struct as described in “Scripted Simulation and Analysis Options” (on page 251). This enables you e.g. to generate code for different parameter values without having to modify the model file.

If *outputDir* is provided, the generated files will be placed in this folder instead of the folder that is specified in the model.

## General

**Discretization step size** This parameter specifies the base sample time of the generated code and is used to discretize the physical model equations (see “Physical Model Discretization” on page 33).

**Discretization method** This parameter specifies the algorithm used to discretize the physical model equations (see “Physical Model Discretization” on page 33) and continuous state variables of control blocks.

**Floating point format** This parameter specifies the default data type (float or double) that is used for floating point variables in the generated code.

**Usage of absolute time** This setting allows you to specify the diagnostic action to be taken if PLECS generates code for a component that depends on the absolute time. In order to minimize round-off errors, PLECS generates code to calculate the absolute time using a signed 64-bit integer tick counter. If a simulation runs for an infinite time, this tick counter will eventually reach its maximum value, where it is halted to avoid problems that might occur if the counter was wrapped to the (negative) minimum value, such as a Step block (see page 602) resetting to its initial value. To put things into perspective, assuming a step size of  $1\mu s$  this would occur after 292 271 years.

Depending on this setting, PLECS will either ignore this condition or it will issue a warning or error message that indicates the components that use the absolute time.

**Base name** This parameter allows you to specify a custom prefix used to name the generated files and the exported symbols such as the interface functions or the input, output and parameter structs. By default, i.e. if you clear this field, the base name is derived from the model or subsystem name.

**Output directory** This parameter allows you to specify, where the generated files are stored. This can be an absolute path or a relative path with respect to the location of the model file. The default path is a directory *model\_codegen* next to the model file.




## Parameter Inlining

These two settings specify how PLECS handles tunable parameters in the generated code.

**Default behavior** This setting specifies whether PLECS inlines the parameter values as numeric constants directly into the code (Inline parameter values) or generates a data structure from which the values are read (Keep parameters tunable).

Inlining parameter values reduces the code size and increases the execution speed. However, changing an inlined parameter value requires regenerating and recompiling the code. On the other hand, the values of tunable parameters can be changed at execution time without recompiling the code.

**Exceptions** For the components listed here, the *opposite* of the default behavior applies. If the default behavior is to inline all parameter values, the components listed here will keep their parameters tunable and vice versa. To add components to this list, simply drag them from the schematic into the list. Use the **Remove** – button to remove components from the list. To view the selected component in the schematic editor, click the **Show component**  button.

---

**Note** Physical component parameters that affect the physical model equations, such as resistances or inductances, cannot be kept tunable, because changing such parameters in general requires a recalculation of the complete equation system. You can, however, keep the parameters of source blocks tunable.

---

## Target

On this tab you can select a code generation target and configure target-specific settings. By default, only the **Generic** target is available, which does not have any specific settings. For information on how to install additional code generation targets, see “Coder Configuration” (on page 44).

## External Mode

This tab is only enabled if you have selected a target that supports external mode operation and if the corresponding target setting is enabled. External mode operation enables you to

- Capture data on a target device and show them in Scope (see page 573), XY Plot (see page 722) and Display (see page 378) blocks in the model on the host computer.
- Change values of tunable parameters (see “Parameter Inlining” (on page 267)) in the model on the host computer and upload the changed values to an application running on a target device.

## Simulating a Subsystem in CodeGen Mode

Enabling a subsystem for code generation also enables the **Simulation Mode** parameter in the **Execution Settings** dialog (see the Subsystem block on page 603). When this parameter is set to **Normal**, which is the default, the subsystem is simulated like a normal atomic subsystem. When the parameter is set to **CodeGen**, the generated code is compiled and linked to PLECS to be executed instead of the subsystem during a simulation.

In connection with the “Traces” feature of the scopes (see “Adding Traces” on page 96), this allows you to easily verify the fidelity of the generated code against a normal simulation.

## Code Generation with PLECS Blockset

### Standalone Code Generation

PLECS Blockset can generate C code in the same format as PLECS Standalone, and you can choose to generate code for a complete Circuit block or for an individual subsystem. To generate code, choose **Coder options...** from the **Coder** menu. This menu only appears if you have a license for the PLECS Coder. For details on the coder options dialog please refer to the previous section “Code Generation with PLECS Standalone” (on page 264).

Code generation can also be initiated via the MATLAB commands

```
plecs('codegen', 'path')  
plecs('codegen', 'path', 'outputDir')
```

If *outputDir* is provided, the generated files will be placed in this folder instead of the folder that is specified in the model.

### Integration with Simulink Coder

In addition, PLECS Blockset fully integrates with Simulink Coder (formerly Real-Time Workshop) to generate C code for your simulation model. Whenever you start a build process from within Simulink, PLECS automatically generates the code for a circuit block and inserts it at the appropriate places into the code generated by Simulink Coder.

---

**Note** Scopes that 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.

---

### Simulink Coder Options

The code generation options for the Simulink Coder are configured on the **Simulink Coder** pane of the PLECS simulation parameters.

**Code generation target** This parameter specifies the code generation target (see “Code Generation Targets” on page 270). The default, `auto`, means that PLECS selects the target depending on the Simulink Coder target.

**Inline circuit parameters for RSim target** Specifies for the RSim target whether component parameters should be evaluated at compile time and inlined into the code (`on`) or evaluated at run time (`off`). See also “Tunable Circuit Parameters in Rapid Simulations” (on page 272).

**Generate license-free code (requires PLECS Coder license)** If this option is checked, PLECS will attempt to check-out a PLECS Coder license at build time and, if successful, will generate code for the RSim target that will run without requiring a PLECS license. Otherwise, the generated executable will require a PLECS license at run time.

**Show code generation progress** If this option is checked, PLECS opens a dialog window during code generation that shows the progress of the code generation process. You can abort the process by clicking the **Cancel** button or closing the dialog window.

## Code Generation Targets

PLECS can generate code for two different Simulink Coder 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

	<b>RSim Target</b>	<b>Real-Time Target</b>
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 naturally commutated devices and non-linear components.

**Comparison of Code Generation Targets (contd.)**

	<b>RSim Target</b>	<b>Real-Time Target</b>
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	If a PLECS Coder license is available at <i>build time</i> , the generated code will run without a license. Otherwise, a PLECS license is required at <i>run time</i> .	Requires a PLECS Coder license at <i>build time</i> .

By default, PLECS automatically selects the appropriate target depending on the target settings of Simulink Coder. To overrule this selection, open the PLECS simulation parameters, and on the **Advanced** pane change the setting **Target** to either RSim or RealTime.

**Real-Time Target**

The Real-Time Target is selected by default when you generate code using any of the real-time targets of Simulink Coder. Code generation for the Real-Time target requires a separate license for the PLECS Coder for PLECS Blockset.

For a detailed description of the code generation process for physical systems and its current limitations see “Code Generation for Physical Systems” (on page 257).

**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 Simulink Coder. 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 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

Depending on the build settings the RSim module may check 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.

## 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 47).
- 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 Simulink Coder 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 Simulink Coder 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.





# Processor-in-the-Loop

As a separately licensed feature, PLECS offers support for *Processor-in-the-Loop* (PIL) simulations, allowing the execution of control code on external hardware tied into the virtual world of a PLECS model.

At the PLECS level, the PIL functionality consists of a specialized PIL block that can be found in the Processor-in-the-loop library, as well as the Target Manager, accessible from the **Window** menu. Also included with the PIL library are high-fidelity peripheral models of MCUs used for the control of power conversion systems.

On the embedded side, a *PIL Framework* library is provided to facilitate the integration of PIL functionality into your project.

## Motivation

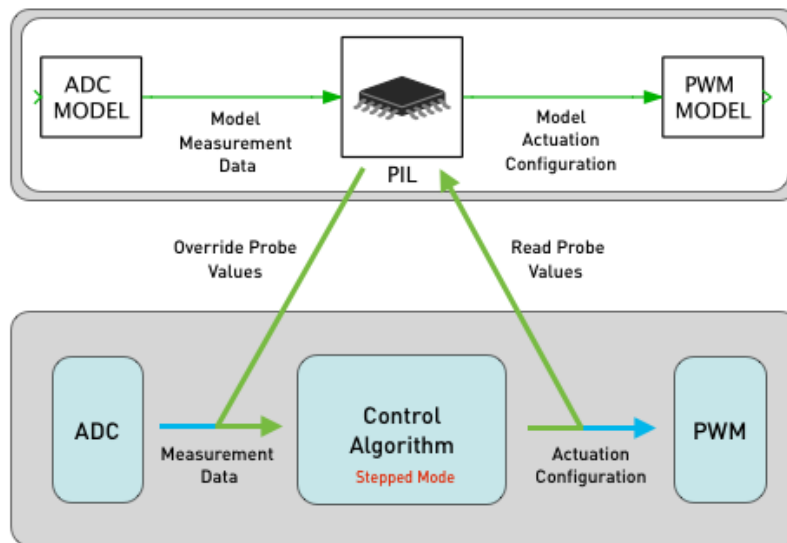
When developing embedded control algorithms, it is quite common to be testing such code, or portions thereof, by executing it inside a circuit simulator. Using PLECS, this can be easily achieved by means of a C-Script or DLL block. This approach is referred to as *Software-in-the-loop* (SIL). A SIL simulation compiles the embedded source code for the native environment of the simulation tool (e.g. Win64) and executes the algorithms within the simulation environment.

The PIL approach, on the other hand, executes the control algorithms on the real embedded hardware. Instead of reading the actual sensors of the power converter, values calculated by the simulation tool are used as inputs to the embedded algorithm. Similarly, outputs of the control algorithms executing on the processor are fed back into the simulation to drive the virtual environment. Note that SIL and PIL testing are also relevant when the embedded code is automatically generated from the simulation model.

One of the major advantages of PIL over SIL is that during PIL testing, actual compiled code is executed on the real MCU. This allows the detection of platform-specific software defects such as overflow conditions and casting errors. Furthermore, while PIL testing does not execute the control algorithms in true real-time, the control tasks *do* execute at the normal rate between two simulation steps. Therefore, PIL simulation can be used to detect and analyze potential problems related to the multi-threaded execution of control algorithms, including jitter and resource corruption. PIL testing can also provide useful metrics about processor utilization.

## How PIL Works

At the most basic level, a PIL simulation can be summarized as follows:



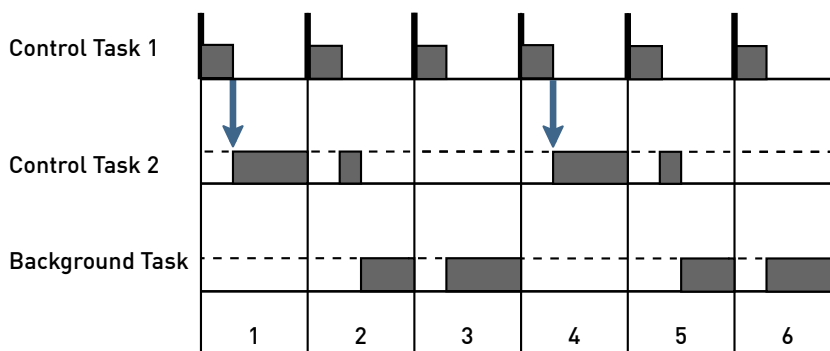
### Principle of a PIL simulation

- Input variables on the target, such as current and voltage measurements, are overridden with values provided by the PLECS simulation.
- The control algorithms are executed for one control period.
- Output variables on the target, such as PWM peripheral register values, are read and fed back into the simulation.

We refer to variables on the target which are overridden by PLECS as *Override Probes*. Variables read by PLECS are called *Read Probes*.

While *Override Probes* are set and *Read Probes* are read the dispatching of the embedded control algorithms must be stopped. The controls must remain halted while PLECS is updating the simulated model. In other words, the control algorithm operates in a stepped mode during a PIL simulation. However, as mentioned above, when the control algorithms are executing, their behavior is identical to a true real-time operation. We therefore call this mode of operation *pseudo real-time*.

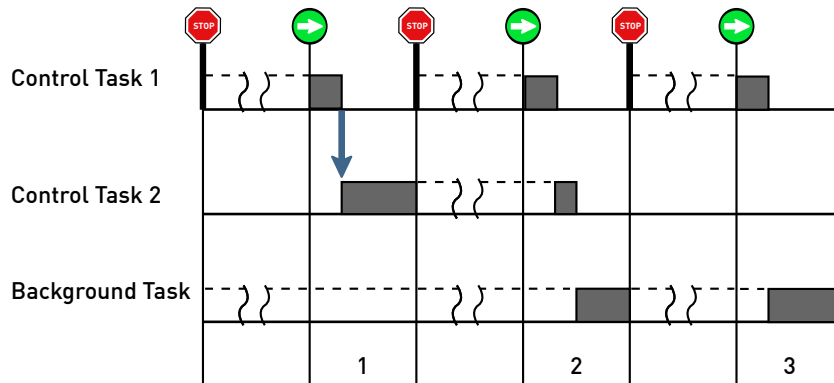
Let us further examine the pseudo real-time operation in the context of an embedded application utilizing nested control loops where fast high-priority tasks (such as current control) interrupt slower lower-priority tasks (such as voltage control). An example of such a configuration with two control tasks is illustrated in the figure below. With every hardware interrupt (bold vertical bar), the lower priority task is interrupted and the main interrupt service routine is executed. In addition, the lower priority task is periodically triggered using a software interrupt. Once both control tasks have completed, the system continues with the background task where lowest priority operations are processed. The timing in this figure corresponds to true real-time operation.



### Nested Control Tasks

The next figure illustrates the timing of the same controller during a PIL simulation, with the *stop* and *go* symbols indicating when the dispatching of the control tasks is halted and resumed.

After the hardware interrupt is received, the system stops the control dispatching and enters a communication loop where the values of the *Override Probes* and *Read Probes* can be exchanged with the PLECS model. Once a new step request is received from the simulation, the task dispatching is



### Pseudo real-time operation

restarted and the control tasks execute freely during the duration of one interrupt period. This pseudo real-time operation allows the user to analyze the control system in a simulation environment in a fashion that is behaviorally identical to a true real-time operation. Note that only the dispatching of the control tasks is stopped. The target itself is never halted as communication with PLECS must be maintained.

## PIL Modes

The concept of using Override Probes and Read Probes allows tying actual control code executing on a real MCU into a PLECS simulation without the need to specifically recompile it for PIL.

You can think of Override Probes and Read Probes as the equivalent of test points which can be left in the embedded software as long as desired. Software modules with such test points can be tied into a PIL simulation at any time.

Often, Override Probes and Read Probes are configured to access the registers of MCU peripherals, such as analog-to-digital converters (ADCs) and pulse-width modulation (PWM) modules. Additionally, specific software modules, e.g. a filter block, can be equipped with Override Probes and Read Probes. This allows unit-testing the module in a PIL simulation isolated from the rest of the embedded code.

To permit safe and controlled transitions between real-time execution of the control code, driving an actual plant, and pseudo real-time execution, in con-

junction with a simulated plant, the following two PIL modes are distinguished:

- **Normal Operation** – Regular target operation in which PIL simulations are inhibited.
- **Ready for PIL** – Target is ready for a PIL simulation, which corresponds to a safe state with the power-stage disabled.

The transition between the two modes can either be controlled by the embedded application, for example based on a set of digital inputs, or from PLECS using the Target Manager.

## Configuring PLECS for PIL

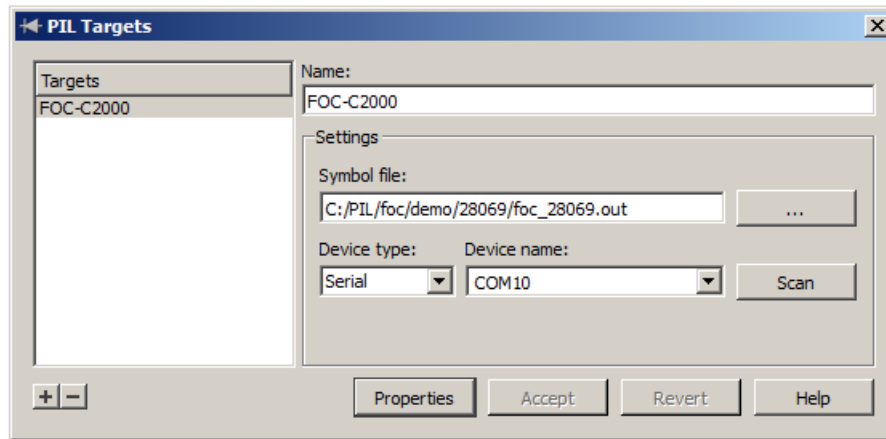
Once an embedded application is equipped with the PIL framework, and appropriate Override Probes and Read Probes are defined, it is ready for PIL simulations with PLECS.

PLECS uses the concept of *Target Configurations* to define global high-level settings that can be accessed by any PLECS model. At the circuit level, the *PIL block* is utilized to define lower level configurations such as the selection of Override Probes and Read Probes used during simulation.

This is explained in further detail in the following sections.

## Target Manager

The high-level configurations are made in the *Target Manager*, which is accessible in PLECS by means of the corresponding item in the **Window** menu. The target manager allows defining and configuring targets for PIL simulation, by associating them with a symbol file and specifying the communication parameters. Target configurations are stored globally at the PLECS level and are not saved in \*.plecs or Simulink files. An example target configuration is shown in the figure below.



### Target Manager

The left hand side of the dialog window shows a list of targets that are currently configured. To add a new target configuration, click the button marked + below the list. To remove the currently selected target, click the button marked -. You can reorder the targets 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 target. Each target configuration must have a unique **Name**.

The target configuration specifies the **Symbol file** and the communication link settings.

The symbol file is the binary file (also called “object file”) corresponding to the code executing on the target. PLECS will obtain most settings for PIL simulations, as well as the list of Override Probes and Read Probes and their attributes, from the symbol file.

### Communication Links

A number of links are supported for communicating with the target. The desired link can be selected in the **Device type** combo box. For communication links that allow detecting connected devices, pressing the **Scan** button will populate the **Device name** combo box with the names of all available devices.

## Serial Device

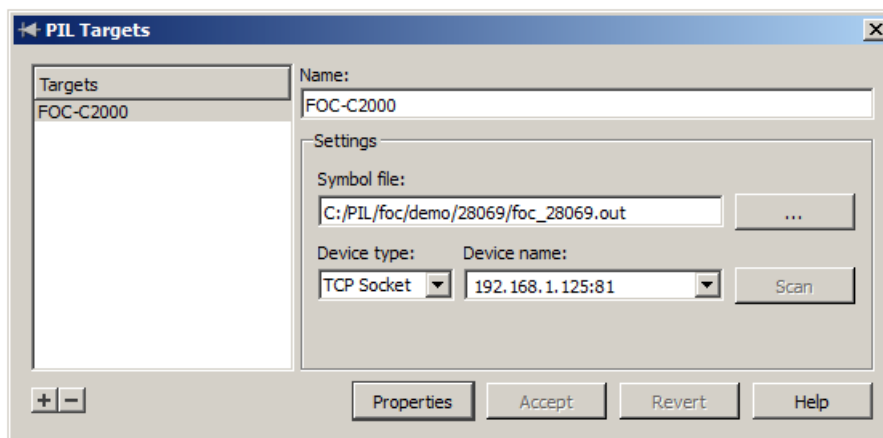
The **Serial device** selection corresponds to conventional physical or virtual serial communication ports. On a Windows machine, such ports are labeled COMn, where n is the number of the port.

## FTDI Device

If the serial adapter is based on an FTDI chip, the low-level FTDI driver can be used directly by selecting the **FTD2XX** option. This device type offers improved communication speed over the virtual communication port (VCP) associated with the FTDI adapter.

## TCP/IP Socket

The communication can also be routed over a TCP/IP socket by selecting the **TCP Socket** device type.



## TCP/IP Communication

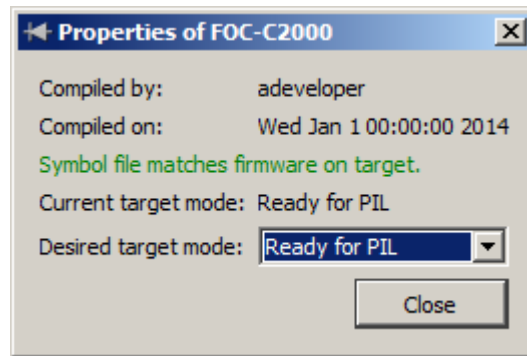
In this case the **Device name** corresponds to the IP address (or URL) and port number, separated by a colon (:).

## Serial over GDB

The **Serial over GDB** device type is used in conjunction with communication over JTAG. It requires that a GDB server be running and connected to the embedded target. The configuration of this device is similar to the **TCP Socket** and consists of specifying a URL (typically the localhost 127.0.0.1) and a TCP/IP port. Please review the documentation of your GDB server for more information regarding port settings.

## Target Properties

By pressing the **Properties** button, target information can be displayed as shown in the figure below.



## Target Properties

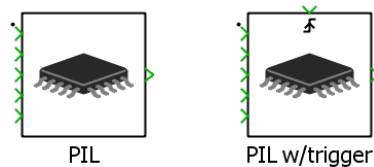
In addition to reading and displaying information from the symbol file, PLECS will also query the target for its identity and check the value against the one stored in the symbol file. This verifies the device settings and ensures that the correct binary file has been selected. Further, the user can request for a target mode change to configure the embedded code to run in **Normal Operation** mode or in **Ready for PIL** mode.

## PIL Block

The PIL block ties a processor into a PLECS simulation by making Override Probes and Read Probes, configured on the target, available as input and out-

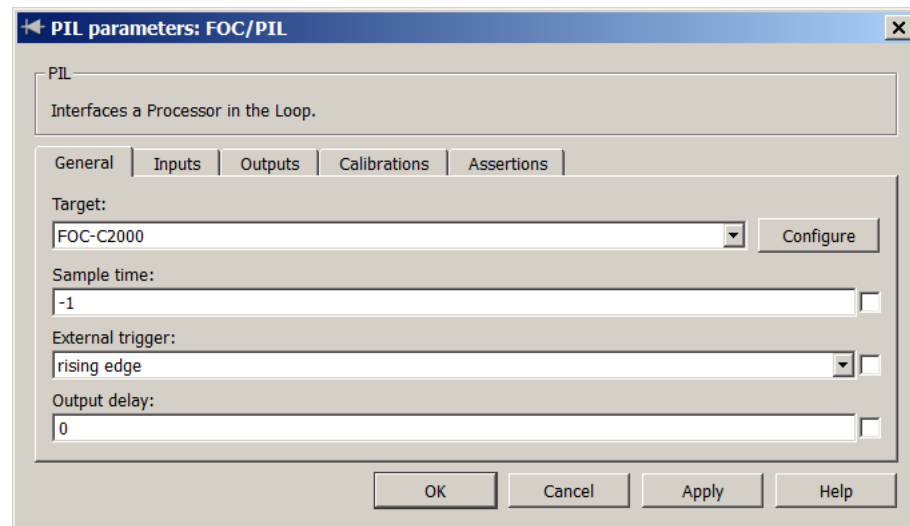


put ports, respectively.



### PIL Block

A PIL block is associated with a target defined in the target manager, which is selected from the **Target** combo box. The **Configure...** button provides a convenient shortcut to the target manager for configuring existing and new targets.



### PIL Block General Tab

The execution of the PIL block can be triggered at a fixed **Discrete-Periodic** rate by configuring the **Sample time** to a positive value. As with other PLECS components, an **Inherited** sample time can be selected by setting the parameter to **-1** or **[-1 0]**.

A trigger port can be enabled using the **External trigger** combo box. This is useful if the control interrupt source is part of the PLECS circuit, such as an

ADC or PWM peripheral model.

Typically, an **Inherited** sample time is used in combination with a trigger port. If a **Discrete-Periodic** rate is specified, the trigger port will be sampled at the specified rate.

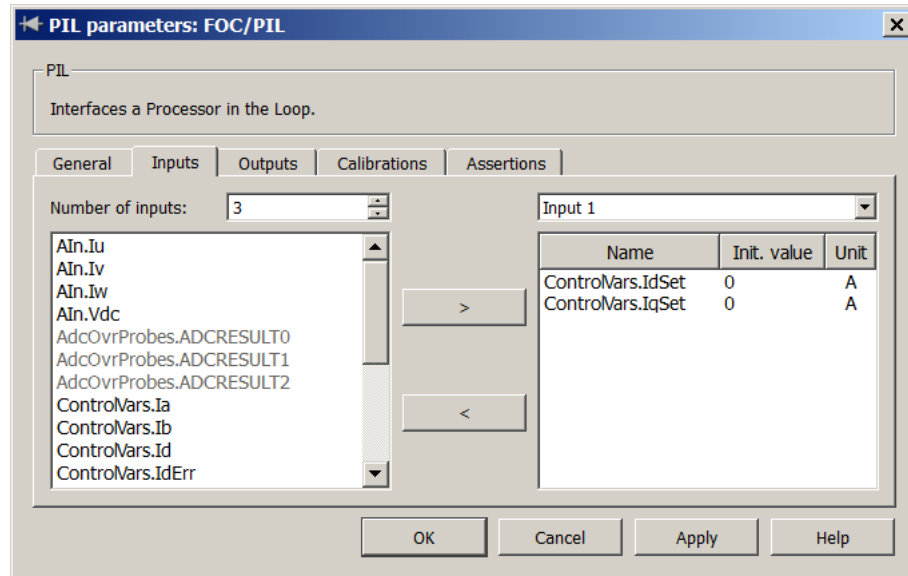
Similar to the DLL block, the **Output delay** setting permits delaying the output of each simulation step to approximate processor calculation time.

---

**Note** Make sure the value for the **Output delay** does not exceed the sample time of the block, or the outputs will never be updated.

---

A delay of **0** is a valid setting, but it will create direct-feedthrough between inputs and outputs.



### PIL Block Inputs Tab

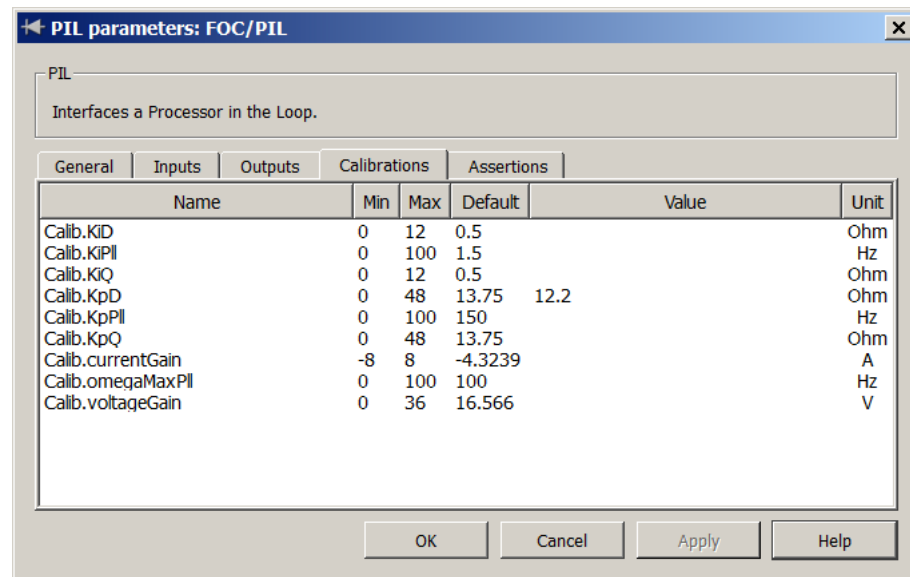
The PIL block extracts the names of Override Probes and Read Probes from the symbol file selected in the target configuration and presents lists for selection as input and output signals, as shown in the figure above.

The number of inputs and outputs of a PIL block is configurable with the **Number of inputs** and **Number of outputs** settings. To associate Override Probes or Read Probes with a given input or output, select an input/output from the combo box on the right half of the dialog. Then drag the desired Override Probes or Read Probes from the left into the area below or add them by selecting them and clicking the > button. To remove an Override Probe or Read Probe, select it and either press the **Delete** key or < button.

**Note** It is possible to multiplex several Override/Read Probe signals into one input/output. The sequence can be reordered by dragging the signals up and down the list.

Starting with PLECS 3.7, the PIL block allows setting initial conditions for Override Probes.

Also new with PLECS 3.7 is the Calibrations tab, which permits modifying embedded code settings such as regulator gains and filter coefficients.



### PIL Block Calibrations Tab

Calibrations can be set in the **Value** column. If no entry is provided, the embedded code will use the default value as indicated in the **Default** column.

# Components by Category

This chapter lists the blocks of the Component library by category.

## System

<b>Configurable Subsystem</b>	Provide subsystem with exchangeable implementations
<b>Display</b>	Display signal values in the schematic
<b>Electrical Ground</b>	Connect to common electrical ground
<b>Electrical Label</b>	Connect electrical potentials by name
<b>Electrical Port</b>	Add electrical connector to subsystem
<b>Enable</b>	Control execution of an atomic subsystem
<b>Manual Signal Switch</b>	Manually select one of two input signals
<b>Pause / Stop</b>	Pause or stop the simulation
<b>Scope</b>	Display simulation results versus time
<b>Signal Demultiplexer</b>	Split vectorized signal
<b>Signal From</b>	Reference signal from Signal Goto block by name
<b>Signal Goto</b>	Make signal available by name
<b>Signal Inport</b>	Add signal input connector to subsystem
<b>Signal Multiplexer</b>	Combine several signals into vectorized signal
<b>Signal Outport</b>	Add signal output connector to subsystem

<b>Signal Selector</b>	Select or reorder elements from vectorized signal
<b>Signal Switch</b>	Select one of two input signals depending on control signal
<b>Subsystem</b>	Create functional entity in hierarchical simulation model
<b>To File</b>	Write time and signal values to file
<b>Trigger</b>	Control execution of an atomic subsystem
<b>Wire Multiplexer</b>	Bundle several wires into bus
<b>Wire Selector</b>	Select or reorder elements from wire bus
<b>XY Plot</b>	Display correlation between two signals

## Assertions

<b>Assert Dynamic Lower Limit</b>	Check whether a signal stays above another signal
<b>Assert Dynamic Range</b>	Check whether a signal stays between two other signals
<b>Assert Dynamic Upper Limit</b>	Check whether a signal stays below another signal
<b>Assertion</b>	Check whether a condition is true
<b>Assert Lower Limit</b>	Check whether a signal stays above a constant
<b>Assert Range</b>	Check whether a signal stays within a constant range
<b>Assert Upper Limit</b>	Check whether a signal stays below a constant

## Control

### Sources

<b>Clock</b>	Provide current simulation time
<b>Constant</b>	Generate constant signal

<b>Initial Condition</b>	Output specified initial value in the first simulation step
<b>Pulse Generator</b>	Generate periodic rectangular pulses
<b>Ramp</b>	Generate constantly rising or falling signal
<b>Sine Wave</b>	Generate time-based sine wave with optional bias
<b>Step</b>	Generate constant signal with instantaneous step change
<b>Triangular Wave Generator</b>	Generate periodic triangular or sawtooth waveform
<b>White Noise</b>	Generate normally distributed random numbers
<b>Random Numbers</b>	Generate uniformly distributed random numbers

## Math

<b>Abs</b>	Calculate absolute value of input signal
<b>Algebraic Constraint</b>	Enforce an algebraic constraint
<b>Data Type</b>	Cast the input signal to the specified data type
<b>Gain</b>	Multiply input signal by constant
<b>Math Function</b>	Apply specified mathematical function
<b>Minimum / Maximum</b>	Output input signal with highest resp. lowest value
<b>Offset</b>	Add constant to input signal
<b>Product</b>	Multiply and divide scalar or vectorized input signals
<b>Rounding</b>	Round floating point signal to integer values
<b>Signum</b>	Provide sign of input signal
<b>Sum</b>	Add and subtract input signals
<b>Trigonometric Function</b>	Apply specified trigonometric function

## Continuous

<b>Integrator</b>	Integrate input signal with respect to time
<b>State Space</b>	Implement linear time-invariant system as state-space model
<b>Transfer Function</b>	Model linear time-invariant system as transfer function

## **Delays**

<b>Memory</b>	Provide input signal from previous major time step
<b>Pulse Delay</b>	Delay discrete-value input signal by fixed time
<b>Transport Delay</b>	Delay continuous input signal by fixed time
<b>Turn-on Delay</b>	Delay rising flank of input pulses by fixed dead time

## **Discontinuous**

<b>Comparator</b>	Compare two input signals with minimal hysteresis
<b>Dead Zone</b>	Output zero while input signal is within dead zone limits
<b>Hit Crossing</b>	Detect when signal reaches or crosses given value
<b>Quantizer</b>	Apply uniform quantization to input signal
<b>Rate Limiter</b>	Limit rising and falling rate of change
<b>Relay</b>	Toggle between on- and off-state with configurable threshold
<b>Saturation</b>	Limit input signal to upper and/or lower value

## **Discrete**

<b>Delay</b>	Delay input signal by given number of samples
<b>Discrete Fourier Transform</b>	Perform discrete Fourier transform on input signal



<b>Discrete Integrator</b>	Calculate discrete integral of input signal
<b>Discrete Mean Value</b>	Calculate running mean value of input signal
<b>Discrete RMS Value</b>	Calculate root mean square (RMS) value of input signal
<b>Discrete State Space</b>	Implement discrete time-invariant system as state-space model
<b>Discrete Total Harmonic Distortion</b>	Calculate total harmonic distortion (THD) of input signal
<b>Discrete Transfer Function</b>	Model discrete system as transfer function
<b>Zero-Order Hold</b>	Sample and hold input signal periodically

## Filters

<b>Moving Average</b>	Continuously average input signal over specified time period
<b>Periodic Average</b>	Periodically average input signal over specified time
<b>Periodic Impulse Average</b>	Periodically average Dirac impulses over specified time

## Functions & Tables

<b>1D Look-Up Table</b>	Compute piece-wise linear function of one input signal
<b>2D Look-Up Table</b>	Compute piece-wise linear function of two input signals
<b>3D Look-Up Table</b>	Compute piece-wise linear function of three input signals
<b>C-Script</b>	Execute custom C code
<b>DLL</b>	Interface with externally generated dynamic-link library

<b>Fourier Series</b>	Synthesize periodic output signal from Fourier coefficients
<b>Function</b>	Apply arbitrary arithmetic expression to scalar or vectorized input signal

## Logical

<b>Combinatorial Logic</b>	Use binary input signals to select one row from truth table
<b>Compare to Constant</b>	Compare input signal to constant threshold
<b>D Flip-flop</b>	Implement edge-triggered flip-flop
<b>Edge Detection</b>	Detect edges of pulse signal in given direction
<b>JK Flip-flop</b>	Implement edge-triggered JK flip-flop
<b>Logical Operator</b>	Combine input signals logically
<b>Monoflop</b>	Generate pulse of specified width when triggered
<b>Relational Operator</b>	Compare two input signals
<b>SR Flip-flop</b>	Implement set-reset flip-flop

## Modulators

<b>2-Pulse Generator</b>	Generate firing pulses for H-bridge thyristor rectifier
<b>3-Phase Overmodulation</b>	Extend linear range of modulation index for 3-phase inverters
<b>6-Pulse Generator</b>	Generate firing pulses for 3-phase thyristor rectifier
<b>Blanking Time</b>	Generate commutation delay for 2-level inverter bridges
<b>Blanking Time (3-Level)</b>	Generate commutation delay for 3-level inverter bridges
<b>Peak Current Controller</b>	Implement peak current mode control

<b>Sawtooth PWM</b>	Generate PWM signal using sawtooth carrier
<b>Sawtooth PWM (3-Level)</b>	Generate 3-level PWM signal using sawtooth carriers
<b>Space Vector PWM</b>	Generate PWM signals for 3-phase inverter using space-vector modulation
<b>Space Vector PWM (3-Level)</b>	Generate PWM signals for 3-phase NPC inverter using space-vector modulation
<b>Symmetrical PWM</b>	Generate PWM signal using symmetrical triangular carrier
<b>Symmetrical PWM (3-Level)</b>	Generate 3-level PWM signal using symmetrical triangular carriers

## Transformations

<b>Polar to Rectangular</b>	Convert polar coordinates to Cartesian coordinates
<b>Rectangular to Polar</b>	Convert Cartesian coordinates to polar coordinates
<b>Transformation 3ph-&gt;RRF</b>	Transform 3-phase signal to rotating reference frame
<b>Transformation 3ph-&gt;SRF</b>	Transform 3-phase signal to stationary reference frame
<b>Transformation RRF-&gt;3ph</b>	Transform vector in rotating reference frame into 3-phase signal
<b>Transformation RRF-&gt;SRF</b>	Transform vector from rotating to stationary reference frame
<b>Transformation SRF-&gt;3ph</b>	Transform vector in stationary reference frame into 3-phase signal
<b>Transformation SRF-&gt;RRF</b>	Transform vector from stationary to rotating reference frame

## State Machine

<b>State Machine</b>	Model a state machine
----------------------	-----------------------

## Small Signal Analysis

*(PLECS Standalone only)*

<b>Small Signal Gain</b>	Measure loop gain of closed control loop using small-signal analysis
<b>Small Signal Perturbation</b>	Generate perturbation signal for small-signal analysis
<b>Small Signal Response</b>	Measure system response for small-signal analysis

## Electrical

### Sources

<b>Current Source (Controlled)</b>	Generate variable current
<b>Current Source AC</b>	Generate sinusoidal current
<b>Current Source DC</b>	Generate constant current
<b>Voltage Source (Controlled)</b>	Generate variable voltage
<b>Voltage Source AC</b>	Generate sinusoidal voltage
<b>Voltage Source AC (3-Phase)</b>	Generate 3-phase sinusoidal voltage
<b>Voltage Source DC</b>	Generate constant voltage

### Meters

<b>Ammeter</b>	Output measured current as signal
<b>Meter (3-Phase)</b>	Measure voltages and currents of 3-phase system
<b>Voltmeter</b>	Output measured voltage as signal

### Passive Components

<b>Capacitor</b>	Ideal capacitor
------------------	-----------------

<b>Electrical Algebraic Component</b>	Enforce an algebraic constraint in terms of voltage and current
<b>Inductor</b>	Ideal inductor
<b>Mutual Inductor</b>	Ideal mutual inductor
<b>Mutual Inductance (2 Windings)</b>	Magnetic coupling between two lossy windings
<b>Mutual Inductance (3 Windings)</b>	Magnetic coupling between three lossy windings
<b>Pi-Section Line</b>	Single-phase pi-section transmission line
<b>Piece-wise Linear Resistor</b>	Resistance defined by voltage-current pairs
<b>Resistor</b>	Ideal resistor
<b>Saturable Capacitor</b>	Capacitor with piece-wise linear saturation
<b>Saturable Inductor</b>	Inductor with piece-wise linear saturation
<b>Transmission Line (3ph)</b>	3-phase transmission line
<b>Variable Capacitor</b>	Capacitance controlled by signal
<b>Variable Inductor</b>	Inductance controlled by signal
<b>Variable Resistor</b>	Resistance controlled by signal
<b>Variable Resistor with Constant Capacitor</b>	Controlled resistance in parallel with constant capacitance
<b>Variable Resistor with Constant Inductor</b>	Controlled resistance in series with constant inductance
<b>Variable Resistor with Variable Capacitor</b>	Controlled resistance in parallel with controlled capacitance
<b>Variable Resistor with Variable Inductor</b>	Controlled resistance in series with controlled inductance

## **Power Semiconductors**

<b>Diode</b>	Ideal diode with optional forward voltage and on-resistance
<b>Diode with Reverse Recovery</b>	Dynamic diode model with reverse recovery

<b>GTO</b>	Ideal GTO with optional forward voltage and on-resistance
<b>GTO (Reverse Conducting)</b>	Ideal GTO with ideal anti-parallel diode
<b>IGBT</b>	Ideal IGBT with optional forward voltage and on-resistance
<b>IGBT with Diode</b>	Ideal IGBT with ideal anti-parallel diode
<b>IGBT with Limited di/dt</b>	Dynamic IGBT model with finite current slopes during turn-on and turn-off
<b>IGCT (Reverse Blocking)</b>	Ideal IGCT with optional forward voltage and on-resistance
<b>IGCT (Reverse Conducting)</b>	Ideal IGCT with ideal anti-parallel diode
<b>MOSFET</b>	Ideal MOSFET with optional on-resistance
<b>MOSFET with Diode</b>	Ideal MOSFET with ideal anti-parallel diode
<b>MOSFET with Limited di/dt</b>	Dynamic MOSFET model with finite current slopes during turn-on and turn-off
<b>Thyristor</b>	Ideal thyristor (SCR) with optional forward voltage and on-resistance
<b>Thyristor with Reverse Recovery</b>	Dynamic thyristor (SCR) model with reverse recovery
<b>TRIAC</b>	Ideal TRIAC with optional forward voltage and on-resistance
<b>Zener Diode</b>	Zener diode with controlled reverse breakdown voltage

## **Power Modules**

<b>IGBT Chopper (Low-Side Switch)</b>	Chopper used in boost converters.
<b>IGBT Chopper (High-Side Switch)</b>	Chopper used in buck converters.
<b>IGBT Chopper (Low-Side Switch with Reverse Diode)</b>	Chopper used in boost converters.

**IGBT Chopper (High-Side Switch with Reverse Diode)**

Chopper used in buck converters.

**IGBT Half Bridge**

Single leg of a 2-level voltage source inverter.

**IGBT 3-Level Half Bridge (NPC)**

Single leg of a 3-level neutral-point clamped voltage source inverter.

**IGBT Half Bridges (Low-/High-Side Connected)**

Series-connected inverter cells for modular multi-level converters.

**IGBT Full Bridges (Series Connected)**

Series-connected inverter cells for modular multi-level converters.

## Switches

**Breaker**

AC circuit breaker opening at zero current

**Double Switch**

Changeover switch with two positions

**Manual Double Switch**

Manual changeover switch with two positions

**Manual Switch**

Manual on-off switch

**Manual Triple Switch**

Manual changeover switch with three 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 (Simple)**

Simplified model of brushless DC machine excited by permanent magnets

**DC Machine**

Simple model of DC machine

**Induction Machine (Slip Ring)**

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

**Non-Excited Synchronous Machine**

Non-excited synchronous machine configurable with lookup tables

**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

**Synchronous Reluctance Machine**

Synchronous reluctance machine configurable with lookup tables



## Converters

<b>Diode Rectifier (3ph)</b>	3-phase diode rectifier
<b>Ideal 3-Level Converter (3ph)</b>	Switch-based 3-phase 3-level converter
<b>Ideal Converter (3ph)</b>	Switch-based 3-phase converter
<b>IGBT 3-Level Converter (3ph)</b>	3-phase 3-level neutral-point clamped IGBT converter
<b>IGBT Converter (3ph)</b>	3-phase IGBT converter
<b>MOSFET Converter (3ph)</b>	3-phase MOSFET converter
<b>Thyristor Rectifier/Inverter</b>	3-phase thyristor rectifier/inverter

## Electronics

<b>Op-Amp</b>	Ideal operational amplifier with finite gain
<b>Op-Amp with Limited Output</b>	Ideal operational amplifier with limited output voltage

## Model Settings

<b>Electrical Model Settings</b>	Configure settings for an individual electrical circuit
----------------------------------	---

## Thermal

<b>Ambient Temperature</b>	Connect to Heat Sink on which component is placed
<b>Constant Heat Flow</b>	Generate constant heat flow
<b>Constant Temperature</b>	Provide constant temperature
<b>Controlled Heat Flow</b>	Generate variable heat flow
<b>Controlled Temperature</b>	Provide variable temperature

<b>Heat Flow Meter</b>	Output measured heat flow as signal
<b>Heat Sink</b>	Isotherm environment for placing components
<b>Thermal Capacitor</b>	Thermal capacitance of piece of material
<b>Thermal Chain</b>	Thermal impedance implemented as RC chain
<b>Thermal Ground</b>	Connect to common reference temperature
<b>Thermal Model Settings</b>	Configure settings for an individual thermal system
<b>Thermal Port</b>	Add thermal connector to subsystem
<b>Thermal Resistor</b>	Thermal resistance of piece of material
<b>Thermometer</b>	Output measured temperature as signal

## **Magnetic**

<b>Winding</b>	Ideal winding defining an electro-magnetic interface
<b>Magnetic Permeance</b>	Linear magnetic permeance
<b>Linear Core</b>	Linear magnetic core element
<b>Air Gap</b>	Air gap in a magnetic core
<b>Leakage Flux Path</b>	Permeance of linear leakage flux path
<b>Saturable Core</b>	Magnetic core element with saturation
<b>Hysteretic Core</b>	Magnetic core element with static hysteresis
<b>Variable Magnetic Permeance</b>	Variable permeance controlled by external signal
<b>Magnetic Resistance</b>	Effective magnetic resistance for modeling losses
<b>MMF Meter</b>	Output the measured magneto-motive force
<b>Flux Rate Meter</b>	Output the measured rate-of-change of magnetic flux
<b>MMF Source (Constant)</b>	Generate a constant magneto-motive force
<b>MMF Source (Controlled)</b>	Generate a variable magneto-motive force

**Magnetic Port**

Add magnetic connector to subsystem

**Mechanical****Translational****Sources****Force (Constant)**

Generate constant force

**Force (Controlled)**

Generate variable force

**Translational Speed (Constant)**

Maintain constant linear speed

**Translational Speed (Controlled)**

Maintain variable linear speed

**Sensors****Force Sensor**

Output measured force as signal

**Translational Speed Sensor**

Output measured linear speed as signal

**Position Sensor**

Output measured absolute or relative position as signal

**Components****Translational Reference**

Connect to translational reference frame

**Translational Port**

Add translational flange to subsystem

**Mass**

Model sliding mass with inertia

**Translational Spring**

Ideal translational spring

**Translational Damper**

Ideal viscous translational damper

**Translational Clutch**

Ideal translational clutch

**Translational Backlash**

Ideal translational backlash

<b>Translational Hard Stop</b>	Ideal translational single- or double-sided hard stop
<b>Translational Friction</b>	Ideal translational stick/slip friction
<b>Rack and Pinion</b>	Ideal conversion between translational and rotational motion
<b>Translational Algebraic Component</b>	Define an algebraic constraint in terms of force and speed

### Model Settings

<b>Translational Model Settings</b>	Configure settings for an individual mechanical system
-------------------------------------	--

## Rotational

### Sources

<b>Torque (Constant)</b>	Generate constant torque
<b>Torque (Controlled)</b>	Generate variable torque
<b>Rotational Speed (Constant)</b>	Maintain constant angular speed
<b>Rotational Speed (Controlled)</b>	Maintain variable angular speed

### Sensors

<b>Torque Sensor</b>	Output measured torque as signal
<b>Rotational Speed Sensor</b>	Output measured angular speed as signal
<b>Angle Sensor</b>	Output measured absolute or relative angle as signal

### Components

<b>Rotational Reference</b>	Connect to rotational reference frame
<b>Rotational Port</b>	Add rotational flange to subsystem
<b>Inertia</b>	Model rotating body with inertia
<b>Torsion Spring</b>	Ideal torsion spring
<b>Rotational Damper</b>	Ideal viscous rotational damper
<b>Rotational Clutch</b>	Ideal rotational clutch
<b>Rotational Backlash</b>	Ideal rotational backlash
<b>Rotational Hard Stop</b>	Ideal rotational single- or double-sided hard stop
<b>Rotational Friction</b>	Ideal rotational stick/slip friction
<b>Gear</b>	Ideal gear
<b>Planetary Gear Set</b>	Ideal planetary gear set
<b>Rack and Pinion</b>	Ideal conversion between translational and rotational motion
<b>Rotational Algebraic Component</b>	Define an algebraic constraint in terms of torque and angular speed

### Model Settings

<b>Rotational Model Settings</b>	Configure settings for an individual mechanical system
----------------------------------	--

## Additional Simulink Blocks

*(PLECS Blockset only)*

<b>AC Sweep</b>	Perform AC sweep
<b>Impulse Response Analysis</b>	Perform impulse response analysis
<b>Loop Gain Analysis</b>	Determine loop gain of closed control loop
<b>Steady-State Analysis</b>	Determine periodic steady-state operating point
<b>Timer</b>	Generate piece-wise constant signal



# Component Reference

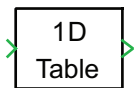
This chapter lists the contents of the Component library in alphabetical order.

## 1D Look-Up Table

**Purpose** Compute piece-wise linear function of one input signal

**Library** Control / Functions & Tables

### Description



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.

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. Provided zero-crossing signals are not active (see “Locate discontinuities” below) this behavior can be overridden by defining *three* output values for the same input value; in this case the middle output value will be chosen.

Use the 2D Look-Up Table block (see page 308) to map two input signals to an output signal.

### Parameters

#### 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.

#### 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.

#### Locate discontinuities

When set to on, the Look-Up Table defines zero-crossing signals that prevent a variable-step solver from inadvertently stepping over a discontinuity of the input-output mapping or its derivative.

Note that the use of zero-crossing signals forces the mapping to be left- or right-continuous. Therefore, it is not possible to specify an arbitrary output value at a step discontinuity: if three output values are defined for the



same input, the middle output value is neglected. Moreover, if the step discontinuity is at input value 0, no averaging is performed: the first output value corresponding to 0 is selected instead.

When set to off, the Look-Up Table does not explicitly influence the steps taken by a variable-step solver.

**Probe Signals****Input**

The block input signal.

**Output**

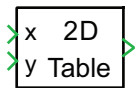
The block output signal.

## 2D Look-Up Table

**Purpose** Compute piece-wise linear function of two input signals

**Library** Control / Functions & Tables

### Description



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 *rows* of the output matrix, the input vector  $y$ , to the *columns*.

The output value is interpolated or extrapolated from the block parameters using the technique described for the 1D Look-Up Table block (see page 306).

### Parameters

#### 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.

#### 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.

#### 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.

#### Locate discontinuities

When set to on, the Look-Up Table defines zero-crossing signals that prevent a variable-step solver from inadvertently stepping over a discontinuity of the input-output mapping or its derivative.

When set to off, the Look-Up Table does not explicitly influence the steps taken by a variable-step solver.

### Probe Signals

#### Input $x$

The block input signal  $x$ .

#### Input $y$

The block input signal  $y$ .

#### Output

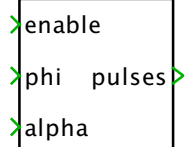
The block output signal.

## 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  $\varphi$  (produced e.g. by a PLL), and the firing angle alpha.

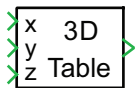


## 3D Look-Up Table

**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 306).

### 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.

#### Locate discontinuities

When set to on, the Look-Up Table defines zero-crossing signals that prevent a variable-step solver from inadvertently stepping over a discontinuity of the input-output mapping or its derivative.

When set to off, the Look-Up Table does not explicitly influence the steps taken by a variable-step solver.

### Probe Signals

#### Input $x$

The block input signal  $x$ .

#### Input $y$

The block input signal  $y$ .

**Input  $z$** 

The block input signal  $z$ .

**Output**

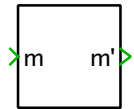
The block output signal.

## 3-Phase Overmodulation

**Purpose** Extend linear range of modulation index for 3-phase inverters

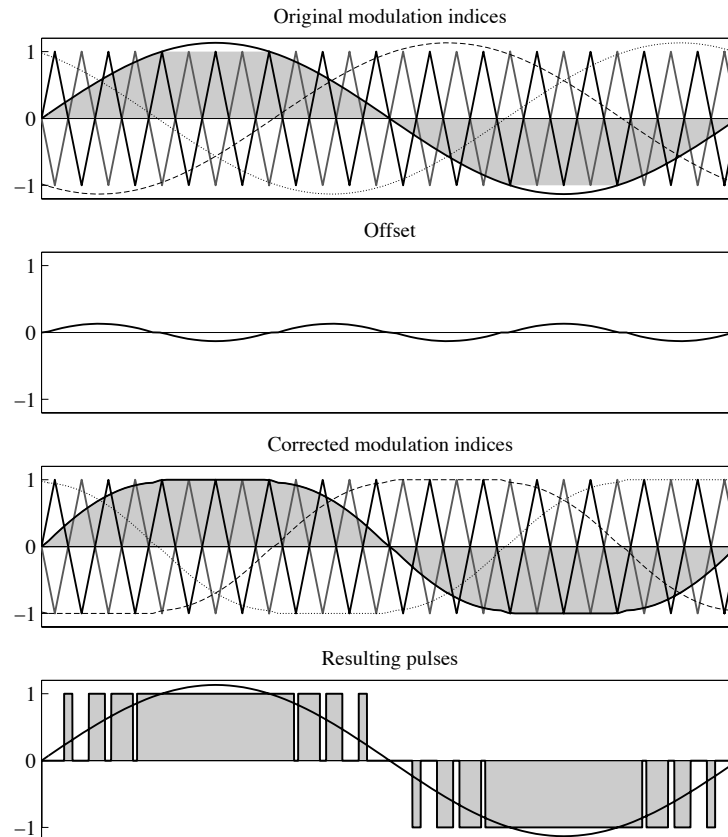
**Library** Control / Modulators

**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 426).

The figures below illustrates the working principle of the 3-Phase Overmodulation block in conjunction with the Symmetrical PWM (see page 612).

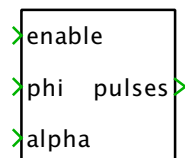


## 6-Pulse Generator

**Purpose** Generate firing pulses for 3-phase thyristor rectifier

**Library** Control / Modulators

### Description



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  $\varphi$  (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

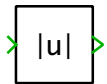
“Double pulses” enables a second firing pulse for each thyristor.

## 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.

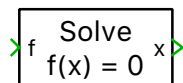


## Algebraic Constraint

**Purpose** Enforce an algebraic constraint by solving a system of equations

**Library** Control / Math

### Description



The Algebraic Constraint block outputs the value that produces zero at the input. The output must affect the input by means of a direct feedthrough path. The block ensures that the input signal is zero at all times.

The direct feedthrough path defines a function  $f$  of the output signal  $x$ . The Algebraic Constraint block solves the system of equations  $f(x) = 0$ . The input and output signals must have the same width, i.e. the number of equations must equal the number of unknowns.

---

**Note** The Algebraic Constraint block creates an algebraic loop. See section “Block Sorting” (on page 29) for more information on algebraic loops.

---

### Parameters

#### Initial guess

A guess of the solution  $x$  at the start of the simulation. The parameter can be either a scalar or a vector with the same width as the output and input signals. If a scalar is specified, the value is used as an initial guess for all components of the output signal. The default is 0.

### Probe Signals

#### Residual

The constraint residual  $f$ .

#### Solution

The computed solution  $x$ .

## 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 120).

---

**Note** Ambient Temperature blocks may not be used in schematics that contain Thermal Port blocks (see page 635).

---

## 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  $\Phi$  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

#### Cross-sectional area

Effective cross-sectional area  $A$  of the air gap, in  $\text{m}^2$ .

#### Length of flux path

Effective length  $l$  of the air gap, in m.

#### 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 air gap, in A/m.

#### Flux density

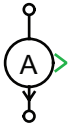
The magnetic flux density  $B$  in the air gap, in teslas (T).

## Ammeter

**Purpose** Output measured current as signal

**Library** Electrical / Meters

**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 583) 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 Signal**

**Measured current**

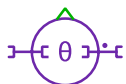
The measured current in amperes (A).

## Angle Sensor

**Purpose** Output measured absolute or relative angle as signal

**Library** Mechanical / Rotational / Sensors

### Description



The Angle Sensor measures the relative angle of the flange marked with a dot with respect to the other flange. If the other flange is connected to the reference frame, the absolute angle is measured.

---

**Note** Speed and angle sensors are ideally compliant. Hence, if multiple speed or angle sensors are connected in series the speed or angle measured by an individual sensor is undefined. This produces a run-time error.

---

### Parameters

#### Second flange

Controls whether the second flange is accessible or connected to the rotational reference frame.

#### Initial angle

The angle at simulation start, in radians.

### Probe Signal

#### Angle

The measured angle, in radians.

## Assert Dynamic Lower Limit

**Purpose** Issue a warning or an error message when the input exceeds the specified dynamic lower limit

**Library** Assertions

**Description** This block checks whether the middle input signal lies above the other input signal. When the check fails, a message is added to the diagnostics window and the simulation may be paused or stopped.



### Parameters

#### Limit

Either include or exclude, indicating whether the limit is included in or excluded from the allowed range.

#### Action

One of the following: ignore: the assertion is ignored; warning: when the assertion fails, a warning is added to the diagnostics window; warning/pause: when the assertion fails, a warning is added to the diagnostics window and the simulation is paused; error: when the assertion fails, an error is added to the diagnostics window and the simulation is stopped.

This parameter can be overwritten on a per model basis (see “Simulation Parameters” on page 103). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).

#### Message

The message that is displayed in the diagnostics window when the assertion fails.

### Probe Signals

#### Input

The middle input signal.

#### Assertion value

1 while the input is above the lower limit and 0 otherwise.

#### Lower limit

The lower limit input signal.

## Assert Dynamic Range

**Purpose** Issue a warning or an error message when the input leaves the specified dynamic range

**Library** Assertions

**Description** This block checks whether the middle input signal lies between the two other input signals. When the check fails, a message is added to the diagnostics window and the simulation may be paused or stopped.



### Parameters

#### Limits

Either include or exclude, indicating whether the limits are included in or excluded from the allowed range.

#### Action

One of the following: ignore: the assertion is ignored; warning: when the assertion fails, a warning is added to the diagnostics window; warning/pause: when the assertion fails, a warning is added to the diagnostics window and the simulation is paused; error: when the assertion fails, an error is added to the diagnostics window and the simulation is stopped.

This parameter can be overwritten on a per model basis (see “Simulation Parameters” on page 103). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).

#### Message

The message that is displayed in the diagnostics window when the assertion fails.

### Probe Signals

#### Input

The middle input signal.

#### Assertion value

1 while the input is within the range and 0 otherwise.

#### Upper limit

The upper limit input signal.

#### Lower limit

The lower limit input signal.

## Assert Dynamic Upper Limit

**Purpose** Issue a warning or an error message when the input exceeds the specified dynamic upper limit

**Library** Assertions

**Description** This block checks whether the middle input signal lies below the other input signal. When the check fails, a message is added to the diagnostics window and the simulation may be paused or stopped.



### Parameters

#### Limit

Either include or exclude, indicating whether the limit is included in or excluded from the allowed range.

#### Action

One of the following: ignore: the assertion is ignored; warning: when the assertion fails, a warning is added to the diagnostics window; warning/pause: when the assertion fails, a warning is added to the diagnostics window and the simulation is paused; error: when the assertion fails, an error is added to the diagnostics window and the simulation is stopped.

This parameter can be overwritten on a per model basis (see “Simulation Parameters” on page 103). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).

#### Message

The message that is displayed in the diagnostics window when the assertion fails.

### Probe Signals

#### Input

The middle input signal.

#### Assertion value

1 while the input is below the upper limit and 0 otherwise.

#### Upper limit

The upper limit input signal.

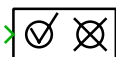


## Assertion

**Purpose** Issue a warning or an error message when the input becomes 0

**Library** Assertions

### Description



The assertion block checks whether a condition stays true during the simulation. While the input signal is non-zero, the block does nothing. When the input signal becomes zero, the specified message appears. Depending on the “action” setting, the message is added either as a warning or as an error to the diagnostics window. If it is added as a warning, it is possible to additionally automatically pause the simulation. If it is added as an error, the simulation always stops. To disable the assertion, set its action to ignore.

### Parameters

#### Action

One of the following: ignore: the assertion is ignored; warning: when the assertion fails, a warning is added to the diagnostics window; warning/pause: when the assertion fails, a warning is added to the diagnostics window and the simulation is paused; error: when the assertion fails, an error is added to the diagnostics window and the simulation is stopped.

This parameter can be overwritten on a per model basis (see “Simulation Parameters” on page 103). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).


#### Message

The message that is displayed in the diagnostics window when the assertion fails.


#### Highlight level

The number of levels the highlight is propagated upwards in the component hierarchy when the assertion fails. A highlight level of 0 means that the assertion block itself will be highlighted when the assertion fails. A highlight level of 1 means that the component containing the assertion block will be highlighted when the assertion fails, etc.

## Assert Lower Limit

<b>Purpose</b>	Issue a warning or an error message when the input exceeds the specified lower limit
<b>Library</b>	Assertions
<b>Description</b>	This block checks whether the input signal lies above the specified lower limit. When the check fails, a message is added to the diagnostics window and the simulation may be paused or stopped.
	
<b>Parameters</b>	<p><b>Lower limit</b> A constant specifying the lower limit for the input signal.</p> <p><b>Limit</b> Either <code>include</code> or <code>exclude</code>, indicating whether the limit is included in or excluded from the allowed range.</p> <p><b>Action</b> One of the following: <code>ignore</code>: the assertion is ignored; <code>warning</code>: when the assertion fails, a warning is added to the diagnostics window; <code>warning/pause</code>: when the assertion fails, a warning is added to the diagnostics window and the simulation is paused; <code>error</code>: when the assertion fails, an error is added to the diagnostics window and the simulation is stopped.</p> <p>This parameter can be overwritten on a per model basis (see “Simulation Parameters” on page 103). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).</p> <p><b>Message</b> The message that is displayed in the diagnostics window when the assertion fails.</p>
<b>Probe Signals</b>	<p><b>Input</b> The middle input signal.</p> <p><b>Assertion value</b> 1 while the input is above the lower limit and 0 otherwise.</p>

## Assert Range

<b>Purpose</b>	Issue a warning or an error message when the input leaves the specified range
<b>Library</b>	Assertions
<b>Description</b>	This block checks whether the input signal lies between the specified lower and upper limits. When the check fails, a message is added to the diagnostics window and the simulation may be paused or stopped.
	
<b>Parameters</b>	<p><b>Upper limit</b> A constant specifying the upper limit for the input signal.</p> <p><b>Lower limit</b> A constant specifying the lower limit for the input signal.</p> <p><b>Limits</b> Either include or exclude, indicating whether the limits are included in or excluded from the allowed range.</p> <p><b>Action</b> One of the following: ignore: the assertion is ignored; warning: when the assertion fails, a warning is added to the diagnostics window; warning/pause: when the assertion fails, a warning is added to the diagnostics window and the simulation is paused; error: when the assertion fails, an error is added to the diagnostics window and the simulation is stopped.</p> <p>This parameter can be overwritten on a per model basis (see “Simulation Parameters” on page 103). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).</p> <p><b>Message</b> The message that is displayed in the diagnostics window when the assertion fails.</p>
<b>Probe Signals</b>	<p><b>Input</b> The middle input signal.</p> <p><b>Assertion value</b> 1 while the input is within the range and 0 otherwise.</p>

## Assert Upper Limit

**Purpose** Issue a warning or an error message when the input exceeds the specified upper limit

**Library** Assertions

**Description** This block checks whether the input signal lies below the specified upper limit. When the check fails, a message is added to the diagnostics window and the simulation may be paused or stopped.



### Parameters

#### Upper limit

A constant specifying the upper limit for the input signal.

#### Limit

Either `include` or `exclude`, indicating whether the limit is included in or excluded from the allowed range.

#### Action

One of the following: `ignore`: the assertion is ignored; `warning`: when the assertion fails, a warning is added to the diagnostics window; `warning/pause`: when the assertion fails, a warning is added to the diagnostics window and the simulation is paused; `error`: when the assertion fails, an error is added to the diagnostics window and the simulation is stopped.

This parameter can be overwritten on a per model basis (see “Simulation Parameters” on page 103). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” on page 85).

#### Message

The message that is displayed in the diagnostics window when the assertion fails.

### Probe Signals

#### Input

The middle input signal.

#### Assertion value

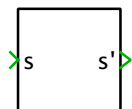
1 while the input is below the upper limit and 0 otherwise.

## 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 612). 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

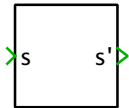
The delay in seconds (s) between the turn-off of one switch and the turn-on of the other switch in an inverter leg.

## 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 614). 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

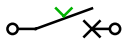
The delay in seconds ( $s$ ) between the turn-off of one switch and the turn-on of another switch in an inverter leg.

## 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

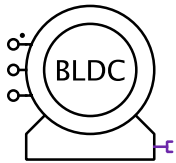
Conduction state of the internal switch. The signal outputs 0 if the breaker is open, and 1 if it is closed.

## Brushless DC Machine

**Purpose** Detailed model of brushless DC machine excited by permanent magnets

**Library** Electrical / Machines

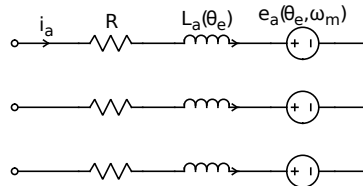
### Description



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  $\omega_m$ . The shape function in turn is expressed as a fourier series of the electrical rotor angle  $\theta_e$ :

$$e_x(\theta_e, \omega_m) = k_{e,x}(\theta_e) \cdot \omega_m$$

$$k_{e,a}(\theta_e) = \sum_n K_{c,n} \cos(n\theta_e) + K_{s,n} \sin(n\theta_e)$$

$$k_{e,b}(\theta_e) = \sum_n K_{c,n} \cos\left(n\theta_e - \frac{2\pi n}{3}\right) + K_{s,n} \sin\left(n\theta_e - \frac{2\pi n}{3}\right)$$

$$k_{e,c}(\theta_e) = \sum_n K_{c,n} \cos\left(n\theta_e + \frac{2\pi n}{3}\right) + K_{s,n} \sin\left(n\theta_e + \frac{2\pi n}{3}\right)$$



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:

$$T_{\text{cog}}(\theta_e) = \sum_n T_{c,n} \cos(n\theta_e) + T_{s,n} \sin(n\theta_e)$$

### Mechanical System

Mechanical rotor speed:

$$\dot{\omega}_m = \frac{1}{J} (T_e + T_{\text{cog}}(\theta_e) - F\omega_m - T_m)$$

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 ( $\Omega$ ).

### 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  $\text{Nms}^2$ .

**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 ( $\text{s}^{-1}$ ).

**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).

**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  $\omega_m$  of the rotor in radians per second ( $\text{s}^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  in radians.

**Electrical torque**

The electrical torque  $T_e$  of the machine in Nm.

**Cogging torque**

The cogging torque  $T_{\text{cog}}$  of the machine in Nm.

**References**

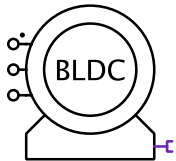
- D. Hanselman, "Brushless permanent magnet motor design, 2nd ed.", The Writers' Collective, Mar. 2003.
- P. Pillay, R. Krishnan, "Modeling, simulation, and analysis of permanent-magnet motor drives, Part II: The brushless DC motor drive", IEEE Trans. on Ind. App., Vol. 25, No. 2, Mar./Apr. 1989.

## Brushless DC Machine (Simple)

**Purpose** Simple model of brushless DC machine excited by permanent magnets

**Library** Electrical / Machines

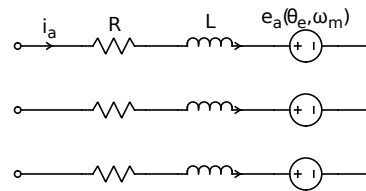
### Description



The simplified Brushless DC Machine is a model of a permanent magnet synchronous machine with sinusoidal or trapezoidal back EMF.

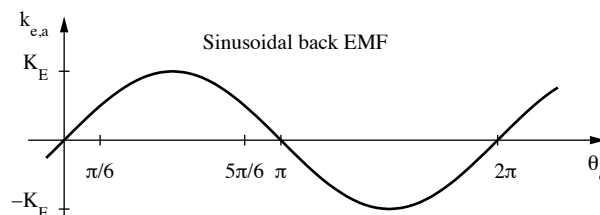
The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. In the component icon, phase a of the stator winding is marked with a dot.

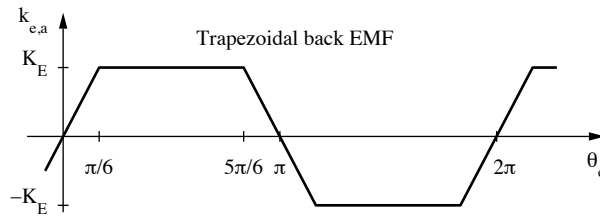
### Electrical System



The back EMF voltages are determined by a shape function  $k_e$  and the mechanical rotor speed  $\omega_m$ . 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:

$$\begin{aligned} \dot{\theta}_m &= \omega_m \\ \theta_e &= p \cdot \theta_m \end{aligned}$$

### 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 ( $\Omega$ ).

#### Stator inductance

The stator inductance  $L - M$  in henries (H).

#### Inertia

Combined rotor and load inertia  $J$  in  $\text{Nms}^2$ .

#### 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 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).

**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  $\Psi_d$  and  $\Psi_q$  in the stationary reference frame in Vs.

**Rotational speed**

The rotational speed  $\omega_m$  of the rotor in radians per second ( $s^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  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 330). 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 518). 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

› C-Script ›

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 195).

The C-Script dialog consists of two tabbed panes that are described below.

## Setup

### Number of inputs, outputs

A positive integer or a vector of positive integers. The length of the vector specifies the number of input or output terminals, the elements in the vector specify the widths of the single terminals. For dynamic sizing set the width of one or more input terminals to -1; the width will then be determined at simulation start depending on the number of elements in the signal that is connected to the input port. All occurrences of -1 in the input and output widths and any other data vector will be expanded to the same width.

### Number of cont. states, disc. states, zero-crossings

A positive or zero integer specifying the sizes of the different data vectors (i.e. continuous and discrete state variables, and zero-crossing signals) that the C-Script registers with the solver.

### 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 200).

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). If the C-Script block has one input terminal, a flag in the vector is applied to the respective single element of the input signal; if the block has multiple input terminals, a flag is applied to the whole signal of the respective terminal. You can also specify a single scalar, which then applies to all input signals of all terminals.

**Language standard**

The language standard used by the compiler. Possible values are C90, C99 and C11. The default is C99.

**Enable GNU extensions**

If this box is checked, the compiler enables GNU C language features not found in ISO standard C. These extensions are disabled by default. For backward compatibility, the extensions are enabled in models saved with PLECS 4.0 or older in C-Script blocks using the C99 language standard.

**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, 3d-arrays or strings.



## 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 196). 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 210).

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.

A “Find” dialog for finding and optionally replacing certain text is available from the context menu or by pressing Ctrl-F. The dialog has an option to search the current code section only (“This section”) or all code sections of the C-Script (“All sections”).

## Probe Signals

### Input $i$

The  $i$ th input signal.

### Output $i$

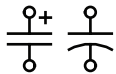
The  $i$ th output signal.

# 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 41) 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:

$$\begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} C_1 & 0 & \dots & 0 \\ 0 & C_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & C_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt} v_1 \\ \frac{d}{dt} v_2 \\ \vdots \\ \frac{d}{dt} v_n \end{bmatrix}$$

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.



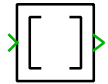
**Probe Signal**      **Output**  
The time signal.

## 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 as a vector of boolean values and outputs a row from the **Truth table** according to the input values. For an input vector of width  $n$ , the truth table must have  $2^n$  rows and the row number is calculated as  $r = 1 + \sum_i 2^{n-i} u_i$  where  $u_i = 1$  if the  $i$ th input signal is greater than 0,  $u_i = 0$  otherwise, i.e. the first element of the input vector is interpreted as the most significant bit and the  $n$ th element as the least significant bit.

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 used to calculate the output. The table must have  $2^n$  rows, and  $n$  determines the width of the input signal. The number of columns determines the width of the output signal.

### Probe Signals

#### Input

The input signals.

#### Output

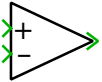
The output signals.

## 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.

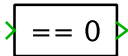
**Output**  
The output signals.

## Compare to Constant

**Purpose** Compare the input signal to a constant threshold

**Library** Control / Logical

**Description** The Compare to Constant block compares the input signal to a threshold value. If the comparison is true it outputs 1, otherwise 0. The input signal is the first argument and the threshold value the second argument for the comparison operator.



**Parameters**

**Relational operator**  
Chooses which comparison operation is applied to the input signal and the threshold value. Available operators are

- equal (==),
- unequal (~=),
- less (<),
- less or equal (<=),
- greater or equal (>=),
- greater (>).

**Threshold value**  
The value to which the input signal is compared.

**Probe Signals**

**Input**  
The input signals.

**Output**  
The output signal.

## 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** from the **Subsystem** submenu of the **Edit** menu or the block's context menu 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. The variable **Configuration** can be used in the mask initialization commands to check the current configuration. It contains an integer value starting at 1 for the first configuration.

Additional parameters for the Configurable Subsystem can be created by masking the block (see “Mask Parameters” (on page 66) for more details).

**Probe Signals**

Probe signals for the Configurable Subsystem can be created by masking the block (see “Mask Probe Signals” (on page 72) for more details).



# Constant

**Purpose** Generate constant signal

**Library** Control / Sources

**Description** The Constant block outputs a constant signal.



The constant value is displayed in the block if it is large enough, otherwise a default text is shown. To resize the block, select it, then drag one of its selection handles.

**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 Signal** **Output**  
The constant signal.

**Output data type**  
*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

## 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.

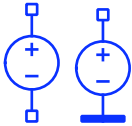
**Probe Signal** **Heat flow**  
The heat flow in watts (W).

## 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 Signal**

**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.

**Parameter**

**Allow state-space inlining**

**For expert use only!** When set to on and the input signal is a linear combination of thermal measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

**Probe Signal**

**Heat flow**

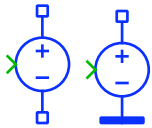
The heat flow in watts (W).

# 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.

## Parameter

### Allow state-space inlining

**For expert use only!** When set to on and the input signal is a linear combination of thermal measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

## Probe Signal

### Temperature

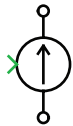
The temperature difference in kelvin (K).

## 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.

---

### Parameters

#### Discretization behavior

Specifies whether a zero-order hold or a first-order hold is applied to the input signal when the model is discretized. For details, see “Physical Model Discretization” (on page 33).

The option **Non-causal zero-order hold** applies a zero-order hold with the input signal value from the *current* simulation step instead of the *previous* one. This option can be used to compensate for a known delay of the input signal.

#### Allow state-space inlining

**For expert use only!** When set to on and the input signal is a linear combination of electrical measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

### 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.

If a variable-step solver is used, the solver step size is automatically limited to ensure that a smooth current waveform is produced.

---

**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  $\omega$ , in  $\text{s}^{-1}$ . The default is  $2 \cdot \pi \cdot 50$  which corresponds to 50 Hz.

#### Phase

The phase shift  $\varphi$ , 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

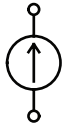
The instantaneous output power of the source, in watts (W).

## 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).

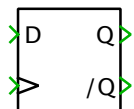


## D Flip-flop

**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:



<b>D</b>	<b>Clk</b>	<b>Q</b>	<b>/Q</b>
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.

### Parameters

#### 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  $/Q$ .

## Data Type

**Purpose** *For code generation only.* Cast the input signal to the specified data type.

**Library** Control / Math

**Description** During code generation, this block prepends a cast operator to the expression of the input signal. If the input signal already has the specified data type, this block does nothing.

`>double>`

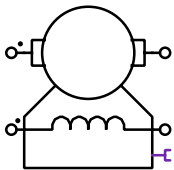
**Parameters** **Data type**  
The data type of the output signal. See “Data Types” (on page 261).

# DC Machine

**Purpose** Simple model of DC machine

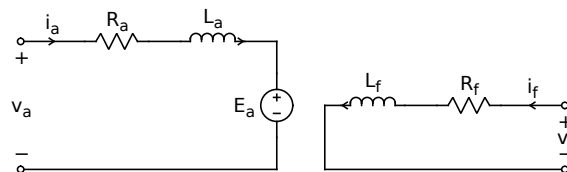
**Library** Electrical / Machines

## 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 ( $\Omega$ ).

### Armature inductance

Armature winding inductance  $L_a$  in henries (H).

### Field resistance

Field winding resistance  $R_f$  in ohms ( $\Omega$ ).

**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  $\text{Nms}^2$ .

**Friction coefficient**

Viscous friction  $F$  in Nms.

**Initial rotor speed**

Initial mechanical speed  $\omega_{m,0}$  in radians per second ( $\text{s}^{-1}$ ).

**Initial rotor position**

Initial mechanical rotor angle  $\theta_{m,0}$  in radians.

**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).

**Probe Signals****Armature current**

The current  $i_a$  in the armature winding, in amperes (A).

**Field current**

The current  $i_f$  in the field winding, in amperes (A).

**Rotational speed**

The rotational speed  $\omega_m$  of the rotor in radians per second ( $\text{s}^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  in radians.

**Electrical torque**

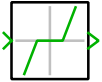
The electrical torque  $T_e$  of the machine in Nm.

## 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

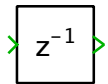
The block output signal.

## 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

A scalar, vector or matrix that defines the initial values of the internal states. The number of rows must be 1 or equal to the width of the input signal; the number of columns must be 1 or equal to the delay order. Scalar dimensions are expanded as needed.

#### Sample time

The length of the sample period in sec. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 36).

### 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 when the voltage between anode and cathode becomes positive and opens when the current through the component becomes negative. 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 ( $\Omega$ ). 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 115). 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

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

---

**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 141) 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.

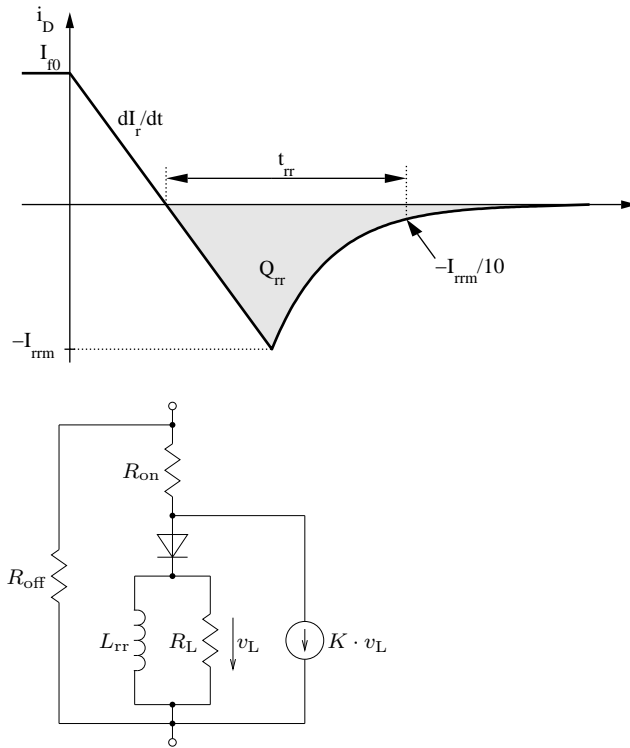
The following figure illustrates the relationship between the diode parameters and the turn-off current waveform.  $I_{f0}$  and  $dI_r/dt$  denote the continuous forward current and the rated turn-off current slope under test conditions. The turn-off time  $t_{rr}$  is defined as the period between 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 ( $\Omega$ ). The default is 0.

### Off-resistance

The resistance  $R_{off}$  of the blocking device, in ohms ( $\Omega$ ). The default is 1e6. This parameter may be set to inf unless multiple diodes are connected in series.

### Continuous forward current

The continuous forward current  $I_{f0}$  under test conditions.

### Current slope at turn-off

The turn-off current slope  $dI_f/dt$  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**

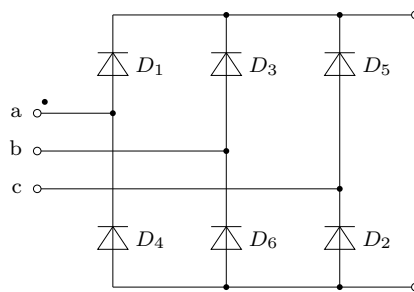
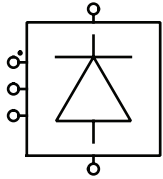
- A. Courtay, "MAST power diode and thyristor models including automatic parameter extraction", SABER User Group Meeting Brighton, UK, Sept. 1995.

## Diode Rectifier (3ph)

**Purpose** 3-phase diode rectifier

**Library** Electrical / Converters

**Description** Implements a three-phase rectifier based on the Diode model (see page 361). The electrical circuit for the rectifier is given below:



**Parameters** For a description of the parameters see the documentation of the Diode (on page 361).

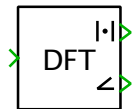
**Probe Signals** The Diode Rectifier provides six probe signals, each a vector containing the appropriate quantities of the six individual diodes: voltage, current, conductivity, conduction loss and switching loss. The vector elements are ordered according to the natural sequence of commutation.

## 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.

### Parameters

#### Sample time

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 36).

#### 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.

### Probe Signals

#### Input

The input signal.

#### Magnitude

The first output signal, i.e. the computed magnitude.

#### Phase

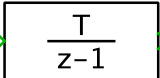
The second output signal, i.e. the computed phase angle, in radians.

## Discrete Integrator

**Purpose** Discrete integration of the input signal

**Library** Control / Discrete

**Description** The Discrete Integrator block outputs the integral of its input signal at the current sample 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.



$$\frac{T}{z-1}$$

### Integration Methods

You can choose between three integration methods using the **Integration method** parameter: Forward Euler, Backward Euler and Trapezoidal. The output and update equations for these methods are listed below:

#### Forward Euler

First simulation step:

$$y[0] = x[0]$$

$$x[1] = y[0]$$

Subsequent simulation steps:

$$y[k] = x[k] + T \cdot u[k - 1]$$

$$x[k + 1] = y[k]$$

#### Backward Euler

First simulation step:

$$y[0] = x[0]$$

$$x[1] = y[0]$$

Subsequent simulation steps:

$$y[k] = x[k] + T \cdot u[k]$$

$$x[k + 1] = y[k]$$

## Trapezoidal

First simulation step:

$$y[0] = x[0]$$

$$x[1] = y[0]$$

Subsequent simulation steps:

$$y[k] = x[k] + \frac{T}{2} \cdot (u[k-1] + u[k])$$

$$x[k+1] = y[k]$$

In the above equations, if the block has a fixed-step discrete sample time (either inherited or specified explicitly),  $T$  equals the sample period. If the block is executed within a triggered subsystem,  $T$  equals the time span between the previous and the current trigger.

The *first simulation step* refers to the first time the block executes after a simulation has been started from the initial block parameters (as opposed to a stored system state) or – if the block is executed within an enabled subsystem – after the block has been enabled. If the block is executed within a triggered subsystem, the first execution after simulation start is always treated as a *first simulation step* even if the simulation is restarted from a stored system state because the value of  $T$  cannot be determined in this case.

---

**Note** If the Backward Euler or Trapezoidal integration method is used, the input signal has direct feedthrough on the output signal.

---

## Reset Behavior

The integrator may be reset to its initial condition by an external input signal. This is controlled by the **External reset** parameter. The available options are rising/falling/either edge or level as described below:

## Rising Edge Reset

The block output and state are reset to the initial condition if the current reset input value is non-zero and the previous reset input value was zero.

## Falling Edge Reset

The block output and state are reset to the initial condition if the current reset input value is zero and the previous reset input value was non-zero.

## Either Edge Reset

The block output and state are reset to the initial condition if the current reset input value is non-zero and the previous reset input value was zero or if the current reset input value is zero and the previous reset input value was non-zero.

## Level Reset

The block output and state are reset to and held at the initial condition while the current reset input value is non-zero.

---

**Note** Both the external reset input and the initial condition input have direct feedthrough on the output signal. Therefore, feeding back the output signal to create the reset signal or an initial value will create an algebraic loop. This can be avoided by using the state port instead.

---

## Parameters

### External reset

The behaviour of the external reset input. See **Reset Behavior** above.

### Initial condition source

Specifies whether the initial condition is provided via the **Initial condition** parameter (internal) or via an input signal (external).

### Initial condition

The initial condition of the integrator. The value may be a scalar or a vector corresponding to the width of the component. This parameter is shown only if the **Initial condition source** parameter is set to internal.



**Show state port**

Specifies whether to show an additional state output port. The state port is updated at a slightly different point in the block execution order (i.e. *before* the reset and initial condition inputs are evaluated) and may therefore be used to calculate an input signal for the external reset input or the initial condition input.

**Upper saturation limit**

An upper limit for the output signal. If the value is `inf` the output signal is unlimited.

**Lower saturation limit**

A lower limit for the output signal. If the value is `-inf` the output signal is unlimited.

**Integration method**

The method used to integrate the input signal. See **Integration Methods** above.

**Sample time**

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 36).

**Probe Signal****State**

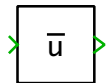
The internal state of the integrator.

## 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.

### 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 36).

#### Number of samples

The number of samples used to calculate the mean value.

### Probe Signals

#### Input

The input signal.

#### Mean

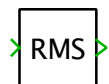
The output signal, i.e. the computed mean value.

## 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 372).

### 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 36).

#### Number of samples

The number of samples used to calculate the RMS value.

## Discrete State Space

**Purpose** Implement discrete time-invariant system as state-space model

**Library** Control / Discrete

**Description** The Discrete State Space block models a state space system of the form  $x_{i+1} = Ax_i + Bu_i$ ,  $y_i = Cx_i + Du_i$ , where  $x_i$  is the state vector at sample time step  $i$ ,  $u$  is the input vector, and  $y$  is the output vector.

$$\begin{cases} x_{i+1} = Ax_i + Bu_i \\ y_i = Cx_i + Du_i \end{cases}$$

**Parameters**

**A,B,C,D**

The coefficient matrices for the discrete state space system. The dimensions for the coefficient matrices must conform to the dimensions shown in the diagram below:

$$\begin{array}{c} n \\ \left[ \begin{array}{c} \\ \\ \\ \end{array} \right] \\ n \\ \left[ \begin{array}{c} \\ \\ \\ \end{array} \right] \\ p \\ \left[ \begin{array}{c} \\ \\ \\ \end{array} \right] \end{array} \quad \begin{array}{c} n \\ \left[ \begin{array}{c} \\ \\ \\ \end{array} \right] \\ m \\ \left[ \begin{array}{c} \\ \\ \\ \end{array} \right] \\ m \\ \left[ \begin{array}{c} \\ \\ \\ \end{array} \right] \end{array}$$

where  $n$  is the number of states,  $m$  is the width of the input signal and  $p$  is the width of the output signal.

**Sample time**

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 36).

**Initial condition**

A vector of initial values for the state vector,  $x$ .

**Probe Signals**

**Input**

The input vector,  $u$ .

**Output**

The output vector,  $y$ .

## 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  $U_{\nu}$  is the RMS value of the  $\nu$ th harmonic of the input signal and  $U_{\text{rms}}$  is its overall RMS value. The fundamental frequency  $f_1$  of the running window is

$$f_1 = \frac{1}{\text{sample time} \times \text{number of samples}}$$

### Parameters

#### Sample time

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 36).

#### Number of samples

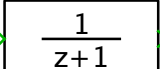
The number of samples used to calculate the THD.

## 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:



$$\frac{1}{z+1}$$

$$\frac{Y(z)}{U(z)} = \frac{n_n z^n + \cdots + n_1 z + n_0}{d_n z^n + \cdots + d_1 z + d_0}$$

The transfer function is displayed in the block if it is large enough, otherwise a default text is shown. To resize the block, select it, then drag one of its selection handles.

**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.

---

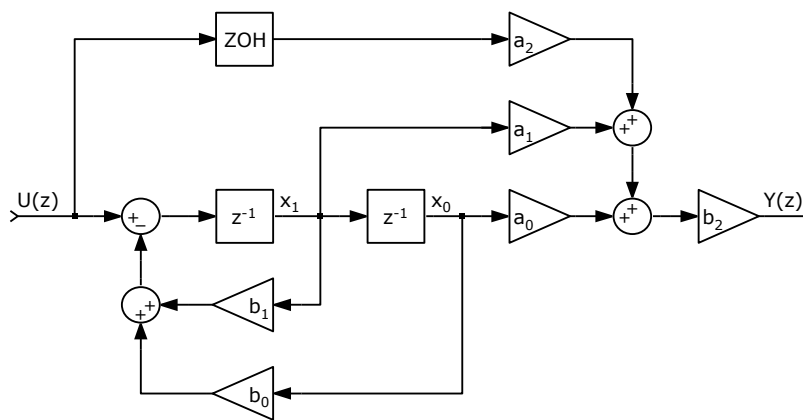
### Sample time

The time interval between samples. See also the **Discrete-Periodic** sample time type in section “Sample Times” (on page 36).

### 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

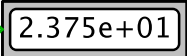
The output signal.

## Display

**Purpose** Display signal values in the schematic

**Library** System

**Description** The display block shows the numeric value of the input signal. The value is updated while the simulation is running.

A screenshot of a display block showing the value 2.375e+01 in scientific notation. The value is enclosed in a rectangular box with a thin border. A small green arrow points to the left of the box.

### Parameters

#### Notation

This parameter determines if the value of the signal is displayed in decimal or scientific notation, e.g. “0.123” or “1.23e-1”, respectively.

#### Precision

The precision determines the number of digits after the dot. The allowed range of this parameter is 0 to 16.



## DLL

**Purpose** Interface with externally generated dynamic-link library

**Library** Control / Functions & Tables

### Description



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.

The DLL must supply two functions, `plecsSetSizes` and `plecsOutput`. Additionally it may implement the functions `plecsStart` and `plecsTerminate`.

The complete DLL interface is described in the file `include/plecs/DllHeader.h` in the PLECS installation directory. This file should be included when implementing the 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, filename.dll, filename_32.dll</i>
Windows 64-bit	<i>filename, filename.dll, filename_64.dll</i>
macOS 32-bit	<i>filename, filename.dylib, filename_32.dylib</i>
macOS 64-bit	<i>filename, filename.dylib, filename_64.dylib</i>
Linux 64-bit	<i>filename, filename.so, filename_64.so</i>

### Sample time

Determines the time steps, at which the output function of the DLL is called. Valid inputs are a **Discrete-Periodic** or an **Inherited** sample time. See also section “Sample Times” (on page 36).

### 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. If an inherited sample time is specified, the output delay must be zero.

### 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 plocsSetSizes method.

## Probe Signals

### Input

The input signal.

### Output

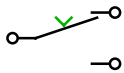
The output signal.

## 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 Signal**

**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** The output of the edge detection block changes to 1 when an edge is detected on the input signal. It returns to 0 in the following simulation step.



The block allows you to detect the following edges:

**rising**

The output is set to 1 when the input changes from 0 to a non-zero value.

**falling**

The output is set to 1 when the input changes from a non-zero value to 0.

**either**

The output is set to 1 when the input changes from 0 to a non-zero value or vice versa.

**Parameter** **Edge direction**  
The direction of the edges to detect, as described above.

**Probe Signals** **Input**  
The input signal.

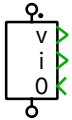
**Output**  
The output signal.

## Electrical Algebraic Component

**Purpose** Define an algebraic constraint in terms of voltage and current

**Library** Electrical / Passive Components

### Description



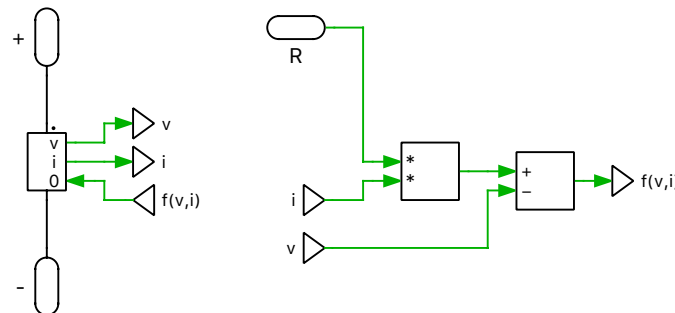
The Electrical Algebraic Component enforces an arbitrary algebraic constraint involving voltage and current. The small dot marks the positive terminal.

The output signal “v” measures the voltage across the component from the positive to the negative terminal. The output signal “i” measures the current flowing into the positive terminal. The two output signals must affect the input signal “0” by means of a direct feedthrough path. The component ensures that the input signal is zero at all times.

The direct feedthrough path defines a function  $f(v, i)$ , which in turn implicitly determines the current-voltage characteristic of the component through the constraint  $f(v, i) = 0$ . For instance, the choice  $f(v, i) := v - R \cdot i$  causes the Electrical Algebraic Component to act as an ideal resistor with resistance  $R$ .

The Electrical Algebraic Component can be vectorized. The three signals “v”, “i” and “0” must have the same width.

By way of illustration, the following schematic shows a possible implementation of a variable resistor:



**Note** The Electrical Algebraic Component creates an algebraic loop. See section “Block Sorting” (on page 29) for more information on algebraic loops.

## **Probe Signals**

### **Component voltage**

The voltage measured across the component from the positive to the negative terminal.

### **Component current**

The current flowing into the positive terminal of the component.

### **Component power**

The power consumed by the component.



---

## 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.



The parameter dialog of the Electrical Label block provides a list of links to the corresponding blocks, i.e. all other Electrical Label blocks with a matching tag name and scope. Note that the list is not updated until you click the **Apply** button after changing the tag name or scope.

**Parameters**

**Tag name:**

The tag name is used to find other matching Electrical Label blocks to connect to.

**Scope:**

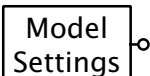
The scope specifies the search depth for the matching Electrical Label blocks. Using the value **Global** the complete PLECS circuit is searched. When set to **Schematic** only the schematic containing the Electrical Label block is searched. The setting **Masked Subsystem** causes a lookup within the hierarchy of the masked subsystem in which the block is contained. If the block is not contained in a masked subsystem a global lookup is done.

## Electrical Model Settings

**Purpose** Configure settings for an individual electrical model.

**Library** Electrical / Model Settings

### Description



The Electrical Model Settings block lets you configure parameter settings that influence the code generation for a particular electrical system, see also “Code Generation for Physical Systems” (on page 257).

The block affects the electrical circuit that it is attached to by its electrical terminal. At most one Model Settings block may be attached to an individual state-space system. An electrical model can be split into multiple state-space systems if the underlying model equations are fully decoupled. See also the options **Enable state-space splitting** and **Display state-space splitting** in the “Simulation Parameters” (on page 103).

### Parameters

#### Switching algorithm

This parameter allows you to choose between two algorithms to determine the switch conduction states in the generated code. See “Switching Algorithm” (on page 259) for details.

#### Matrix coding style

This setting allows you to specify the format used for storing the state-space matrices for a physical model. When set to `sparse`, only the non-zero matrix entries and their row and column indices are stored. When set to `full`, matrices are stored as full  $m \times n$  arrays. When set to `full (inlined)`, the matrices are additionally embedded in helper functions, which may enable the compiler to further optimize the matrix-vector-multiplications at the cost of increased code size.

#### Topologies

This parameter lets you specify a matrix containing the combinations of switch conduction states, for which code should be generated, see also “Reducing the Code Size” (on page 258). The columns represent the switch elements of the physical model and each row represents one combination of switch states (zero for an open switch and non-zero for a closed switch).

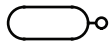
The default is an empty matrix `[]`, which means that *all* possible combinations are included.

## Electrical Port

**Purpose** Add electrical connector to subsystem

**Library** System

### Description



Electrical ports are used to establish electrical connections between a schematic and the subschematic of a subsystem (see page 603). 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, the terminal label will also disappear.

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.

### Electrical Ports in a Top-Level Schematic

In PLECS Blockset, if an Electrical Port is placed in a top-level schematic, the PLECS Circuit block in the Simulink model will show a corresponding electrical terminal, which may be connected with other electrical terminals of the same or a different PLECS Circuit block. The Electrical Port is also assigned a unique *physical port number*. Together with the parameter **Location on circuit block** the port number determines the position of the electrical terminal of the PLECS Circuit block.

For compatibility reasons you can also place an Electrical Port in a top-level schematic in PLECS Standalone. However, since there is no parent system to connect to, such a port will act like an isolated node.

### Parameter

#### Width

The width of the connected wire. The default auto means that the width is inherited from connected components.

#### Port number

If an Electrical Port is placed in a top-level schematic in PLECS Blockset, this parameter determines the position, at which the corresponding terminal appears on the PLECS Circuit block.

#### Location on circuit block

If an Electrical Port is placed in a top-level schematic in PLECS Blockset, this parameter specifies the side of the PLECS Circuit block on which

the corresponding terminal appears. By convention, `left` refers to the side on which also input terminals are shown, and `right` refers to the side on which also output terminals are shown.

## Enable

**Purpose** Control execution of an atomic subsystem

**Library** System

### Description



The Enable block is used in an atomic subsystem (see “Virtual and Atomic Subsystems” on page 603) to create an enabled subsystem. When you copy an Enable block into the schematic of a subsystem, a corresponding enable terminal will be created on the Subsystem block. In order to move this terminal around the edges of the Subsystem block, hold down the **Shift** key while dragging the terminal with the left mouse button or use the middle mouse button.

An enabled subsystem is executed while the enable signal is non-zero. The enable signal may be a vector signal. In this case the enabled subsystem is executed while *any* enabled signal is non-zero.

If the sample time of the Subsystem block is not inherited, the enable signal will be evaluated only at the instants specified by the sample time parameter.

---

**Note** An enabled subsystem may not contain any physical components.

---

### Parameters

#### Width

The width of the enable signal. The default auto means that the width is inherited from connected blocks.

#### Show output port

When this parameter is set to on, the Enable block shows an output terminal for accessing the enable signal within the subsystem.

#### Output data type

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

### Probe Signal

#### Output

The output signal of the Enable block.

## 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 583) or by dragging the component into the dialog box of a Probe block.

The magnetic flux  $\Phi$  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 Signal

#### Flux rate

The rate-of-change  $\dot{\Phi}$  of magnetic flux flowing through the component, in Wb/s or V.

## Force (Constant)

**Purpose** Generate constant force

**Library** Mechanical / Translational / Sources

**Description** The Constant Force generates a constant force between its two flanges. The direction of a positive force is indicated by the arrow.




---

**Note** A force source may not be left unconnected or connected in series with a spring or any other force source.

---

**Parameters**

**Second flange**

Controls whether the second flange is accessible or connected to the translational reference frame.

**Force**

The magnitude of the torque, in Newton (N). The default value is 1.

**Probe Signals**

**Force**

The generated force, in Newton (N).

**Speed**

The speed of the flange that the arrow points to with respect to the other flange, in  $\frac{m}{s}$ .

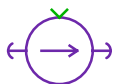


## Force (Controlled)

**Purpose** Generate variable force

**Library** Mechanical / Translational / Sources

**Description**



The Controlled Force generates a variable force between its two flanges. The direction of a positive force is indicated by the arrow. The momentary force is determined by the signal fed into the input of the component.

---

**Note** A force source may not be left unconnected or connected in series with a spring or any other force source.

---

**Parameters**

**Second flange**

Controls whether the second flange is accessible or connected to the translational reference frame.

**Allow state-space inlining**

**For expert use only!** When set to on and the input signal is a linear combination of mechanical measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

**Probe Signals**

**Force**

The generated force, in Newton (N).

**Speed**

The speed of the flange that the arrow points to with respect to the other flange, in  $\frac{m}{s}$ .

## Force Sensor

**Purpose** Output measured force as signal

**Library** Mechanical / Translational / Sensors

**Description**



The Force Sensor measures the force between its two flanges and provides it as a signal at the output of the component. A force flow from the unmarked flange towards the flange marked with a dot is considered positive.

---

**Note** A force sensor is ideally rigid. Hence, if multiple force sensors are connected in parallel the force measured by an individual sensor is undefined. This produces a run-time error.

---

**Parameter**

**Second flange**

Controls whether the second flange is accessible or connected to the translational reference frame.

**Probe Signal**

**Force**

The measured force, in Newton (N).

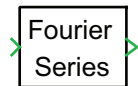
## Fourier Series

**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  $x$ .

**Parameters**

**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**

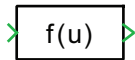
The output signal.

## 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 `pi`
- 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

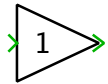
The output signal.

# 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 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.

### Output data type

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

## Probe Signals

### Input

The input signal.

### Output

The output signal.

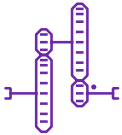
## Gear

**Purpose**

Ideal gear

**Library**

Mechanical / Rotational / Components

**Description**

The Gear models an ideal gearbox with two shafts rotating in the same direction. The shaft moving with  $\omega_2$  is marked with a dot. The relation between the torques and angular speeds of the two shafts is described with the following equations:

$$\omega_2 = g \cdot \omega_1$$

$$\tau_1 = g \cdot \tau_2$$

where  $g$  is the gearbox ratio.

**Parameter****Gear ratio**

The gearbox ratio  $g$ . A negative gearbox ratio will cause the shafts to rotate in opposite directions.

# 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 when the voltage between anode and cathode is positive and a positive gate signal is applied. It opens when the current becomes negative or 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 ( $\Omega$ ). 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 115). 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

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

## 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 115) and “Losses of Semiconductor Switch with Diode” (on page 142) for more information.

**Initial temperature**

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

**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 583) or by dragging the component into the dialog box of a Probe block.

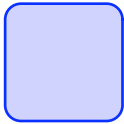
**Probe Signal** **Measured heat flow**  
The measured heat flow in watts (W).

## Heat Sink

**Purpose** Isotherm environment for placing components

**Library** Thermal

### Description



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.

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.

For additional information see chapter “Thermal Modeling” (on page 115).

### Parameters

#### **Number of terminals**

This parameter allows you to change the number of external thermal connectors of a heat sink. The default is 0.

#### **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.

#### **Initial temperature**

The initial temperature difference between the heat sink and the thermal reference at simulation start, in kelvin (K). The default is 0. If left blank or if the value is nan, PLECS will initialize the value based on a thermal “DC” analysis, see “Temperature Initialization” (on page 121).

### Probe Signal

#### **Temperature**

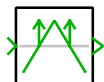
The temperature difference between the heat sink and the thermal reference, in kelvin (K).

## 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

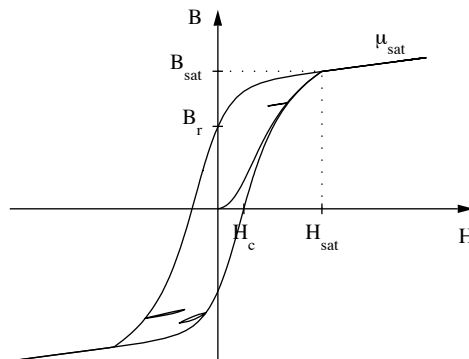
**Library** Magnetic

### 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  $(H_{\text{sat}}, B_{\text{sat}})$ , the coercitive field strength  $H_c$ , the remanence flux density  $B_r$  and the saturated permeability  $\mu_{\text{sat}}$ .



### Parameters

#### Cross-sectional area

Cross-sectional area  $A$  of the flux path, in  $\text{m}^2$ .

#### 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  $H_{\text{sat}}$  at the saturation point, in A/m.

**Saturation flux density**

Flux density  $B_{\text{sat}}$  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.

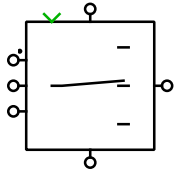
## Ideal 3-Level Converter (3ph)

**Purpose** Switch-based 3-phase 3-level converter

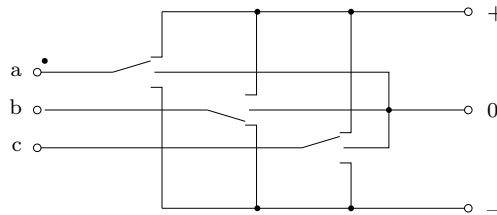
**Library** Electrical / Converters

**Description**

Implements a three-phase three-level converter with ideal switches. The converter is modeled using the Triple Switch component (see page 691). 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:



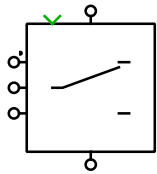


## Ideal Converter (3ph)

**Purpose** Switch-based 3-phase converter

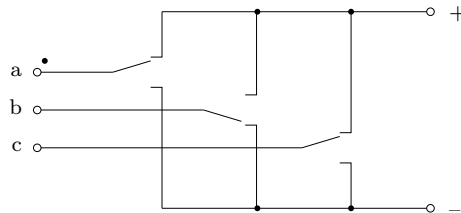
**Library** Electrical / Converters

### Description



Implements a three-phase two-level converter with ideal bi-positional switches. The converter is modeled using the Double Switch component (see page 383). 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:

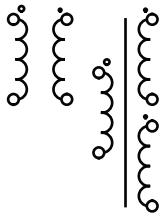


## Ideal Transformer

**Purpose** Ideally coupled windings with or without magnetizing 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:

$$\frac{v_1}{n_1} = \frac{v_2}{n_2} = \frac{v_3}{n_3} = \dots$$

If the transformer does not have a finite magnetizing inductance (i.e. the inductance value is set to `inf`), the currents  $i_w$  of all windings multiplied with the corresponding number of turns add up to zero:

$$0 = i_1 \cdot n_1 + i_2 \cdot n_2 + i_3 \cdot n_3 + \dots$$

If the transformer *does* have a finite magnetizing inductance, the currents  $i_w$  of all windings multiplied with the corresponding number of turns add up to the magnetizing current multiplied with the number of turns in the first winding:

$$i_m \cdot n_1 = i_1 \cdot n_1 + i_2 \cdot n_2 + i_3 \cdot n_3 + \dots$$

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. Currents entering a terminal marked with the circle or a dot are considered positive.

Use the **Polarity** parameter to change the orientation of a specific winding. This is equivalent to making the corresponding number of turns  $n_w$  negative.

### Parameters

#### 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.

#### 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.

**Polarity**

A string consisting of one + or - per winding specifying the winding polarity. A single + or - is applied to all windings.

**Magnetizing inductance**

A non-zero scalar specifying the magnetizing inductance referred to the first winding, in henries (H).

**Initial magnetizing current**

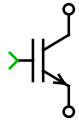
A scalar specifying the initial current through the magnetizing inductance at simulation start, in Amperes (A). Must be zero if the magnetizing inductance is infinite inf.

## 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 ( $\Omega$ ). 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 115). 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

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

### 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**

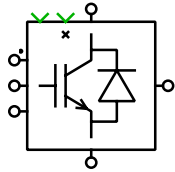
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

## 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  $\geq 1$ , and the second IGBT if the signal is  $\geq 0$ . The third IGBT is turned on for signals  $\leq 0$  and the lowest one for signals  $\leq -1$ . 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 434). No parameters can be entered.
- The **IGBT 3-Level Converter with Parasitics** is based on individual IGBT (see page 414) and Diode (see page 361) components. In this model you may specify forward voltages and on-resistances separately for the IGBTs and the diodes.

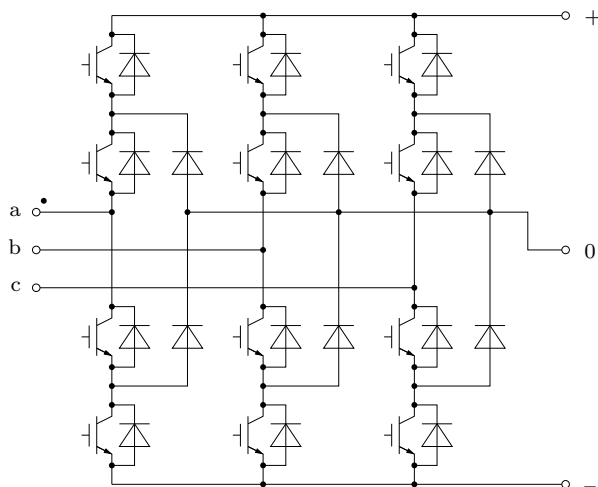
---

**Note** Due to the switching conditions of the IGBT with Diode (see page 434), this device *cannot* be turned off actively while the current is exactly zero. This may result in unexpected voltage waveforms if the converter is not loaded.

To resolve this problem, either use the **IGBT 3-Level Converter with Parasitics**, or allow a small non-zero load current to flow by connecting a large load resistance to the converter.

---

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 434), the IGBT (on page 414) and the Diode (on page 361).

### 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).

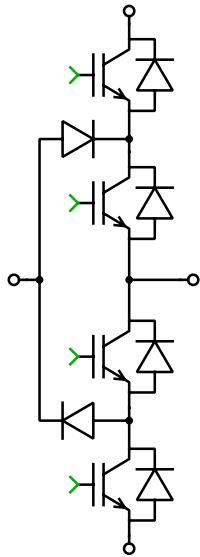
## IGBT 3-Level Half Bridge (NPC)

**Purpose** 3-level neutral-point clamped IGBT module

**Library** Electrical / Power Modules

### Description

This power module implements a single leg of a 3-level neutral-point clamped voltage source inverter. It offers two configurations:



**Switched** All power semiconductors inside the module are modeled with ideal switches. The individual IGBTs are controlled with logical gate signals. An IGBT is on if the corresponding gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The module as a whole is modeled with controlled voltage and current sources. The DC side of the inverter bridge has current source behavior and must be connected to positively biased capacitors or voltage sources. The phase terminal is typically connected to an inductor. The control inputs are the relative on-times of the IGBTs with values between 0 and 1.

In the average configuration the half bridge can be operated in two ways:

- The control signals are instantaneous logical gate signals having the values 0 and 1.
- The control signals are the duty cycles of the individual IGBTs. They are either computed directly from the modulation index or by periodically averaging the digital gate signals over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation correctly accounts for blanking times, i.e. when during commutation less than two IGBTs are turned on. It also supports discontinuous conduction mode, e.g. when charging the DC link capacitors via the reverse diodes.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies.



---

**Note** The sub-cycle average implementation cannot model a shoot-through or clamping of the DC side. Therefore, the sums of the control signals for the first and third IGBT and second and fourth IGBT must not exceed 1 at any time. Also, the applied DC voltages must never become negative.

---

## Parameters

### Configuration

Switched or averaged circuit model.

### Assertions

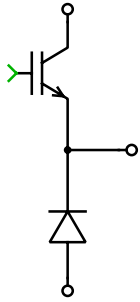
When set to on, the block will flag an error if the sums of the control signals for IGBT 1 and 3 or IGBT 2 and 4 exceed 1.

## IGBT Chopper (High-Side Switch)

**Purpose** Buck converter IGBT module

**Library** Electrical / Power Modules

### Description



This power module implements a chopper used in buck converters. It offers two configurations:

**Switched** The power semiconductors inside the module are modeled with ideal switches. The IGBT is controlled with a logical gate signal; it is on if the gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The module as a whole is modeled with controlled voltage and current sources. The electrical input of the chopper has current source behavior and must be connected to a positively biased capacitor or voltage source. The output terminal is typically connected to an inductor. The control input is the relative on-time of the IGBT with values between 0 and 1.

In the average configuration the chopper can be operated in two ways:

- The control signal is the instantaneous logical gate signal having the values 0 and 1.
- The control signal is the duty cycle of the IGBT. It is either computed directly from the modulation index or by periodically averaging the digital gate signal over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation supports continuous and discontinuous conduction mode.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies.

---

**Note** The sub-cycle average implementation cannot model clamping of the input side. Therefore, the applied DC voltage must never become negative.

---

### Parameter

#### Configuration

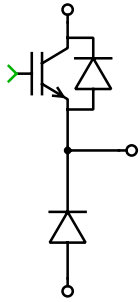
Switched or averaged circuit model.

## IGBT Chopper (High-Side Switch with Reverse Diode)

**Purpose** Buck converter IGBT module with reverse diode

**Library** Electrical / Power Modules

**Description** This power module implements a chopper used in buck converters. It offers two configurations:



**Switched** The power semiconductors inside the module are modeled with ideal switches. The IGBT is controlled with a logical gate signal; it is on if the gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The module as a whole is modeled with controlled voltage and current sources. The electrical input of the chopper has current source behavior and must be connected to a positively biased capacitor or voltage source. The output terminal is typically connected to an inductor. The control input is the relative on-time of the IGBT with values between 0 and 1.

In the average configuration the chopper can be operated in two ways:

- The control signal is the instantaneous logical gate signal having the values 0 and 1.
- The control signal is the duty cycle of the IGBT. It is either computed directly from the modulation index or by periodically averaging the digital gate signal over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation supports continuous and discontinuous conduction mode.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies. If the reversed diode is not needed, the alternative chopper module (see page 420) is preferred to minimize model complexity.

---

**Note** The sub-cycle average implementation cannot model clamping of the input side. Therefore, the applied DC voltage must never become negative.

---

**Parameter****Configuration**

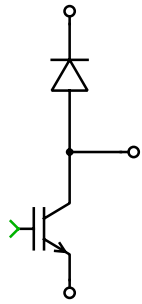
Switched or averaged circuit model.

## IGBT Chopper (Low-Side Switch)

**Purpose** Boost converter IGBT module

**Library** Electrical / Power Modules

### Description



This power module implements a chopper used in boost converters. It offers two configurations:

**Switched** The power semiconductors inside the module are modeled with ideal switches. The IGBT is controlled with a logical gate signal; it is on if the gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The module as a whole is modeled with controlled voltage and current sources. The electrical input of the chopper has current source behavior and must be connected to a positively biased capacitor or voltage source. The output terminal is typically connected to an inductor. The control input is the relative on-time of the IGBT with values between 0 and 1.

In the average configuration the chopper can be operated in two ways:

- The control signal is the instantaneous logical gate signal having the values 0 and 1.
- The control signal is the duty cycle of the IGBT. It is either computed directly from the modulation index or by periodically averaging the digital gate signal over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation supports continuous and discontinuous conduction mode.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies.

---

**Note** The sub-cycle average implementation cannot model clamping of the input side. Therefore, the applied DC voltage must never become negative.

---

### Parameter

#### Configuration

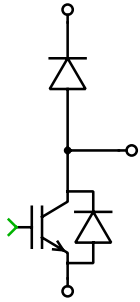
Switched or averaged circuit model.

## IGBT Chopper (Low-Side Switch with Reverse Diode)

**Purpose** Boost converter IGBT module with reverse diode

**Library** Electrical / Power Modules

**Description** This power module implements a chopper used in boost converters. It offers two configurations:



**Switched** The power semiconductors inside the module are modeled with ideal switches. The IGBT is controlled with a logical gate signal; it is on if the gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The module as a whole is modeled with controlled voltage and current sources. The electrical input of the chopper has current source behavior and must be connected to a positively biased capacitor or voltage source. The output terminal is typically connected to an inductor. The control input is the relative on-time of the IGBT with values between 0 and 1.

In the average configuration the chopper can be operated in two ways:

- The control signal is the instantaneous logical gate signal having the values 0 and 1.
- The control signal is the duty cycle of the IGBT. It is either computed directly from the modulation index or by periodically averaging the digital gate signal over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation supports continuous and discontinuous conduction mode.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies. If the reversed diode is not needed, the alternative chopper module (see page 423) is preferred to minimize model complexity.

---

**Note** The sub-cycle average implementation cannot model clamping of the input side. Therefore, the applied DC voltage must never become negative.

---

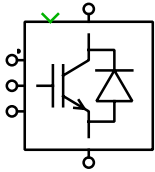
<b>Parameter</b>	<b>Configuration</b>
	Switched or averaged circuit model.

## 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 434). 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 414) and Diode (see page 361) components. In this model you may specify all electrical and thermal parameters separately for the IGBTs and the diodes.

---

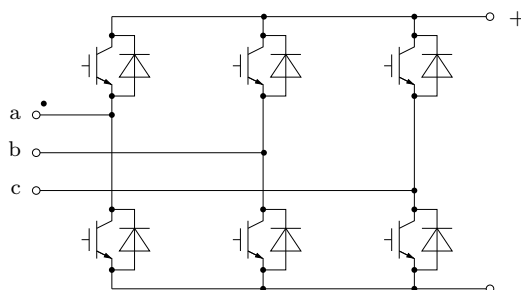
**Note** Due to the switching conditions of the IGBT with Diode (see page 434), this device *cannot* be turned off actively while the current is exactly zero. This may result in unexpected voltage waveforms if the converter is not loaded.

To resolve this problem, either use the **IGBT Converter with Parasitics**, or allow a small non-zero load current to flow by connecting a large load resistance to the converter.

---



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 434), the IGBT (on page 414) and the Diode (on page 361).

### Probe Signals

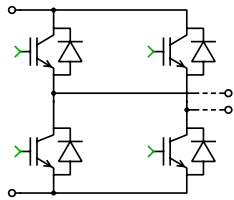
The two-level IGBT converters provide six or twelve probe signals, each a vector containing the appropriate quantities of the individual devices: voltage, current, conductivity, conduction loss and switching loss. The vector elements are ordered top-to-bottom, left-to-right: a+, a-, b+, b-, c+, c-.

## IGBT Full Bridges (Series Connected)

**Purpose** Series-connected IGBT full-bridge inverters

**Library** Electrical / Power Modules

### Description



This component implements multiple series-connected inverter cells for use in modular multilevel converters. All control signals as well as the DC link terminals are vectorized with their width matching the number of cells. The component offers two configurations:

**Switched** All power semiconductors are modeled with ideal switches. The individual IGBTs are controlled with logical gate signals. An IGBT is on if the corresponding gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The component as a whole is modeled with controlled voltage and current sources. The DC sides of the inverter cells have current source behavior and must be connected to positively biased capacitors or voltage sources. The DC sides must remain galvanically isolated from each other. The output terminals are typically connected to an inductor. The control inputs are the relative on-times of the IGBTs with values between 0 and 1.

In the average configuration the inverters can be operated in two ways:

- The control signals are instantaneous logical gate signals having the values 0 and 1.
- The control signals are the duty cycles of the individual IGBTs. They are either computed directly from the modulation index or by periodically averaging the digital gate signals over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation correctly accounts for blanking times, i.e. when during commutation both IGBTs in one inverter leg are turned off. It also supports discontinuous conduction mode, e.g. when charging the DC link capacitors via the reverse diodes.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies. It can also increase the speed of off-line simulations, because the number of internal switches is greatly reduced.

---

**Note** The sub-cycle average implementation cannot model a shoot-through, i.e. the situation where both IGBTs in a leg are turned on at the same time. Therefore, the sum of the control signals for the upper and lower IGBT in a leg must not exceed 1 at any time. Since the DC sides are not clamped, the DC voltages must never become negative.

---

## Parameters

### Configuration

Switched or averaged circuit model.

### Number of cells

Number of series-connected full-bridge inverters.

### Assertions

When set to on, the block will flag an error if the sum of the control signals for the upper and lower IGBT in a leg exceeds 1.

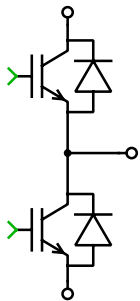
## IGBT Half Bridge

**Purpose** 2-level half-bridge IGBT module

**Library** Electrical / Power Modules

### Description

This power module implements a single leg of 2-level voltage source inverter. It offers two configurations:



**Switched** The power semiconductors are modeled with ideal switches. The individual IGBTs are controlled with logical gate signals. An IGBT is on if the corresponding gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The module as a whole is modeled with controlled voltage and current sources. The DC side of the inverter has current source behavior and must be connected to a positively biased capacitor or voltage source. The output terminal is typically connected to an inductor. The control inputs are the relative on-times of the IGBTs with values between 0 and 1.

In the average configuration the half bridge can be operated in two ways:

- The control signals are instantaneous logical gate signals having the values 0 and 1.
- The control signals are the duty cycles of the individual IGBTs. They are either computed directly from the modulation index or by periodically averaging the digital gate signals over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation correctly accounts for blanking times, i.e. when during commutation both IGBTs are turned off. It also supports discontinuous conduction mode, e.g. when charging the DC link capacitor via the reverse diodes.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies.

---

**Note** The sub-cycle average implementation cannot model a shoot-through, i.e. the situation where both IGBTs in a leg are turned on at the same time. Therefore, the sum of the control signals for the upper and lower IGBT must not exceed 1 at any time. Since the DC side is not clamped, the DC voltage must never become negative.

---

## Parameters

### Configuration

Switched or averaged circuit model.

### Assertions

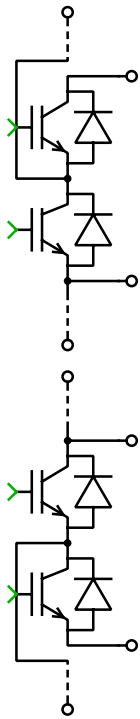
When set to on, the block will flag an error if the sum of the control signals for the upper and lower IGBT exceeds 1.

## IGBT Half Bridges (Low-/High-Side Connected)

**Purpose** Series-connected IGBT half-bridge inverters

**Library** Electrical / Power Modules

### Description



These components implement multiple series-connected inverter cells for use in modular multilevel converters. All control signals as well as the DC link terminals are vectorized with their width matching the number of cells. The components offer two configurations:

**Switched** All power semiconductors are modeled with ideal switches. The individual IGBTs are controlled with logical gate signals. An IGBT is on if the corresponding gate signal is not zero. For compatibility with the averaged configuration it is recommended to use the value 1 for non-zero gate signals.

**Sub-cycle average** The component as a whole is modeled with controlled voltage and current sources. The DC sides of the inverter cells have current source behavior and must be connected to positively biased capacitors or voltage sources. The DC sides must remain galvanically isolated from each other. The output terminals are typically connected to an inductor. The control inputs are the relative on-times of the IGBTs with values between 0 and 1.

In the average configuration the inverters can be operated in two ways:

- The control signals are instantaneous logical gate signals having the values 0 and 1.
- The control signals are the duty cycles of the individual IGBTs. They are either computed directly from the modulation index or by periodically averaging the digital gate signals over a fixed period of time, e.g. using the Periodic Average block (see page 516). The averaging period does not need to be synchronized with the PWM and can be as large as the inverse of the switching frequency.

In both use cases, the average implementation correctly accounts for blanking times, i.e. when during commutation both IGBTs in one inverter cell are turned off. It also supports discontinuous conduction mode, e.g. when charging the DC link capacitors via the reverse diodes.

Since the duty cycle is simulated accurately even with relatively large time steps, the average configuration is particularly well suited for real-time simulations with high switching frequencies. It can also increase the speed of off-line simulations, because the number of internal switches is greatly reduced.

---

**Note** The sub-cycle average implementation cannot model a shoot-through, i.e. the situation where both IGBTs in a leg are turned on at the same time. Therefore, the sum of the control signals for the upper and lower IGBT must not exceed 1 at any time. Since the DC sides are not clamped, the DC voltages must never become negative.

---

## Parameters

### Configuration

Switched or averaged circuit model.

### Number of cells

Number of series-connected half-bridge inverters.

### Assertions

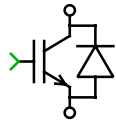
When set to on, the block will flag an error if the sum of the control signals for the upper and lower IGBT exceeds 1.

## 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.

This device is modeled as a single ideal switch that closes when the gate signal is not zero or the voltage becomes negative and opens when the gate signal is zero and the current becomes positive.

---

**Note** Due to the switching conditions described above, this device *cannot* be turned off actively while the current is exactly zero. This may result in unexpected voltage waveforms if the device is used e.g. in an unloaded converter.

To resolve this problem, either use an individual IGBT (see page 414) with an individual anti-parallel Diode (see page 361), or allow a small non-zero load current to flow by connecting a large load resistance to the converter.

---

### Parameters

#### 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.

#### Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapters “Thermal Modeling” (on page 115) and “Losses of Semiconductor Switch with Diode” (on page 142).

#### Initial temperature

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).



**Probe Signals****Device voltage**

The voltage measured between collector/cathode and emitter/anode. The device voltage can never be negative.

**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.

**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**

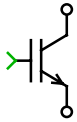
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

## IGBT with Limited di/dt

**Purpose** Dynamic IGBT model with finite current slopes during turn-on and turn-off

**Library** Electrical / Power Semiconductors

### Description



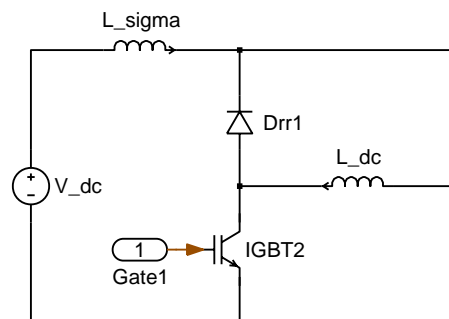
In contrast to the ideal IGBT model (see page 414) that switches instantaneously, this model includes collector current transients during switching. Thanks to the continuous current decay during turn-off, stray inductances may be connected in series with the device. In converter applications, the di/dt 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.

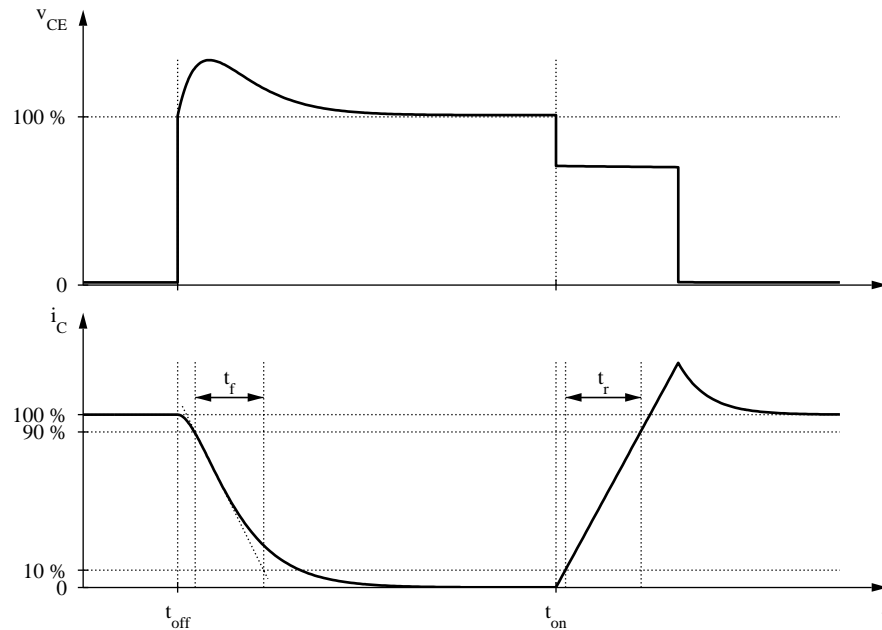
### Note

- 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 363).



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

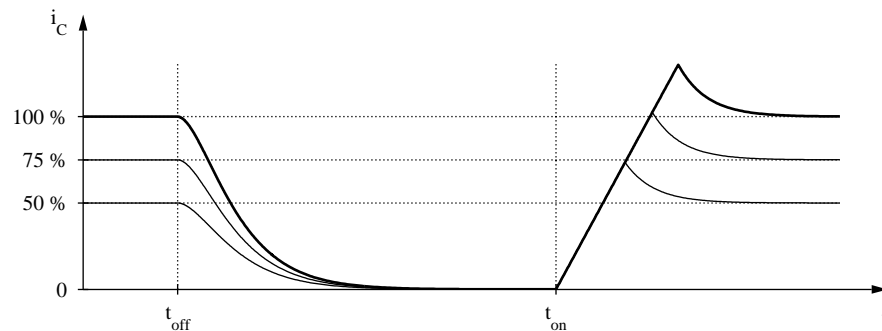
$$i_C(t) = i_C(t_{\text{off}}) \left[ e^{-\frac{2.4(t-t_{\text{off}})}{t_f}} \left( 1 + \frac{2.4(t-t_{\text{off}})}{t_f} \right) \right]$$

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 ( $\Omega$ ). The default is 0.

### Off-resistance

The resistance  $R_{off}$  of the blocking device, in ohms ( $\Omega$ ). The default is 1e6. This parameter may be set to `inf` unless multiple IGBTs are connected in series.

### 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  $L_\sigma$  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** 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 IGCT is conducting. The default is 0.

### On-resistance

The resistance  $R_{on}$  of the conducting device, in ohms ( $\Omega$ ). The default is 0.

### Initial conductivity

Initial conduction state of the IGCT. The IGCT 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 115). 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

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

**Probe Signals** **IGCT voltage**  
The voltage measured between anode and cathode.

**IGCT current**  
The current through the IGCT flowing from anode to cathode.

**IGCT gate signal**

The gate input signal of the IGCT.

**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

#### 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.

#### Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapters “Thermal Modeling” (on page 115) and “Losses of Semiconductor Switch with Diode” (on page 142).

#### Initial temperature

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

### Probe Signals

#### Device voltage

The voltage measured between anode and cathode. The device voltage can never be negative.

#### 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.

#### 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**

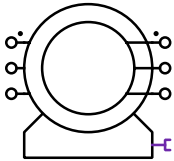
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

## Induction Machine (Slip Ring)

**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 454).

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 **Subsystem** submenu of the **Edit** menu. If you want to make changes, you must first choose **Break library link** and then **Unprotect**, both from the same menu.

### Electrical System

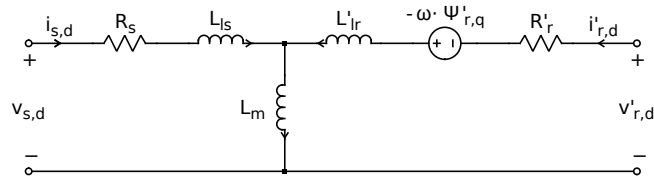
The rotor flux is computed as

$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

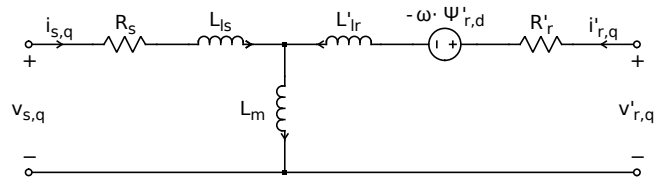
$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q})$$

The three-phase voltages  $v_{s,ab}$  and  $v_{s,bc}$  at the stator terminals are transformed into dq quantities:

$$\begin{bmatrix} v_{s,d} \\ v_{s,q} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} v_{s,ab} \\ v_{s,bc} \end{bmatrix}$$



### d-axis



### q-axis

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  $\theta$  being the electrical rotor position:

$$\begin{bmatrix} v'_{r,d} \\ v'_{r,q} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos \theta & -\cos \left( \theta - \frac{2\pi}{3} \right) \\ \sin \theta & -\sin \left( \theta - \frac{2\pi}{3} \right) \end{bmatrix} \cdot \begin{bmatrix} v'_{r,ab} \\ v'_{r,bc} \end{bmatrix}$$

$$\begin{bmatrix} i'_{r,a} \\ i'_{r,b} \\ i'_{r,c} \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ \cos \left( \theta + \frac{2\pi}{3} \right) & \sin \left( \theta + \frac{2\pi}{3} \right) \\ \cos \left( \theta - \frac{2\pi}{3} \right) & \sin \left( \theta - \frac{2\pi}{3} \right) \end{bmatrix} \cdot \begin{bmatrix} i'_{r,d} \\ i'_{r,q} \end{bmatrix}$$

## 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  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p\omega_m$$

Mechanical rotor angle  $\theta_m$ :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p\theta_m$$

### Parameters

#### Stator resistance

Stator winding resistance  $R_s$  in ohms ( $\Omega$ ).

#### Stator leakage inductance

Stator leakage inductance  $L_{ls}$  in henries (H).

#### Rotor resistance

Rotor winding resistance  $R'_r$  in ohms ( $\Omega$ ), 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  $\text{Nms}^2$ .

#### Friction coefficient

Viscous friction  $F$  in  $\text{Nms}$ .

#### Number of pole pairs

Number of pole pairs  $p$ .

#### Initial rotor speed

Initial mechanical rotor speed  $\omega_{m,0}$  in  $\text{s}^{-1}$ .

#### 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.

**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  $\omega_m$  of the rotor in radians per second ( $s^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  in radians.

**Electrical torque**

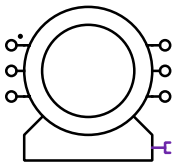
The electrical torque  $T_e$  of the machine in Nm.

## 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 **Subsystem** submenu of the **Edit** menu.

**Electrical System:**

The rotor flux is computed as

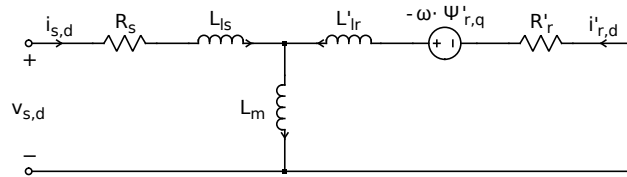
$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q})$$

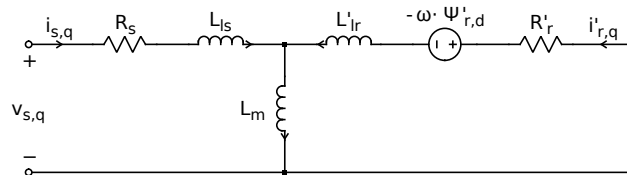
The three-phase voltages  $v_{s,a}$ ,  $v_{s,b}$  and  $v_{s,c}$  across the individual stator windings are transformed into dq0 quantities:

$$\begin{bmatrix} v_{s,d} \\ v_{s,q} \\ v_{s,0} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} v_{s,a} \\ v_{s,b} \\ v_{s,c} \end{bmatrix}$$

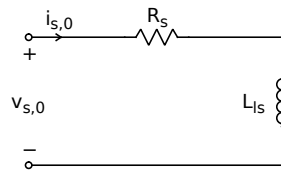
Likewise, the stator currents in the stationary reference frame are transformed back into three-phase currents:



### d-axis



### q-axis



### 0-axis

$$\begin{bmatrix} i_{s,a} \\ i_{s,b} \\ i_{s,c} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} i_{s,d} \\ i_{s,q} \\ i_{s,0} \end{bmatrix}$$

## 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  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p\omega_m$$

Mechanical rotor angle  $\theta_m$ :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p\theta_m$$

### Parameters

Most parameters for the Induction Machine with slip-ring rotor (see page 444) 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).

### Probe Signals

Most probe signals for the Induction Machine with slip-ring rotor (see page 444) 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.

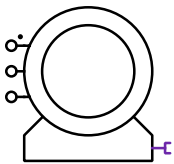


# Induction Machine (Squirrel Cage)

**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 454) 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 **Subsystem** submenu of the **Edit** menu.

## Electrical System

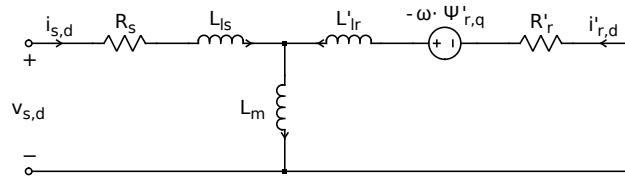
The rotor flux is computed as

$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

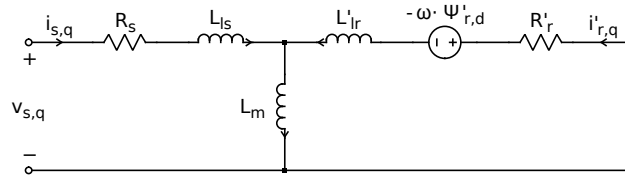
$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q})$$

The three-phase voltages  $v_{s,ab}$  and  $v_{s,bc}$  at the stator terminals are transformed into dq quantities:

$$\begin{bmatrix} v_{s,d} \\ v_{s,q} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} v_{s,ab} \\ v_{s,bc} \end{bmatrix}$$



### d-axis



### q-axis

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  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p\omega_m$$

Mechanical rotor angle  $\theta_m$ :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p\theta_m$$

**Parameters**

Same as for the Induction Machine with slip-ring rotor (see page 444).

**Probe Signals**

Most probe signals for the Induction Machine with slip-ring rotor (see page 444) are also available with this squirrel-cage machine. Only the following probe signal is different:

**Rotor currents**

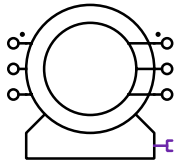
The rotor currents  $i'_{r,d}$  and  $i'_{r,q}$  in the stationary reference frame in A, referred to the stator side.

## Induction Machine with Saturation

### Purpose

Induction machine with slip-ring rotor and main-flux saturation

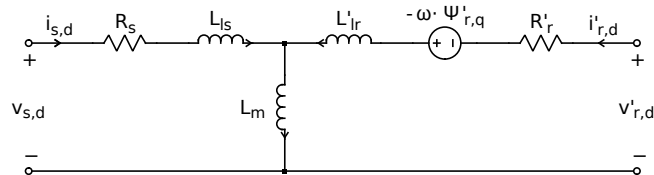
### Description



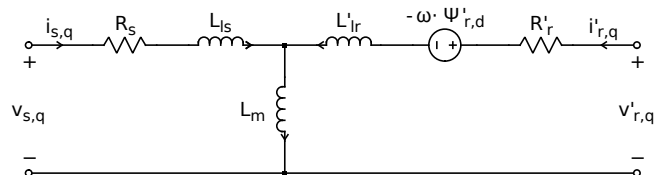
The Induction Machine with Saturation models main flux saturation by means of a continuous function.

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

$$\Psi_{r,d} = L'_{lr} i'_{r,d} + L_m (i_{s,d} + i'_{r,d})$$

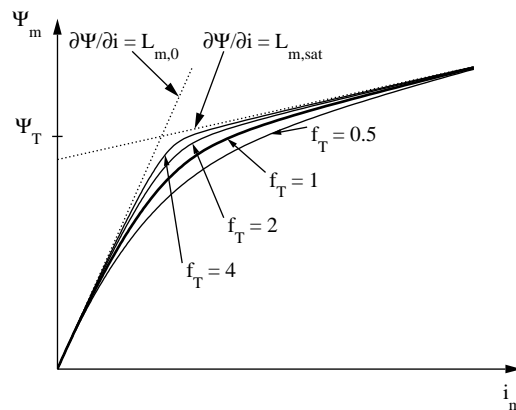
$$\Psi_{r,q} = L'_{lr} i'_{r,q} + L_m (i_{s,q} + i'_{r,q}) .$$

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 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  $\Psi_m$  as illustrated in the  $\Psi_m/i_m$  diagram. For flux linkages far below the transition flux  $\Psi_T$ , 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  $L_{m,sat}$ .  $\Psi_T$  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 **Subsystem** submenu of the **Edit** menu. If you want to make changes, you must first choose **Break library link** and then **Unprotect**, both from the same 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  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\omega = p\omega_m$$

Mechanical rotor angle  $\theta_m$ :

$$\dot{\theta}_m = \omega_m$$

$$\theta = p\theta_m$$

**Parameters****Model**

Implementation in the stationary reference frame or as a voltage behind reactance.

**Stator resistance**

Stator winding resistance  $R_s$  in ohms ( $\Omega$ ).

**Stator leakage inductance**

Stator leakage inductance  $L_{ls}$  in henries (H).

**Rotor resistance**

Rotor winding resistance  $R'_r$  in ohms ( $\Omega$ ), 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.

**Saturated magnetizing inductance**

Saturated main flux inductance  $L_{m,sat}$  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  $\Psi_T$ , 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  $\text{Nms}^2$ .

**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  $\text{s}^{-1}$ .

**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.

**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  $i'_{s,c}$  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  $\omega_m$  of the rotor in radians per second ( $s^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  in radians.

**Electrical torque**

The electrical torque  $T_e$  of the machine in Nm.

**References**

- D. C. Aliprantis, O. Wasynczuk, 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 412).

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

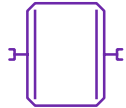
The voltage measured across the inductor, in volts (V).

## Inertia

**Purpose** Model a rotating body with inertia

**Library** Mechanical / Rotational / Components

**Description** This component models a rotating body with inertia and two rigidly connected flanges. The angular speed is determined by the equation



$$\frac{d}{dt}\omega = \frac{1}{J} \cdot (\tau_1 + \tau_2)$$

where  $\tau_1$  and  $\tau_2$  are the torques acting at the two flanges *towards* the body.

### Parameters

#### Moment of inertia

The moment of inertia  $J$ , in  $\frac{\text{kg}\cdot\text{m}^2}{\text{rad}^2}$ .

#### Initial speed

The initial angular speed  $\omega_0$ , in  $\frac{\text{rad}}{\text{s}}$ .

#### Initial angle

The initial angle  $\theta_0$ , in radians. May be specified in order to provide proper initial conditions if absolute angles are measured anywhere in the system. Otherwise, this parameter can be left blank.

### Probe Signals

#### Speed

The angular speed of the body.

#### Angle

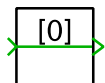
The absolute angle of the body (wrapped between  $-\pi$  and  $\pi$ ).

## Initial Condition

**Purpose** Output specified initial value in the first simulation step

**Library** Control / Sources

**Description** The Initial Condition block outputs the initial value in the very first simulation step and the input value at all subsequent steps.



If this block is placed inside an algebraic loop, it provides an initial guess of the value of its output signal to be used by the equation solver at the start of a simulation. See the section “Block Sorting” (on page 29) for more information on algebraic loops.

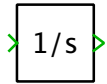
**Parameter** **Initial value**  
The initial output value in the first simulation step.

## 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. The initial value may be provided either via a parameter or via an input signal.

### Interaction with the Solver

**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.

**Parameters**

**External reset**

The behaviour of the external reset input. The values rising, falling and either 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 level is chosen, the output signal keeps the initial value while the reset input is not 0.

**Initial condition source**

Specifies whether the initial condition is provided via the **Initial condition** parameter (internal) or via an input signal (external).

**Initial condition**

The initial condition of the integrator. The value may be a scalar or a vector corresponding to the implicit width of the component. This parameter is shown only if the **Initial condition source** parameter is set to internal.

**Show state port**

Specifies whether to show an additional state output port. The state port is updated at a slightly different point in the block execution order (i.e. *before* the reset and initial condition inputs are evaluated) and may therefore

be used to calculate an input signal for the external reset input or the initial condition input.

**Upper saturation limit**

An upper limit for the output signal. If the value is `inf` the output signal is unlimited.

**Lower saturation limit**

A lower limit for the output signal. If the value is `-inf` the output signal is unlimited.

**Probe Signal****State**

The internal state of the integrator.

---

**Note** Both the external reset input and the initial condition input have direct feedthrough on the output signal. Therefore, feeding back the output signal to create the reset signal or an initial value will create an algebraic loop. This can be avoided by using the state port instead.

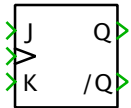
---

## 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.

**Parameters** **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  $Q$ .



$\bar{Q}$  The output signals  $\bar{Q}$ .

## 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  $\Phi$  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 475). The only difference is the symbol.

### Parameters

#### Effective Permeance

Magnetic permeance of the leakage flux path, in webers per ampere-turn (Wb/A).

#### 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.

## 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  $\Phi$  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,  $\mu_r$  is the relative permeability of the material,  $A$  is the cross-sectional area and  $l$  the length of the flux path.

### Parameters

#### Cross-sectional area

Cross-sectional area  $A$  of the flux path, in  $\text{m}^2$ .

#### Length of flux path

Length  $l$  of the flux path, in m.

#### Rel. permeability

Relative permeability  $\mu_r$  of the core material.

#### 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

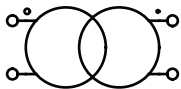
The magnetic flux density  $B$  in the core element, in teslas (T).

## Linear Transformer (2 Windings)

**Purpose** Single-phase transformer with winding resistance and optional core loss

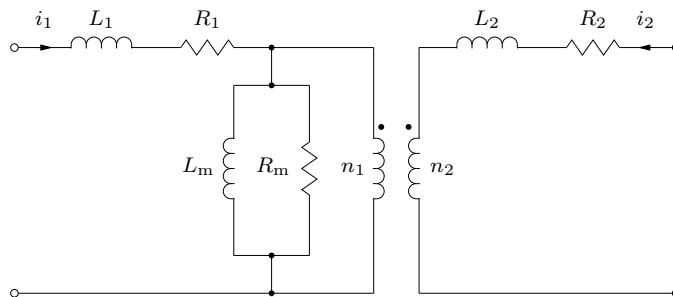
**Library** Electrical / Transformers

### 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 ( $\Omega$ ).

#### 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 ( $\Omega$ ) is referred to the primary side.

**Initial current**

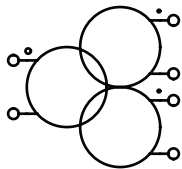
A two-element vector containing the initial currents on the primary side  $i_1$  and the secondary side  $i_2$ , in amperes (A). The currents are considered positive if flowing into the transformer at the marked terminals. The default is  $[0 \ 0]$ .

## Linear Transformer (3 Windings)

**Purpose** Single-phase transformer with winding resistance and optional core loss

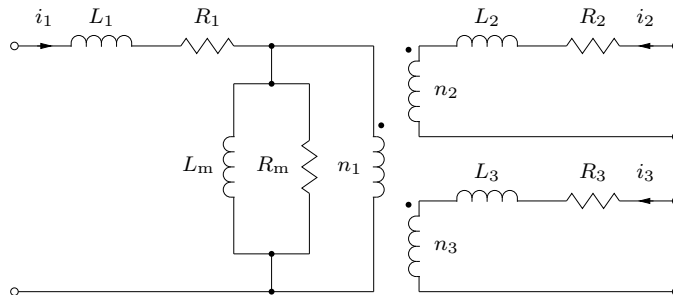
**Library** Electrical / Transformers

### 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 ( $\Omega$ ).

#### 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 ( $\Omega$ ) is referred to the primary side.

**Initial current**

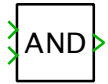
A three-element vector containing the initial currents on the primary side  $i_1$ , the secondary side  $i_2$  and the tertiary side  $i_3$ , in amperes (A). The currents are considered positive if flowing into the transformer at the marked terminals. The default is [0 0 0].

## 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 $i$

The  $i$ th input signal.

#### Output

The block 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  $\Phi$  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

#### Permeance

Magnetic permeance of the flux path, in webers per ampere-turn (Wb/A).

#### 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.

## Magnetic Port

**Purpose** Add magnetic connector to subsystem

**Library** Magnetic

### Description



Magnetic ports are used to establish magnetic connections between a schematic and the subschematic of a subsystem (see page 603). 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.

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.

### Magnetic Ports in a Top-Level Schematic

In PLECS Blockset, if a Magnetic Port is placed in a top-level schematic, the PLECS Circuit block in the Simulink model will show a corresponding magnetic terminal, which may be connected with other magnetic terminals of the same or a different PLECS Circuit block. The Magnetic Port is also assigned a unique *physical port number*. Together with the parameter **Location on circuit block** the port number determines the position of the magnetic terminal of the PLECS Circuit block.

For compatibility reasons you can also place an Magnetic Port in a top-level schematic in PLECS Standalone. However, since there is no parent system to connect to, such a port will act like an isolated node.

### Parameter

#### Width

The width of the connected magnetic path. The default auto means that the width is inherited from connected components.

#### Port number

If a Magnetic Port is placed in a top-level schematic in PLECS Blockset, this parameter determines the position, at which the corresponding terminal appears on the PLECS Circuit block.

#### Location on circuit block

If a Magnetic Port is placed in a top-level schematic in PLECS Blockset, this parameter specifies the side of the PLECS Circuit block on which the

corresponding terminal appears. By convention, left refers to the side on which also input terminals are shown, and right refers to the side on which also output terminals are shown.

## 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.

### Parameter

#### Resistance

Effective magnetic resistance  $R_m$ , in  $\text{A} \cdot (\text{Wb/s})^{-1}$ .

### Probe Signal

#### MMF

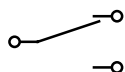
The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).

## Manual Double Switch

**Purpose** Manual changeover switch with two positions

**Library** Electrical / Switches

### Description



This changeover switch provides an ideal switch with two positions. The position can be toggled between upper (0) and lower (1) by double-clicking the component. Which input signal currently is connected to the output signal is indicated in the icon.

### Parameter

#### Switch position

The current position of the switch. Possible values are 0 (upper position) and 1 (lower position).

### Probe Signal

#### Switch position

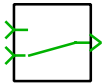
State of the internal switch. The signal outputs 0 if the switch is in the upper position, and 1 if it is in the lower position.

## Manual Signal Switch

**Purpose** Switch between two input signals by double-clicking the component

**Library** Control / Discontinuous

### Description



The Signal Switch can be in the on-state or the off-state, connecting the first or the second input to the output, respectively. The state can be toggled by double-clicking the component. Which input signal currently is connected to the output signal is indicated in the icon.

### Parameter

#### State

The current switch state, either “on” or “off”.

### Probe Signals

#### Inputs

The block input signals.

#### Output

The block output signal.

#### Switch position

The current 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.

## Manual Switch

**Purpose** Manual on-off switch

**Library** Electrical / Switches

**Description**



This Switch provides an ideal short or open circuit between its two electrical terminals. The position can be toggled between upper (0, off) and lower (1, on) by double-clicking the component. Which input signal currently is connected to the output signal is indicated in the icon.

**Parameter**

**Switch position**

The current position of the switch. Possible values are 0 (upper position, off) and 1 (lower position, on).

**Probe Signal**

**Switch position**

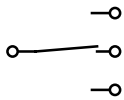
State of the internal switch. The signal outputs 0 if the switch is in the upper position (off), and 1 if it is in the lower position (on).

## Manual Triple Switch

**Purpose** Manual changeover switch with three positions

**Library** Electrical / Switches

**Description** This changeover switch provides an ideal switch with three positions. The position can be toggled between upper (-1), middle (0) and lower (1) by double-clicking the component. Which input signal currently is connected to the output signal is indicated in the icon.



**Parameter** **Switch position**  
The current position of the switch. Possible values are -1 (upper position), 0 (middle position) and 1 (lower position).

**Probe Signal** **Switch position**  
State of the internal switch. 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.



## Mass

**Purpose** Model a sliding body with inertia

**Library** Mechanical / Translational / Components

**Description** This component models a sliding body with inertia and two rigidly connected flanges. The translational speed is determined by the equation



$$\frac{d}{dt}v = \frac{1}{m} \cdot (F_1 + F_2)$$

where  $F_1$  and  $F_2$  are the forces acting at the two flanges *towards* the body.

### Parameters

#### Mass

The mass  $m$ , in kg.

#### Initial speed

The initial speed  $v_0$ , in  $\frac{m}{s}$ .

#### Initial position

The initial position  $x_0$ , in meters. May be specified in order to provide proper initial conditions if absolute positions are measured anywhere in the system. Otherwise, this parameter can be left blank.

### Probe Signals

#### Speed

The speed of the body.

#### Position

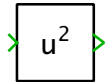
The absolute position of the body.

## 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**

### Function

Chooses which function is applied to the input signals. Available functions are

- 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  $y = \text{rem}(u, v)$

mod and rem both return the floating-point remainder of  $u/v$ . If  $u$  and  $v$  have different signs, the result of rem has the same sign as  $u$  while the result of mod has the same sign as  $v$ .

**Probe Signals**

### Input $i$

The  $i$ th input signal.

### Output

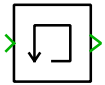
The block output signal.

# 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

- The Memory block implicitly has a *semi-continuous* sample time. If you need to specify a sample time explicitly, please use the Delay block instead (see page 360).
- If a variable-step solver is used the delay time of the Memory block also varies. If you use the Memory block to decouple an algebraic loop, this will produce a dead-time with an uncontrolled jitter. Please consider using a low pass filter instead.

---

**Parameter** **Initial condition**  
The initial output during the first major time step.

**Probe Signals** **Input**  
The input signal.

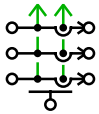
**Output**  
The output signal.

## 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**

**Voltage measurement**

Determine whether the voltages are measured from line to ground or from line to line.

**Probe Signals**

**Measured voltage**

The measured voltages as a vector with three elements.

**Measured current**

The measured currents as a vector with three elements.

## Minimum / Maximum

**Purpose** Output input signal with highest resp. lowest value

**Library** Control / Math

### Description



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.

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.

### Parameters

#### Operation

Selects between Minimum and Maximum as described above.

#### Number of inputs

The number of inputs.

### Probe Signals

#### Input $i$

The  $i$ th input signal.

#### Output

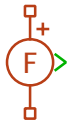
The block output signal.

## 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 583) 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 Signal**

**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 Signal** **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.

---

**Note** An MMF source may not be short-circuited or connected in parallel to a permeance or any other MMF source.

---

### Parameter

#### Allow state-space inlining

**For expert use only!** When set to on and the input signal is a linear combination of magnetic measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

### Probe Signal

#### MMF

The magneto-motive force of the source, in ampere-turns (A).

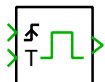
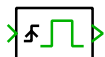


# Monoflop

**Purpose** Generate pulse of specified duration when triggered

**Library** Control / Logical

**Description** 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.



Depending on the trigger type the behavior is as follows:

### rising

The output is set to 1 for the given duration when the input changes from 0 to a non-zero value.

### falling

The output is set to 1 for the given duration when the input changes from a non-zero value to 0.

### level

The output is set to 1 when the input is a non-zero value. It stays 1 for the given duration after the input returns to 0. With this trigger type, the Monoflop acts like a **Turn-off delay**.

The Monoflop can be retriggered, i.e. if the trigger condition is fulfilled again while the output is 1, the pulse duration is extended.

The pulse duration can be specified statically via a parameter or dynamically during the simulation via an input signal.

## Parameters

### Trigger type

The trigger type as described above.

### Pulse duration source

Specifies whether the duration is determined by the **Pulse duration** parameter (*internal*) or by an external input signal (*external*).

### Pulse duration

The duration for which the output is set to 1, in seconds. If set to 0, the Monoflop is disabled and outputs 0 at all times.

### Duration rounding (fixed-step)

If the duration is determined by an external signal and the Monoflop is used with a fixed-step solver, this parameter specifies how the duration is rounded to an integer multiple of the fixed step size.

## Probe Signals

### Input

The input signal.

### Output

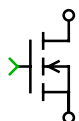
The output signal.

## 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 ( $\Omega$ ). 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 115). If no thermal description is given the losses are calculated based on the voltage drop  $v_{on} = R_{on} \cdot i$ .

**Initial temperature**

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

**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.

**MOSFET switching loss**

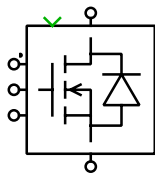
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

## MOSFET Converter (3ph)

**Purpose** 3-phase MOSFET converter

**Library** Electrical / Converters

### Description



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.

You can choose between two different converter models:

- The basic **MOSFET Converter** is modeled using the component MOSFET with Diode (see page 496). 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 493) and Diode (see page 361) components. In this model you may specify forward voltages and on-resistances separately for the MOSFETs and diodes.

---

**Note** Due to the switching conditions of the MOSFET with Diode (see page 496), this device *cannot* be turned off actively while the current is exactly zero. This may result in unexpected voltage waveforms if the converter is not loaded.

To resolve this problem, either use the **MOSFET Converter with Parasitics**, or allow a small non-zero load current to flow by connecting a large load resistance to the converter.

---

**Parameters** For a description of the parameters see the documentation of the MOSFET with Diode (on page 496), the MOSFET (on page 493) and the Diode (on page 361).

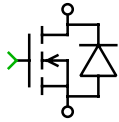
**Probe Signals** The two-level MOSFET converters provide six or twelve probe signals, each a vector containing the appropriate quantities of the individual devices: voltage, current, conductivity, conduction loss and switching loss. The vector elements are ordered top-to-bottom, left-to-right: a+, a-, 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.

This device is modeled as a single ideal switch that closes when the gate signal is not zero or the voltage becomes negative and opens when the gate signal is zero and the current becomes positive.

---

**Note** Due to the switching conditions described above, this device *cannot* be turned off actively while the current is exactly zero. This may result in unexpected voltage waveforms if the device is used e.g. in an unloaded converter.

To resolve this problem, either use an individual MOSFET (see page 493) with an individual anti-parallel Diode (see page 361), or allow a small non-zero load current to flow by connecting a large load resistance to the converter.

---

### Parameters

#### 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.

#### Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapters “Thermal Modeling” (on page 115) and “Losses of Semiconductor Switch with Diode” (on page 142).

#### Initial temperature

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

**Probe Signals****Device voltage**

The voltage measured between drain and source. The device voltage can never be negative.

**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.

**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**

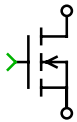
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

## 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 493) 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 436).

---

### 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 ( $\Omega$ ). The default is 0.

#### Off-resistance

The resistance  $R_{off}$  of the blocking device, in ohms ( $\Omega$ ). The default is 1e6. This parameter may be set to `inf` unless multiple MOSFETs are connected in series.



**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  $L_\sigma$  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**

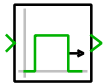
Conduction state of the internal switch. The signal outputs 0 when the MOSFET is blocking, and 1 when it is conducting.

## 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 516).

### 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

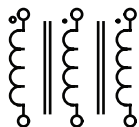
The filtered output signal.

# 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 460). 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 412).

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**

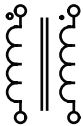
The voltage measured across winding  $i$ , in volts (V).

## Mutual Inductance (2 Windings)

**Purpose** Magnetic coupling between two lossy windings

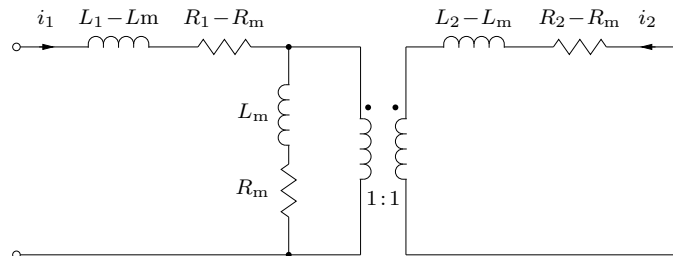
**Library** Electrical / Transformers

### 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 ( $\Omega$ ).

#### Mutual inductance

The mutual inductance  $L_m$ , in henries (H).

#### Mutual resistance

The mutual resistance  $R_m$ , in ohms ( $\Omega$ ).

#### 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

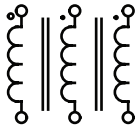
The voltage measured across winding  $i$ , in volts (V).

## Mutual Inductance (3 Windings)

**Purpose** Magnetic coupling between three lossy windings

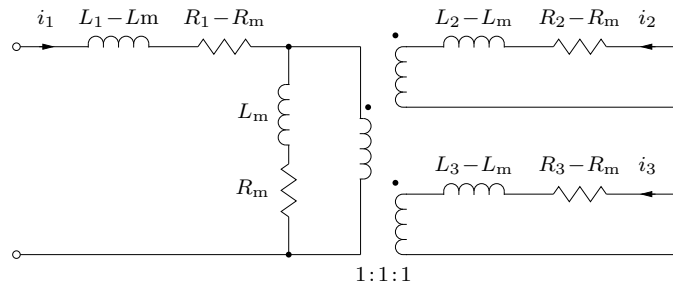
**Library** Electrical / Transformers

### Description



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.

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 terminals of the secondary and tertiary windings are marked with dots.

### Parameters

#### 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).

#### 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 ( $\Omega$ ).

#### Mutual inductance

The mutual inductance  $L_m$ , in henries (H).

#### Mutual resistance

The mutual resistance  $R_m$ , in ohms ( $\Omega$ ).

**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).

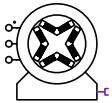


## Non-Excited Synchronous Machine

**Purpose** Non-excited synchronous machine configurable with lookup tables

**Library** Electrical / Machines

### Description



This three-phase synchronous machine has a solid rotor with optional permanent magnets. Magnetization, saliency, saturation and cross-coupling are modeled by means of corresponding flux linkage and incremental inductance lookup tables.

The machine can operate as either a motor or generator. If the mechanical torque has the same sign as the rotational speed, the machine is operating in motor mode; otherwise it is in generator mode. In the component icon, phase a 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 **Subsystem** submenu of the **Edit** menu. If you want to make changes, you must first choose **Break library link** both from the same menu.

### Electrical System

The model is realized by means of the Voltage Behind Reactance (VBR) formulation and is therefore appropriate to simulate switching dead-time and failure modes.

Electrical equation expressed in synchronous frame by means of space-vector notation:

$$\vec{v}_s = R_s \cdot \vec{i}_s + (\mathbf{L}_{\sigma s} + \mathbf{L}_{mi}) \cdot \left( \frac{d\vec{i}_s}{dt} + j \cdot \omega \cdot \vec{i}_s \right) + \vec{e}_s$$

$$\vec{e}_s = j \cdot \omega \cdot \vec{\varphi}_m - \mathbf{L}_{mi} \cdot j \cdot \omega \cdot \vec{i}_s$$

with

$$\vec{x} = \begin{bmatrix} x_d \\ x_q \end{bmatrix}$$

where  $j \cdot \vec{x}$  rotates the synchronous frame vector,  $\vec{x}$ , clockwise by  $90^\circ$ .

These equations are transformed back into the stationary frame to control the VBR network. The zero-sequence impedance of the machine is set to  $L_{ss}$ .

### Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p (\varphi_d i_q - \varphi_q i_d)$$

### Mechanical System

Mechanical rotor speed  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\dot{\theta}_m = \omega_m$$

## Parameters

### General

#### Stator resistance

Armature or stator resistance  $R_s$  in  $\Omega$ .

#### Stator leakage inductance

Leakage inductance of stator windings in henries (H). Stator leakage must be set to a non-zero value.

#### Number of pole pairs

Number of pole pairs  $p$ .

#### 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).  $i_{c,0}$  is calculated assuming a neutral connection.

### Magnetizing Inductance

#### Id lookup vector

d-axis current vector serving as input values to flux linkage and incremental inductance lookup tables. Must be a vector with 2 or more elements, and monotonically increasing, i.e.  $[0 \dots i_{d,max}]$ . The values are in amperes (A).

#### Iq lookup vector

q-axis current vector serving as input values to flux linkage and incremental inductance lookup tables. Must be a vector with 2 or more elements,

and monotonically increasing, i.e.  $[0 \dots i_{q,\max}]$ . The values are in amperes (A).

**Psid (Id, Iq) lookup table**

d-axis flux linkage lookup table (2D). The number of rows and columns must match the size of the d- and q-axis currents, respectively. The values are in Volt-seconds (Vs).

**Psiq (Id, Iq) lookup table**

q-axis flux linkage lookup table (2D). The number of rows and columns must match the size of the d- and q-axis currents, respectively. The values are in Volt-seconds (Vs).

**Lmidd (Id, Iq) lookup table**

d-axis self incremental inductance lookup table (2D). The number of rows and columns must match the size of the d- and q-axis currents, respectively. If Lmi data is not specified, or set to [], the incremental inductance is calculated from the flux linkage data. The values are in henries (H).

**Lmiqq (Id, Iq) lookup table**

q-axis self incremental inductance lookup table (2D). The number of rows and columns must match the size of the d- and q-axis currents, respectively. If Lmi data is not specified, or set to [], the incremental inductance is calculated from the flux linkage data. The values are in henries (H).

**Lmidq (Id, Iq) lookup table**

Mutual incremental inductance lookup table (2D). The number of rows and columns must match the size of the d- and q-axis currents, respectively. If Lmi data is not specified, or set to [], the incremental inductance is calculated from the flux linkage data. The values are in henries (H).

**Generated table size [d, q]**

User-specified dimension to generate derived current vectors and corresponding flux linkage and incremental inductance lookup tables.

If left empty, the supplied data is used as-is. If specified, the dimensions of the rows and columns must be 2 or more.

Specifying a scalar value, n, will generate equally spaced, n-element d- and q-axis current vectors. The corresponding 2D lookup tables for flux linkage and incremental inductance are also generated.

Specifying a vector, [m,n], will generate equally spaced d- and q-axis current vectors. The d-axis current vector will have m elements and the q-axis current vector will have n elements. The corresponding 2D lookup tables for flux linkage and incremental inductance are also generated.

The size of the generated tables affect the model initialization and simulation speeds. A smaller size leads to faster model initialization and simulation speeds, but lower resolution in the generated tables. A larger size increases the resolution but adversely affects the model initialization and simulation speeds. Care must be taken when configuring this parameter.

**Current out of range**

Configure to ignore, warn, warn and pause simulation, or generate error and stop simulation if the d-axis or q-axis currents are outside the specified range.

**Mechanical****Inertia**

Combined rotor and load inertia  $J$  in  $\text{Nms}^2$ .

**Friction coefficient**

Viscous friction  $F$  in  $\text{Nms}$ .

**Initial rotor speed**

Initial mechanical rotor speed  $\omega_{m,0}$  in radians per second ( $\text{s}^{-1}$ ).

**Initial rotor position**

Initial mechanical rotor angle  $\theta_{m,0}$  in radians.

**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  $\varphi_d$  and  $\varphi_q$  in the rotating reference frame.

**Rotational speed**

The rotational speed  $\omega_m$  of the rotor in radians per second ( $\text{s}^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  in radians.

**Electrical torque**

The electrical torque  $T_e$  of the machine in  $\text{Nm}$ .

**References**

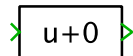
- H. Bühler, “Réglage de systèmes d’électronique de puissance, vol 1: Théorie”, Presses Polytechniques et universitaires romandes, Lausanne, 1997.
- H. Bühler, “Réglage de systèmes d’électronique de puissance, vol 2: Entraînements réglés”, Presses Polytechniques et universitaires romandes, Lausanne, 1997.

## Offset

**Purpose** Add constant offset to input signal

**Library** Control / Math

**Description** The Offset block adds a constant to the input signal.



**Parameter**

**Offset**

The offset to add to the input signal. This value may be negative to subtract an offset from the signal.

**Output data type**

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

**Probe Signals**

**Input**

The input signal.

**Output**

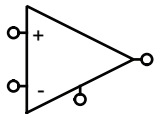
The output signal.

## 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 p1OpAmps and p1ActiveLowPass demonstrate different applications with op-amps.

**Parameter**

**Open-loop gain**

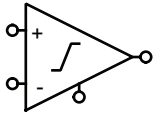
The voltage gain of the Op-Amp. The default is 1e6.

## 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 p1OpAmps 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

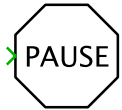
A two-element vector containing the minimum and maximum output voltage  $V_{\min}$  and  $V_{\max}$  in volts (V). The default is  $[-10\ 10]$ .

## Pause / Stop

**Purpose** Pauses or stops the simulation when the input becomes non-zero

**Library** System

**Description** This block pauses or stops the simulation when the input signal changes from zero to a non-zero value.



---

**Note** In PLECS Standalone, pause blocks are ignored during analyses and simulation scripts.

---

**Parameter** **Action**  
Use pause or stop to specify whether the block should pause or stop the simulation. To disable the block, use ignore.

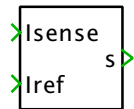


## Peak Current Controller

**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  $I_{\text{sense}}$  input exceeds the  $I_{\text{ref}}$  input, the output is reset.

### Parameters

#### Switching frequency

The switching frequency of the output signal.

#### 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.

#### Maximum duty cycle

This defines the maximum permissible duty cycle of the switch output. If  $I_{\text{sense}} < I_{\text{ref}}$ , the output will turn off if the duty cycle exceeds this maximum value. The maximum duty cycle must be less than 100 %.

#### Slope compensation

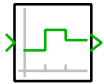
Slope compensation can be applied to ensure stability when the output duty cycle exceeds 50 %. Entering a parameter,  $I_{\text{slope}}$ , reduces  $I_{\text{ref}}$  during each switching cycle as follows:  $I'_{\text{ref}} = I_{\text{ref}} - I_{\text{slope}} \cdot t/T_s$ , where  $t$  is the time elapsed from the start of the switching cycle and  $T_s$  is the switching period. Slope compensation can be omitted by setting  $I_{\text{slope}}$  to 0.

## 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 500) 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 517).

### Parameter

#### 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 36).

### Probe Signals

#### Input

The input signal.

#### Output

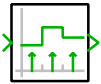
The filtered output signal.

## Periodic Impulse Average

**Purpose** Periodically average Dirac impulses over specified time

**Library** Control / Filters

### Description



This block periodically averages an input signal  $u$  consisting of a series of Dirac impulses  $\delta$ . 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:

$$y(t) = \frac{1}{T} \int_{t-T}^t u(\tau) d\tau \cdot \text{rect}\left(t - \frac{n + 1/2}{T}\right)$$

The block is suited to determine average switching losses of power semiconductors. To determine average conduction losses, use the Periodic Average (see page 516).

### Parameter

#### 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 36).

### Probe Signals

#### Input

The input signal.

#### Output

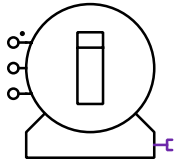
The filtered output signal.

## Permanent Magnet Synchronous Machine

**Purpose** Synchronous machine excited by permanent magnets

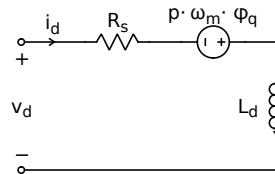
**Library** Electrical / Machines

**Description** This three-phase permanent magnet synchronous machine has a sinusoidal back EMF.

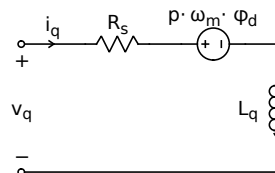


The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a is marked with 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  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\dot{\theta}_m = \omega_m$$

## Parameters

### Model

Implementation in the rotor reference frame or as a voltage behind reactance.

### Stator resistance

Armature or stator resistance  $R_s$  in  $\Omega$ .

**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  $\varphi'_m$  in Vs induced by the magnets in the stator windings.

**Inertia**

Combined rotor and load inertia  $J$  in Nms<sup>2</sup>.

**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<sup>-1</sup>).

**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).

**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  $\varphi_d$  and  $\varphi_q$  in the rotating reference frame in Vs:

$$\varphi_q = L_q i_q$$

$$\varphi_d = L_d i_d + \varphi'_m$$

**Rotational speed**

The rotational speed  $\omega_m$  of the rotor in radians per second (s<sup>-1</sup>).

**Rotor position**

The mechanical rotor angle  $\theta_m$  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 333) with a sinusoidal back EMF. The parameters need to be converted as follows:

$$L - M = L_d = L_q$$

$$K_E = -\phi'_m \cdot p$$

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 330). The sophisticated BLDC machine can be configured as a PMSM with sinusoidal back EMF if the parameters are converted as follows:

$$K_{c,n} = [0]$$

$$K_{s,n} = [-\phi'_m \cdot p]$$

$$L_0 - M = \frac{L_d + L_q}{2}$$

$$L_{c,n} = [0 \quad L_d - L_q]$$

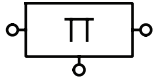
$$L_{s,n} = [0 \quad 0]$$

## Pi-Section Line

**Purpose** Single-phase pi-section transmission line

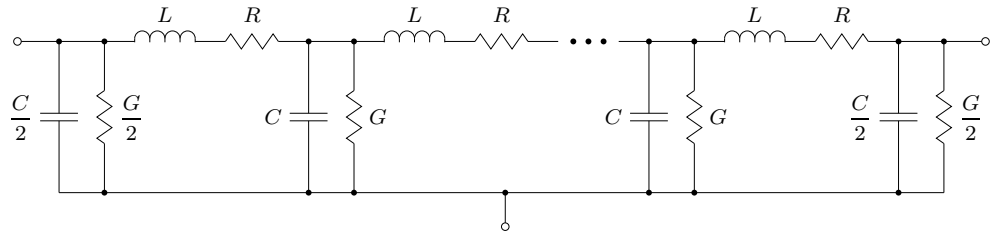
**Library** Electrical / Passive Components

**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 ( $\Omega$ /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**

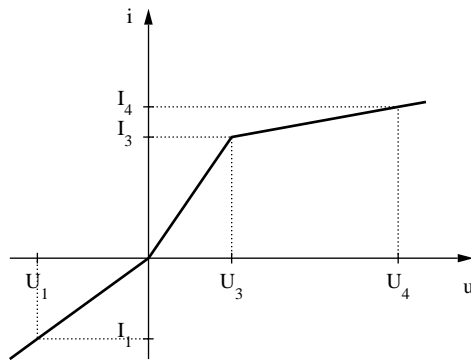
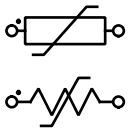
A scalar value specifying the initial voltage of all capacitors at simulation start, in volts (V).

## Piece-wise Linear Resistor

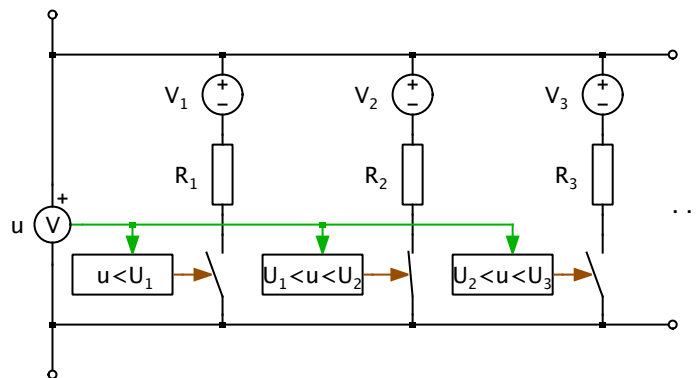
**Purpose** Resistance defined by voltage-current pairs

**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).

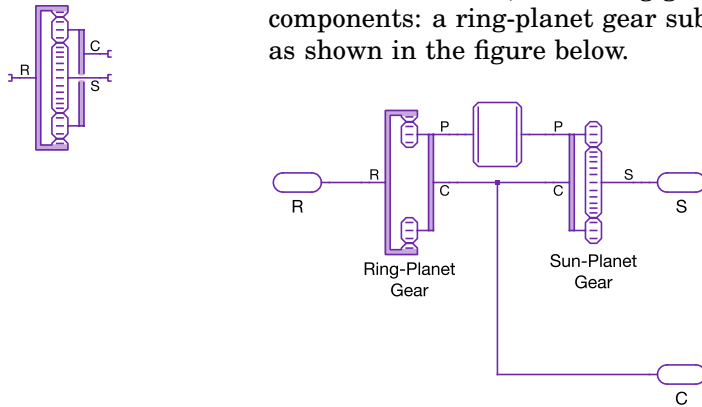
## Planetary Gear Set

**Purpose** Ideal planetary gear set

**Library** Mechanical / Rotational / Components

**Description**

This component models a planetary gear set with a sun gear, planet gears connected via a carrier, and a ring gear. The component is divided into two sub-components: a ring-planet gear subsystem and a sun-planet gear subsystem, as shown in the figure below.



The planetary gear set has three external shafts: ring gear shaft (R), sun gear shaft (S), and carrier shaft (C). The relation between the angular speeds of the gears and carrier are described by the following equations:

$$N_s \omega_s + N_p \omega_p - (N_s + N_p) \omega_c = 0$$

$$N_r \omega_r - N_p \omega_p - (N_r - N_p) \omega_c = 0$$

where  $N_r$ ,  $N_s$ , and  $N_p$  correspond to the number of teeth on the ring, sun, and each planet gear respectively, and  $\omega_r$ ,  $\omega_s$ ,  $\omega_p$ , and  $\omega_c$  correspond to the angular speed of the ring gear, sun gear, planet gears, and carrier respectively.

These equations can further be simplified to

$$N_s \omega_s + N_r \omega_r = (N_s + N_r) \omega_c$$

The model includes an internal lumped moment of inertia representing the planet gears (which is set to zero by default). The moments of inertia of the

ring and sun gears and the carrier can be modeled by connecting an Inertia (see page 462) to the corresponding shaft.

## Parameters

### Main

#### Number of sun teeth

Number of teeth on the sun gear.

#### Number of planet teeth

Number of teeth on each planet gear.

#### Number of ring teeth

Number of teeth on the ring gear.

### Planet gear

#### Moment of inertia of planet gear

Combined planet gear inertia  $J$  in  $\text{Nms}^2$ .

#### Initial speed of planet gear

Initial angular speed of each planet gear ( $\omega_p$ ) in  $\frac{\text{rad}}{\text{s}}$ .

## Probe Signals

### Sun gear speed

Angular speed of sun gear ( $\omega_s$ ) in  $\frac{\text{rad}}{\text{s}}$ .

### Planet gear speed

Angular speed of each planet gear ( $\omega_p$ ) in  $\frac{\text{rad}}{\text{s}}$ .

### Carrier

Angular speed of carrier ( $\omega_c$ ) in  $\frac{\text{rad}}{\text{s}}$ .

### Ring gear speed

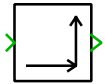
Angular speed of ring gear ( $\omega_r$ ) in  $\frac{\text{rad}}{\text{s}}$ .

## 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  $[x, y]$ :



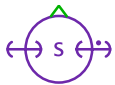
$$\begin{aligned}x &= r * \cos(\theta) \\y &= r * \sin(\theta)\end{aligned}\quad \text{where } \theta \text{ is in radians.}$$

## Position Sensor

**Purpose** Output measured absolute or relative position as signal

**Library** Mechanical / Translational / Sensors

**Description** The Position Sensor measures the relative position of the flange marked with a dot with respect to the other flange. If the other flange is connected to the reference frame, the absolute position is measured.




---

**Note** Speed and position sensors are ideally compliant. Hence, if multiple speed or position sensors are connected in series the speed or position measured by an individual sensor is undefined. This produces a run-time error.

---

### Parameters

#### Second flange

Controls whether the second flange is accessible or connected to the translational reference frame.

#### Initial position

The position at simulation start.

### Probe Signal

#### Position

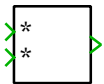
The measured position.

## Product

**Purpose** Multiply and divide scalar or vectorized input signals

**Library** Control / Math

**Description** The Product block multiplies or divides input signals. If the division operator / is used, the reciprocal of the input signal is used for multiplication.



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.

**Parameter** **List of operators 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 inputs to be multiplied.

**Output data type**  
*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

**Probe Signals** **Input  $i$**   
 The  $i$ th input signal.

**Output**  
 The block output signal.

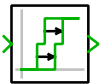


## Pulse Delay

**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 signal can be a scalar or vector.

Whenever a change of an input signal is detected at a simulation time  $t$ , the Pulse Delay records the new signal value in an internal buffer and schedules an event that forces the solver to make a step exactly at the simulation time  $t + T_D$  in order to output the delayed input value.

A typical application of the Pulse Delay is to delay the pulses of a modulator.

### Note

- The Pulse Delay should *not* be used to delay continuous signals as this will lead to excessive memory consumption. Besides, the output of the Pulse Delay is always piece-wise constant. To delay continuously changing signals, use the continuous Transport Delay (see page 686).
- The Pulse Delay should also *not* be used to delay signals that have a fixed sample time. To delay such signals, use a Zero Order Hold (see page 725) or a Delay (see page 360) depending on the duration of the delay with respect to the sample time of the input signal.

Suppose that the input signal has a sample time  $T_s$  and you want to delay it by a time  $T_D$ . Calculate the delay order  $O = \lfloor \frac{T_D}{T_s} \rfloor$  (i.e. the integer part of the division) and an offset time  $T_o = \text{mod}(T_D, T_s)$ . If  $O$  is zero, use a Zero Order Hold with the sample time  $[T_s, T_o]$ . If  $O$  is greater than zero, use a Delay with the sample time  $[T_s, T_o]$  and the delay order  $O$ . For more information regarding sample times, see “Sample Times” (on page 36).

### Parameters

#### Time delay $T_d$

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.



### Parameters

#### 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.

#### Output data type

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

### Probe Signal

#### Output

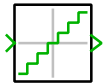
The output signal of the pulse generator.

## 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**

**Quantization interval**

The quantum  $q$  used in the mapping function.

**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.

**Probe Signals**

**Input**

The input signal.

**Output**

The output signal.

## Rack and Pinion

**Purpose** Ideal conversion between translational and rotational motion

**Libraries** Mechanical / Translational / Components  
Mechanical / Rotational / Components

**Description** The Rack and Pinion models an ideal converter between translational and rotational motion. The relation between the torque, force and speeds of the two flanges is described with the following equations:



$$v = R \cdot \omega$$

$$\tau = R \cdot F$$

where  $R$  is the pinion radius.

**Parameter**

**Pinion radius**

The pinion radius  $R$ . A negative value will cause the flanges to move in opposite directions.

## 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.



### Parameters

#### 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 `inf` the output signal is unlimited.

### Probe Signal

#### Output

The output signal of the pulse generator.

# Random Numbers

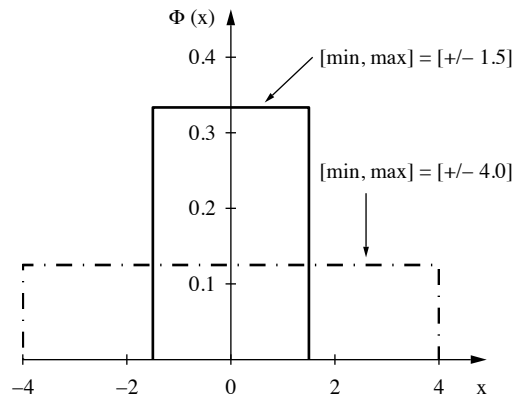
**Purpose** Generate uniformly distributed random numbers

**Library** Control / Sources

## Description



The Random Numbers block generates uniformly distributed random numbers. The boundaries of the generated values can be configured in the component dialog. The figure below illustrates the distribution for two different sets of parameters. The seed of the generator initializes the algorithm at the sim-



ulation start. For the same seed, the sequence of random numbers is reproduced in every simulation run. If this behavior is undesired the system time can be used as a seed. To minimize correlation effects, it is recommended to use different seeds if multiple random generators are used in one model.

## Parameters

### Minimum

The lower boundary of the random numbers.

### Maximum

The upper boundary of the random numbers.

### Seed

The seed used to initialize the Random Numbers generator.

### Sample Time

The sampling period used for generating random output values.

## Reference

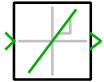
Mersenne Twister: [http://en.wikipedia.org/wiki/Mersenne\\_twister](http://en.wikipedia.org/wiki/Mersenne_twister)

## 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.

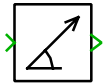


## Rectangular to Polar

**Purpose** Convert Cartesian coordinates to polar coordinates

**Library** Control / Transformations

**Description** This block transforms a signal representing rectangular coordinates  $[x, y]$  into polar coordinates  $[r, \theta]$ :



$$r = \sqrt{x^2 + y^2}$$
$$\theta = \text{atan2}(x, y)$$

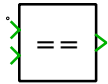
$\theta$  is calculated in the range  $-\pi \leq \theta \leq \pi$ .

## 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

- equal (==),
- unequal (~=),
- less (<),
- less or equal (<=),
- greater or equal (>=),
- greater (>).

**Probe Signals**

**Input**

The input signals.

**Output**

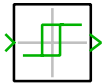
The output signal.

# 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 input 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.

### Output data type

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

## Probe Signals

### Input

The block input signal.

### Output

The block output signal.

## Resistor

**Purpose** Ideal resistor

**Library** Electrical / Passive Components

**Description**



This component provides an ideal resistor between its two electrical terminals. See section “Configuring PLECS” (on page 41) for information on how to change the graphical representation of resistors.

**Parameter**

**Resistance**

The resistance in ohms ( $\Omega$ ). All positive and negative values are accepted, including 0 and inf ( $\infty$ ). The default is 1.

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.

**Probe Signals**

When the resistor is probed, a small dot in the component icon marks the positive terminal.

**Resistor voltage**

The voltage measured across the resistor from the positive to the negative terminal, in volts (V).

**Resistor current**

The current flowing through the resistor, in amperes (A). A current entering the resistor at the positive terminal is counted positive.

**Resistor power**

The power consumed by the resistor, in watts (W).

## Rotational Algebraic Component

**Purpose** Define an algebraic constraint in terms of torque and angular speed

**Library** Mechanical / Rotational / Components

### Description



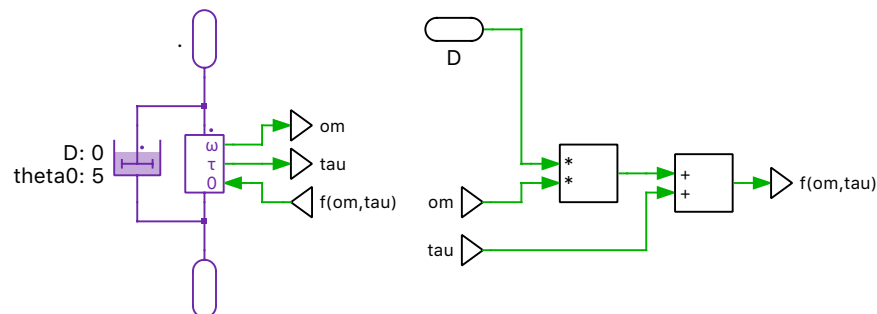
The Rotational Algebraic Component enforces an arbitrary algebraic constraint involving torque and angular speed.

The output signal “ $\omega$ ” measures the angular speed of the marked flange with respect to the unmarked one. The output signal “ $\tau$ ” measures the torque flow from the unmarked towards the marked flange. The two output signals must affect the input signal “0” by means of a direct feedthrough path. The component ensures that the input signal is zero at all times.

The direct feedthrough path defines a function  $f(\omega, \tau)$ , which in turn implicitly determines the characteristic curve of the component through the constraint  $f(\omega, \tau) = 0$ . For instance, the choice  $f(\omega, \tau) := \tau + D \cdot \omega$  causes the Rotational Algebraic Component to act as a Rotational Damper (see page 547) with damping constant  $D$ .

The Rotational Algebraic Component offers no direct way to specify an initial displacement. In case you need to do so, place a Rotational Damper with zero damping constant in parallel to the component and set the initial displacement property thereof.

By way of illustration, the following schematic shows a possible implementation of a rotational damper with variable damping constant and prescribed initial displacement:



---

**Note** The Rotational Algebraic Component creates an algebraic loop. See section “Block Sorting” (on page 29) for more information on algebraic loops.

---

## Probe Signals

### **Component torque**

The torque flow from the unmarked towards the marked flange.


### **Component speed**

The angular speed of the marked flange with respect to the unmarked one.

### **Component power**

The power consumed by the component.

## Rotational Backlash

<b>Purpose</b>	Ideal rotational backlash
<b>Library</b>	Mechanical / Rotational / Components
<b>Description</b>	 <p>The Rotational Backlash models an ideal symmetrical, two-sided Hard Stop (see page 550) in a rotational system, which restricts the relative displacement of the two flanges between an upper and lower limit of <math>\pm \frac{b}{2}</math>. While the displacement is within the limits, no torque is transmitted. When the displacement hits either limit, the flanges become rigidly connected until the transmitted torque reverses.</p>
<b>Parameters</b>	<p><b>Total backlash</b> The total permitted displacement <math>b</math> between the flanges, in radians.</p> <p><b>Initial displacement</b> The initial displacement of the flanges, in radians. May be specified in order to provide proper initial conditions if absolute angles are measured anywhere in the system. Otherwise, this parameter can be left blank.</p>
<b>Probe Signals</b>	<p><b>Torque</b> The transmitted torque flowing from the unmarked to the marked flange, in Newton meters (Nm).</p> <p><b>Displacement</b> The displacement of the marked flange with respect to the unmarked flange, in radians.</p> <p><b>State</b> The internal state of the component: -1 in lower limit, 0 inside limits, +1 in upper limit.</p>

## Rotational Clutch

**Purpose** Ideal rotational clutch

**Library** Mechanical / Rotational / Components

**Description**



The Rotational Clutch models an ideal clutch in a rotational system. When engaged, it makes an ideally rigid connection between the flanges; when disengaged, it transmits zero torque. The clutch engages when the input signal becomes non-zero and disengages when the input signal becomes zero.

**Parameters**

**Initial state**

The initial state (engaged/disengaged) of the clutch.

**Initial displacement**

The initial displacement of the flanges, in radians. May be specified in order to provide proper initial conditions if absolute angles are measured anywhere in the system. Otherwise, this parameter can be left blank.



## Rotational Damper

**Purpose** Ideal viscous rotational damper

**Library** Mechanical / Rotational / Components

**Description** The Rotational Damper models an ideal linear damper in a rotational system described with the following equations:



$$\tau = -D \cdot \omega$$

where  $\tau$  is the torque flow from the unmarked towards the marked flange and  $\omega$  is the angular speed of the marked flange with respect to the unmarked one.

**Parameters**

**Damper constant**

The damping (viscous friction) constant  $D$ , in  $\frac{\text{Nms}}{\text{rad}}$ .

**Initial displacement**

The initial displacement of the flanges, in radians. May be specified in order to provide proper initial conditions if absolute angles are measured anywhere in the system. Otherwise, this parameter can be left blank.

## Rotational Friction

**Purpose** Ideal rotational stick/slip friction

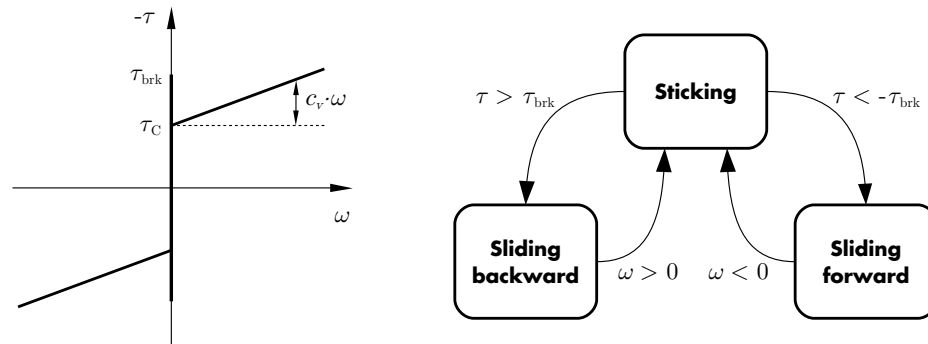
**Library** Mechanical / Rotational / Components

**Description**



The Rotational Friction models any combination of static, Coulomb and viscous friction between two flanges in a rotational system. While the component is stuck, it exerts whatever torque is necessary in order to maintain zero relative speed between the flanges, up to the limit of the breakaway torque  $\tau_{brk}$ . When the breakaway torque is exceeded, the flanges begin sliding against each other, and the component exerts a torque that consists of the Coulomb friction torque  $\tau_C$  and a speed-dependent viscous friction torque  $c_v \cdot \omega$ .

The figure below shows the speed/torque characteristic and the state chart of the component. Note that the friction torque is opposed to the movement, hence the negative sign.



**Parameters**

### Breakaway friction torque

The maximum magnitude of the stiction torque  $\tau_{brk}$ , in Newton meters Nm. Must be greater than or equal to zero.

### Coulomb friction torque

The magnitude of the (constant) Coulomb friction torque  $\tau_C$ , in Newton meters Nm. Must be greater than or equal to zero and less than or equal to the breakaway friction torque.

### Viscous friction coefficient

The proportionality coefficient  $c_v$  that determines the speed dependent viscous friction torque, in  $\frac{\text{Nms}}{\text{rad}}$ .

**Probe Signals****Torque**

The transmitted torque  $\tau$  flowing from the unmarked to the marked flange, in Newton meters (Nm).

**Speed**

The angular speed  $\omega$  of the marked flange with respect to the unmarked flange, in  $\frac{\text{rad}}{\text{s}}$ .

**State**

The internal state of the component: -1 sliding backward, 0 stuck, +1 sliding forward.

## Rotational Hard Stop

**Purpose** Ideal rotational hard stop

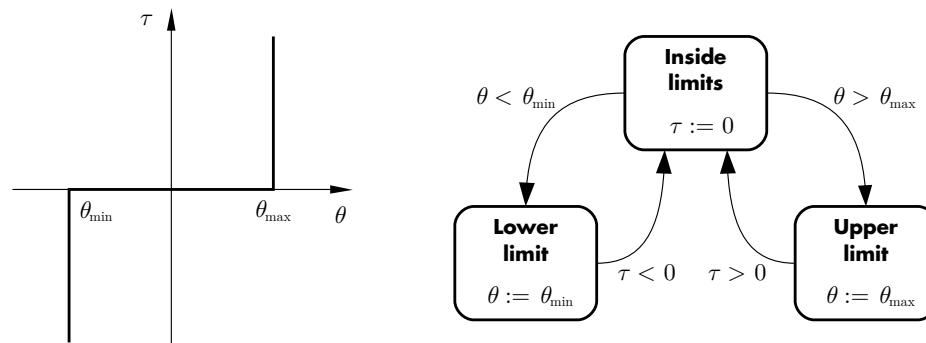
**Library** Mechanical / Rotational / Components

### Description



The Rotational Hard Stop models an ideal one- or two-sided hard stop in a rotational system, which restricts the relative displacement of the two flanges between an upper and lower limit. While the displacement is within the limits, no torque is transmitted. When the displacement hits either limit, the displacement is clamped at the limit and the flanges become rigidly connected until the transmitted torque reverses.

The figure below shows the displacement/torque characteristic and the state chart of the component.



### Parameters

#### Upper limit

The maximum displacement  $\theta_{\max}$  between the flanges. Set to *inf* to disable this limit.

#### Lower limit

The minimum displacement  $\theta_{\min}$  between the flanges. Set to *-inf* to disable this limit.

#### Initial displacement

The initial displacement of the flanges, in radians. May be specified in order to provide proper initial conditions if absolute angles are measured anywhere in the system. Otherwise, this parameter can be left blank.

**Probe Signals****Torque**

The transmitted torque  $\tau$  flowing from the unmarked to the marked flange, in Newton meters (Nm).

**Displacement**

The displacement  $\theta$  of the marked flange with respect to the unmarked flange, in radians.

**State**

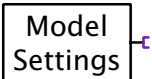
The internal state of the component: -1 in lower limit, 0 inside limits, +1 in upper limit.

## Rotational Model Settings

**Purpose** Configure settings for an individual mechanical model.

**Library** Mechanical / Rotational / Model Settings

### Description



The Rotational Model Settings block lets you configure parameter settings that influence the code generation for a particular mechanical system, see also “Code Generation for Physical Systems” (on page 257).

The block affects the mechanical system that it is attached to by its rotational terminal. At most one Model Settings block may be attached to an individual state-space system. A mechanical model can be split into multiple state-space systems if the underlying model equations are fully decoupled. Note that a rotational system and a translational system that are coupled e.g. with a Rack and Pinion (see page 535) have coupled model equations. See also the options **Enable state-space splitting** and **Display state-space splitting** in the “Simulation Parameters” (on page 103).

### Parameters

#### Switching algorithm

This parameter allows you to choose between two algorithms to determine the clutch states in the generated code. See “Switching Algorithm” (on page 259) for details.

#### Matrix coding style

This setting allows you to specify the format used for storing the state-space matrices for a physical model. When set to `sparse`, only the non-zero matrix entries and their row and column indices are stored. When set to `full`, matrices are stored as full  $m \times n$  arrays. When set to `full (inlined)`, the matrices are additionally embedded in helper functions, which may enable the compiler to further optimize the matrix-vector-multiplications at the cost of increased code size.

#### Topologies

This parameter lets you specify a matrix containing the combinations of clutch states, for which code should be generated, see also “Reducing the Code Size” (on page 258). The columns represent the clutch elements of the physical model and each row represents one combination of clutch states (zero for an open clutch and non-zero for a closed clutch).

The default is an empty matrix `[]`, which means that *all* possible combinations are included.

## Rotational Port

**Purpose** Add rotational flange to subsystem

**Library** Mechanical / Rotational / Components

### Description



Rotational ports are used to establish rotational mechanical connections between a schematic and the subschematic of a subsystem (see page 603). If you copy a Rotational 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.

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.

### Rotational Ports in a Top-Level Schematic

In PLECS Blockset, if a Rotational Port is placed in a top-level schematic, the PLECS Circuit block in the Simulink model will show a corresponding rotational terminal, which may be connected with other rotational terminals of the same or a different PLECS Circuit block. The Rotational Port is also assigned a unique *physical port number*. Together with the parameter **Location on circuit block** the port number determines the position of the rotational terminal of the PLECS Circuit block.

For compatibility reasons you can also place an Rotational Port in a top-level schematic in PLECS Standalone. However, since there is no parent system to connect to, such a port will act like an isolated node.

### Parameter

#### Port number

If a Rotational Port is placed in a top-level schematic in PLECS Blockset, this parameter determines the position, at which the corresponding terminal appears on the PLECS Circuit block.

#### Location on circuit block

If a Rotational Port is placed in a top-level schematic in PLECS Blockset, this parameter specifies the side of the PLECS Circuit block on which the corresponding terminal appears. By convention, `left` refers to the side on which also input terminals are shown, and `right` refers to the side on which also output terminals are shown.

## Rotational Reference

**Purpose** Connect to common rotational reference frame

**Library** Mechanical / Rotational / Components

**Description** The Rotational Reference implements a connection to the rotational reference frame that has a fixed absolute angle of zero.





## Rotational Speed (Constant)

**Purpose** Maintain constant rotational speed

**Library** Mechanical / Rotational / Sources

### Description



The Constant Rotational Speed maintains a constant angular speed between its two flanges regardless of the torque required. The speed is considered positive at the flange marked with a “+”.

---

**Note** A speed source may not be short-circuited or connected in parallel with other speed sources, nor may a speed source directly drive an inertia.

---

### Parameters

#### Second flange

Controls whether the second flange is accessible or connected to the rotational reference frame.

#### Speed

The magnitude of the speed, in  $\frac{\text{rad}}{\text{s}}$ . The default value is 1.

### Probe Signals

#### Torque

The generated torque  $\tau$  flowing from the unmarked to the marked flange, in Newton meters (Nm).

#### Speed

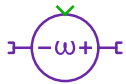
The angular speed, in  $\frac{\text{rad}}{\text{s}}$ .

## Rotational Speed (Controlled)

**Purpose** Maintain variable rotational speed

**Library** Mechanical / Rotational / Sources

### Description



The Controlled Rotational Speed maintains a variable angular speed between its two flanges regardless of the torque required. The speed is considered positive at the flange marked with a “+”. The momentary speed is determined by the signal fed into the input of the component.

---

**Note** A speed source may not be short-circuited or connected in parallel with other speed sources, nor may a speed source directly drive an inertia.

---

### Parameters

#### Second flange

Controls whether the second flange is accessible or connected to the rotational reference frame.

#### Allow state-space inlining

**For expert use only!** When set to on and the input signal is a linear combination of mechanical measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

### Probe Signals

#### Torque

The generated torque  $\tau$  flowing from the unmarked to the marked flange, in Newton meters (Nm).

#### Speed

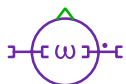
The angular speed, in  $\frac{\text{rad}}{\text{s}}$ .

## Rotational Speed Sensor

**Purpose** Output measured angular speed as signal

**Library** Mechanical / Rotational / Sensors

**Description** The Rotational Speed Sensor measures the angular speed of the flange marked with a dot with respect to the other flange.




---

**Note** Speed and angle sensors are ideally compliant. Hence, if multiple speed or angle sensors are connected in series the speed or angle measured by an individual sensor is undefined. This produces a run-time error.

---

**Parameter** **Second flange**  
Controls whether the second flange is accessible or connected to the rotational reference frame.

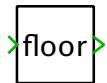
**Probe Signal** **Speed**  
The measured angular speed, in  $\frac{\text{rad}}{\text{s}}$ .

## 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.

**Output data type**

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

**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 to 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:

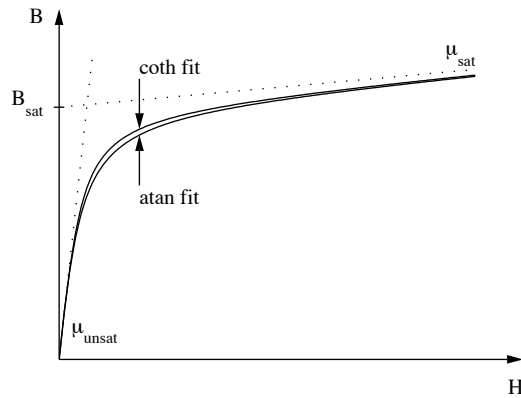
$$B = B_{\text{sat}} \left( \coth \frac{3H}{a} - \frac{a}{3H} \right) + \mu_{\text{sat}} H$$

Both fitting functions have three degrees of freedom which are set by the coefficients  $\mu_{\text{sat}}$ ,  $B_{\text{sat}}$  and  $a$ .  $\mu_{\text{sat}}$  is the fully saturated permeability, which usually corresponds to the magnetic constant  $\mu_0$ , i.e. the permeability of air.  $B_{\text{sat}}$  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  $\mu_{\text{unsat}}$  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  $\mu_{r,unsat} = \mu_{unsat} / \mu_0$  of the core material for  $H \rightarrow 0$ .

**Saturated rel. permeability**

Relative permeability  $\mu_{r,sat} = \mu_{sat} / \mu_0$  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**

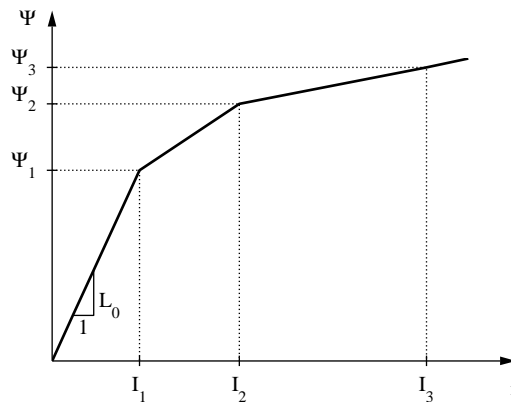
The magnetic flux density  $B$  in the core element, in teslas (T).

## Saturable Inductor

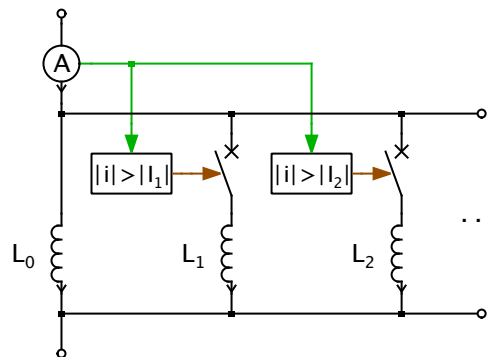
**Purpose** Inductor with piece-wise linear saturation

**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 to 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  $\Psi$  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

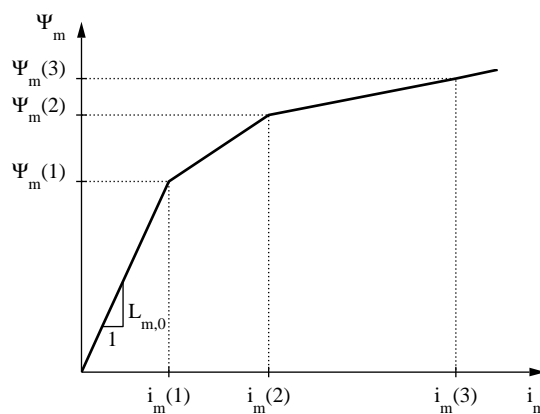
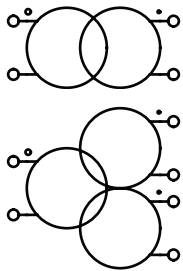
The saturation level indicates which sector of the piece-wise linear characteristic is currently applied. During linear operation, i.e. operation in the first sector, the saturation level is 0. The saturation level is negative for negative flux and current values.

## 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 564). The magnetizing current  $i_m$  and flux  $\Psi_m$  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 ( $\Omega$ ).

**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 ( $\Omega$ ) is referred to the primary side.

**Initial current**

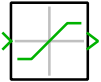
A vector containing the initial currents on the primary side  $i_1$ , the secondary side  $i_2$  and the tertiary side  $i_3$ , if applicable. The currents are given in amperes (A) and considered positive if flowing into the transformer at the marked terminals. The default is [0 0 0].

## 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 `inf` 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 `-inf` the output is unlimited.

**Probe Signals** **Input**  
The block input signal.

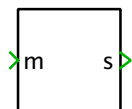
**Output**  
The block output signal.

## Sawtooth PWM

**Purpose** Generate PWM signal using sawtooth carrier

**Library** Control / Modulators

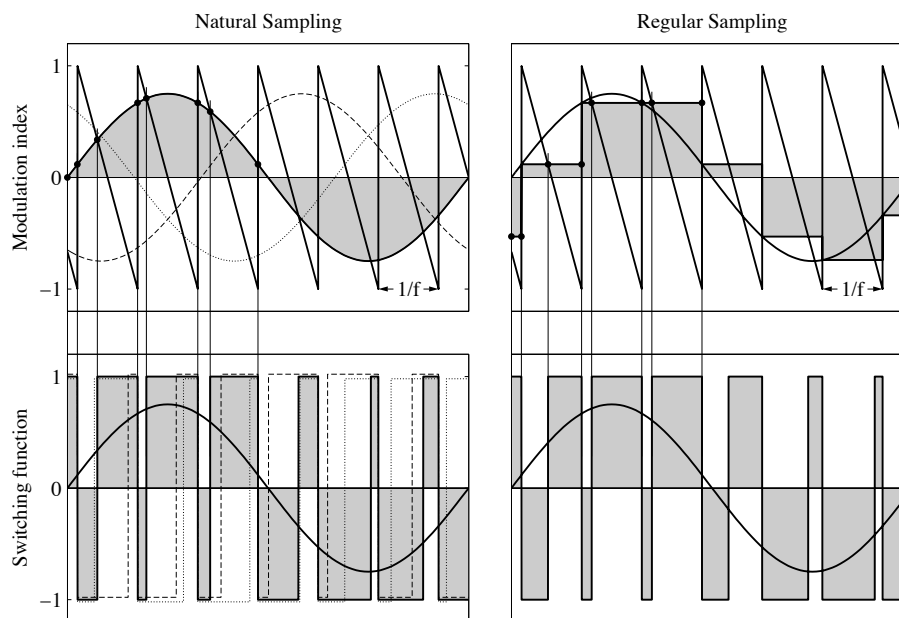
### Description



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 426) or the ideal Converter (see page 411). 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  $[-1 \ 1]$ .

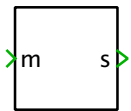


## Sawtooth PWM (3-Level)

**Purpose** Generate 3-level PWM signal using sawtooth carriers

**Library** Control / Modulators

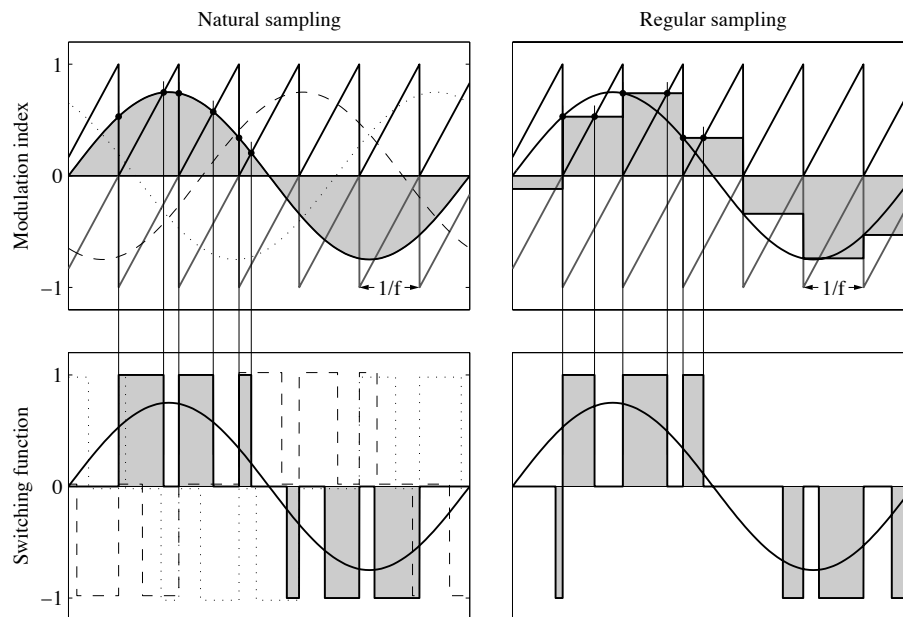
### Description



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 416) or the ideal 3-Level Converter (see page 410). 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

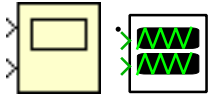
The range of the modulation index. The default is  $[-1 \ 1]$ .

## 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 91).

### Parameters

#### Scope Setup

##### 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.

##### Sample time

The block sample time used to sample the input signals. The default is -1 (inherited). Other valid settings are 0 (continuous) or a valid fixed-step discrete sample time pair (see “Sample Times” on page 36).

##### 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.

#### Time Axis Parameters

##### Display time axis

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

##### Time axis label

The time axis label is shown below the time axis in the scope.

### **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.

### **Scrolling mode**

The scrolling mode determines the way, in which the x-axis is scrolled if during a simulation the current simulation time goes beyond the right x-axis limit.

In the **paged** mode, the plots are cleared when the simulation time reaches the right limit and the x-axis is scrolled by one full x-axis span, i.e. the former right limit becomes the new left limit.

In the **continuous** mode, the plots are continuously scrolled so that new data is always drawn at the right plot border. Note that this mode may affect runtime performance as it causes frequent updates of relatively large screen areas.

## **Individual Plot Parameters**

### **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 limits are automatically chosen based on the minimum and maximum curve value in the visible time range. If you check the **Keep baseline** option, the limits are chosen so that the specified baseline value is always included.

## Set/Reset Switch

**Purpose** Bistable on-off switch

**Library** Electrical / Switches

### Description



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.

### Parameters

#### 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.

#### Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (on page 115).

#### Initial temperature

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

### Probe Signals

#### 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**

tag 

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.

The parameter dialog of the Signal From block provides a link to the corresponding Signal Goto block. Note that the link is not updated until you click the **Apply** button after changing the tag name or scope.

**Parameters**

**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 subsystem 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 Goto blocks with the same tag name within the same scope.

The parameter dialog of the Signal Goto block provides a list of links to the corresponding blocks, i.e. all Signal From blocks with a matching tag name and scope. Note that the list is not updated until you click the **Apply** button after changing the tag name or scope.

### Parameters

#### 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 subsystem in which the block is contained. If the block is not contained in a masked subsystem a global lookup is done.

## Signal Inport

**Purpose** Add signal input connector to subsystem

**Library** System

### Description

1

Inports are used to feed signals from a schematic into a subschematic. In PLECS Blockset, inports 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 *input port number* is assigned to the block. In PLECS Blockset, this port number determines the position of the corresponding input terminal of the PLECS Circuit block in the Simulink model.

For top-level inputs in PLECS Blockset 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 block 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 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 PLECS Blockset if the block is placed in a top-level schematic.

**Output data type**

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261). This parameter appears only if the block is placed in a top-level schematic or in an atomic subsystem (see “Virtual and Atomic Subsystems” on page 603).

## 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 Output

**Purpose** Add signal output connector to subsystem

**Library** System

### Description

1

Outports are used to feed signals from subschematic to the parent schematic. In PLECS Blockset, outports are also used to feed signals from a PLECS circuit back to Simulink. 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.

### Output Blocks in a Top-Level Circuit

If an output block is placed in a top-level circuit a unique *output port number* is assigned to the block. In PLECS Blockset, this port number determines the position of the corresponding output terminal of the PLECS Circuit block in the Simulink model.

### 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.

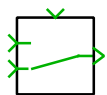
**Output indices**  
A vector with the indices of the input elements that the output vector should contain.

## 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
- $u \sim \text{Threshold}$ .

#### Threshold

The threshold value used for the switch criteria.

### Probe Signals

#### Inputs

The block input signals.

#### Output

The block output signal.

#### Switch position

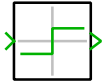
The state of the switch. The output is 0 while the switch is in the off-state and 1 while it is in the on-state.

## 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 according to the equation



$$y = A \cdot \sin(\omega \cdot t + \varphi) + C \quad \omega = 2\pi f$$

If a variable-step solver is used, the solver step size is automatically limited to ensure that a smooth waveform is produced.

### Parameters

#### Amplitude

The amplitude  $A$  of the sine wave signal.

#### Bias

The offset  $C$  that is added to the sine wave signal.

#### Frequency

The frequency of the sine wave in hertz ( $f$ ) or rad/second ( $\omega$ ), see below.

#### Phase

The phase  $\varphi$  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.

#### 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.

### Probe Signal

#### 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 589 and the Small Signal Response block see page 590 to inject a perturbation into a feedback loop and measure the system response. To see the implementation choose **Look under mask** from the **Subsystem** submenu of the block's context menu.

For detailed information regarding small-signal analysis see chapter “Analysis Tools” (on page 161).

### 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 *subtracted* 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.*

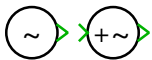
### Purpose

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 161).

### Parameter

#### 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 589.

For detailed information regarding small-signal analysis see chapter “Analysis Tools” (on page 161).

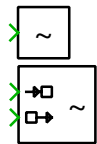
### Parameters

#### 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.

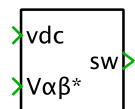


## Space Vector PWM

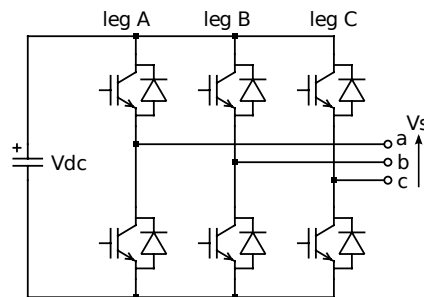
**Purpose** Generate PWM signals for 3-phase inverter using space-vector modulation technique

**Library** Control / Modulators

### Description



The space vector modulator generates a reference voltage vector,  $\vec{V}_s$ , at the terminals of a three phase voltage source converter shown below. The reference vector is defined in the  $\alpha\beta$  coordinate system:  $\vec{V}_s = V_\alpha^* + j V_\beta^*$ .

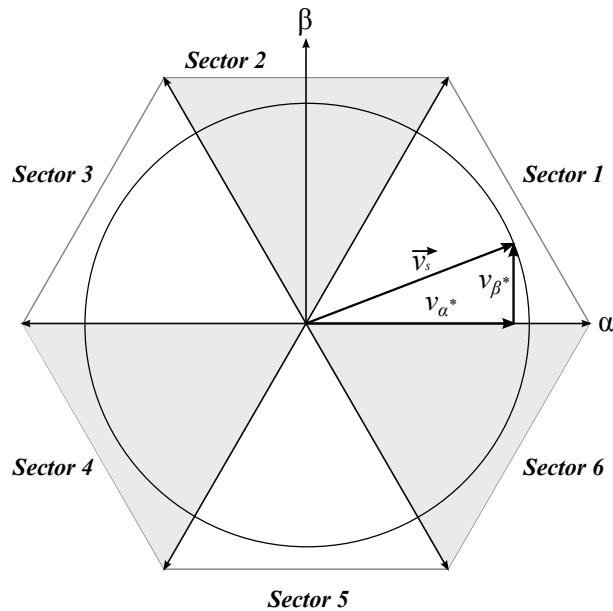


### Operation

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.

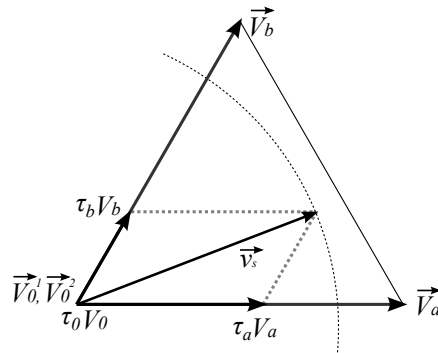
The sector detection calculation determines the sector in which the reference voltage vector  $\vec{V}_s$  resides. The relative on-times,  $\tau_a, \tau_b, \tau_0$ , for the switching vectors  $\vec{V}_a, \vec{V}_b$  and  $\vec{V}_0$  are then calculated. In each sector, two unique switching vectors named  $\vec{V}_a^1$  and  $\vec{V}_b^1$  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  $\vec{V}_a^1$  and  $\vec{V}_b^1$ .

The vector generation and sequencing function creates a switching cycle by time-averaging the switching vectors according to their on-time



### Construction of the reference vector $\vec{v}_s$ .

values. There 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 [www.plexim.com/sites/default/files/plecs\\_svm.pdf](http://www.plexim.com/sites/default/files/plecs_svm.pdf).



**Relationship between relative on times,  $\tau_a, \tau_b, \tau_0$ , 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 inverter 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 inverter 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 inverter’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 inverter.

### Reference voltage

This input, labeled  $V_{\alpha\beta}^*$ , is a two-dimensional vector signal comprising the elements  $[V_\alpha^*, V_\beta^*]$ .

**Switch output**

The output labeled  $sw$  is formed from three switch control signals,  $[S_a, S_b, S_c]$ , which control the inverter 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 inverter legs,  $[S_a, S_b, S_c]$ .

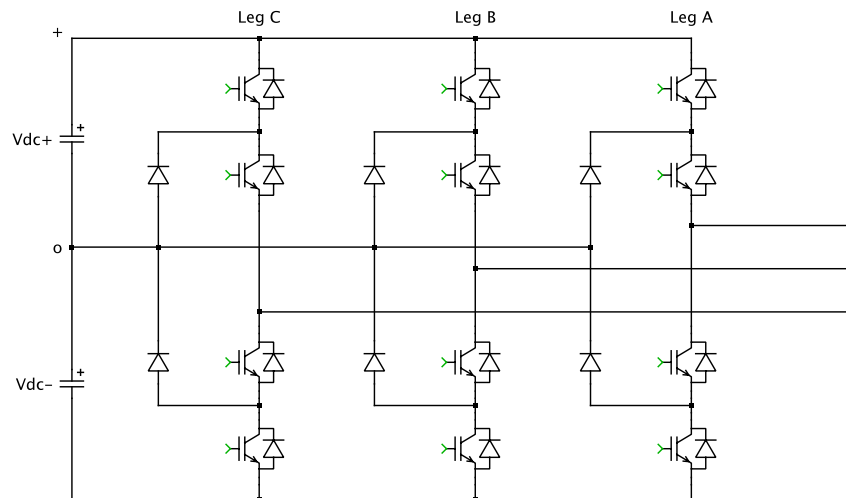
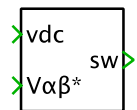


## Space Vector PWM (3-Level)

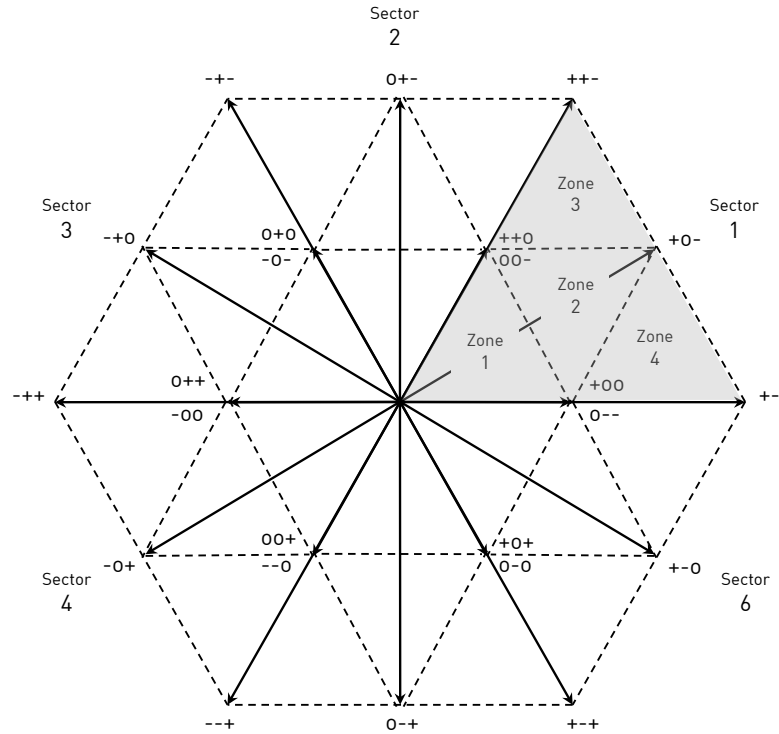
**Purpose** Generate PWM signals for a 3-phase 3-level neutral-point clamped inverter using space-vector modulation technique

**Library** Control / Modulators

**Description** The 3-level space-vector modulator generates a voltage vector on the ac terminals of a neutral-point clamped 3-phase inverter according to a reference signal provided in the stationary  $\alpha\beta$  reference frame.



By controlling the semiconductor gate signals, each ac terminal can be connected either to the high (+), low (-) or neutral (o) point of the dc link. This results in 27 vectors including 12 short vectors, 6 medium vectors and 6 long vectors, as well as 3 zero vectors. Under the assumption of balanced voltages on the capacitors  $V_{dc+}$  and  $V_{dc-}$  the space-vector diagram is graphically depicted below.



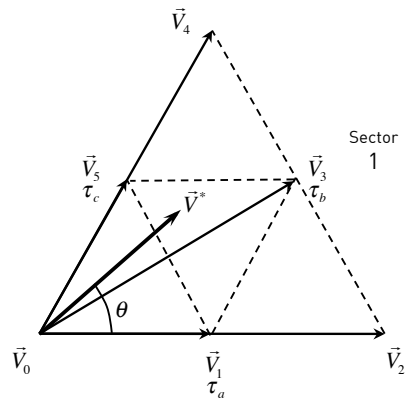
The hexagon area can be divided into six sectors (1 to 6), each of which has four zones (1 to 4). As an example, consider the reference voltage  $\vec{V}^*$  to be located in zone 2 of sector 1. In order to generate the reference voltage  $\vec{V}^*$  on the ac terminals, the adjacent vectors  $\vec{V}_1$ ,  $\vec{V}_3$  and  $\vec{V}_5$  are selected and weighted by time. The on-time of each vector with respect to the switching period is calculated as:

$$\tau_a = 1 - 2k \sin(\theta)$$

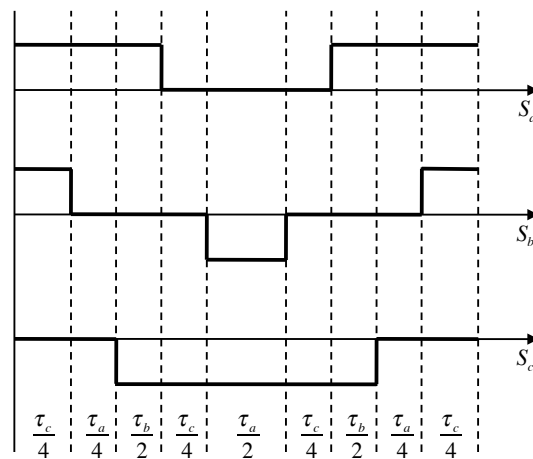
$$\tau_b = 2k \sin\left(\frac{\pi}{3} + \theta\right) - 1$$

$$\tau_c = 1 - 2k \sin\left(\frac{\pi}{3} - \theta\right)$$

This block implements a symmetrical sequence to achieve minimum total harmonic distortion (THD). The short vectors have redundant switch states, e.g.  $\vec{V}_1$  can be either generated by the combination (+00) or (0--). In order to keep the dc link voltages balanced, both switch states must be applied for the same



duration during one switching period. The resulting switch pattern is illustrated below:



### Parameters

#### Switching frequency

The switching frequency in Hz.

#### Output values

The switch output values in the high, neutral and low state. The default values are [-1 0 1].

### Inputs and Outputs

#### DC voltage

The input signal  $V_{dc}$  is the sum of the two dc link voltages  $V_{dc+}$  and  $V_{dc-}$ .

**Reference voltage**

This input, labeled  $V_{\alpha\beta}^*$ , is a two-dimensional vector signal comprising the elements  $[V_{\alpha}^*, V_{\beta}^*]$ .

**Switch output**

The output labeled  $sw$  is formed from the three switch signals  $[S_a, S_b, S_c]$ , which control the converter legs A, B and C. Each switch signal determines if the corresponding ac terminal shall be connected to the positive, neutral or negative side of the dc link.

**Probe Signals****sector**

A value in the set of [1..6] that indicates the sector in which the reference vector,  $\vec{V}^*$ , is located.

**zone**

A value in the set of [1..4] that indicates the zone in which the reference vector,  $\vec{V}^*$ , is located.

**tau**

A vector signal comprising the three relative on-time values,  $[\tau_a, \tau_b, \tau_c]$ .

**sw**

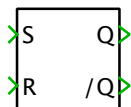
A vector signal consisting of the three gate signals for the inverter legs,  $[S_a, S_b, S_c]$ .

## 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:



<b>S</b>	<b>R</b>	<b>Q</b>	<b>/Q</b>
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.

**Parameter** **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 Machine

**Purpose** Model a state machine

**Library** Control / State Machine

**Description** The State Machine block lets you graphically create and edit state machines and simulate them together with a surrounding system. For a detailed description of State Machines see chapter “State Machines” (on page 215).



## 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.

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

**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:

$$\begin{array}{c} n \\ \left[ \begin{array}{c} A \\ C \end{array} \right] \begin{array}{c} n \\ m \\ m \end{array} \begin{array}{c} \left[ \begin{array}{c} B \\ D \end{array} \right] \\ p \end{array}$$

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.

#### Output data type

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

### Probe Signal

#### 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 physical terminals on the block icon correspond to the input, output, and physical 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.

## Virtual and Atomic Subsystems

By default, PLECS treats subsystems as *virtual*, which means that they only represent a *graphical* grouping of the components that they comprise. At simulation start, virtual subsystems are flattened and the components they comprise are ordered individually when PLECS determines their proper execution order (see “Block Sorting” on page 29).

In an *atomic* subsystem, on the other hand, the components are not only grouped graphically but they are also *executed* as a group. This is necessary if the execution depends on a condition such as a common sample time or an enable and/or a trigger signal (see the Enable block on page 392 and the Trigger block on page 692).

Whether a subsystem is virtual or atomic is controlled by the subsystem execution settings.

---

**Note** A subsystem that has physical terminals cannot be made atomic.

---

## **Execution Settings**

To open the dialog for editing the subsystem settings, select the block, then choose **Execution settings...** from the **Subsystem** submenu of the **Edit** menu or the block's context menu.

### **Treat as atomic unit**

If this parameter is checked, PLECS treats the subsystem as atomic, otherwise as virtual (see “Virtual and Atomic Subsystems” above).

### **Minimize occurrence of algebraic loops**

This parameter only applies to atomic subsystems. If it is *unchecked*, PLECS assumes that all inputs of the subsystem have *direct feedthrough*, i.e. the output functions of the blocks feeding these inputs must be executed before the output function of the subsystem itself can be executed (see “Block Sorting” on page 29). If the atomic subsystem is part of a feedback loop this can result in algebraic loop errors where a virtual subsystem could be used without problems.

If the parameter is *checked*, PLECS determines the actual feedthrough behavior of the individual inputs from the internal connectivity. A subsystem input that is internally only connected to non-direct feedthrough inputs of other blocks (e.g. the inputs of Integrator, Memory or Delay blocks) does *not* have direct feedthrough. This can help reduce the occurrence of algebraic loops.

### **Sample time**

This parameter is enabled only if the subsystem is atomic and specifies the sample time with which the subsystem and the components that it comprises are executed. A setting of auto (which is the default) causes the subsystem to choose its sample time based on the components that it comprises.

The following group of settings is shown only if you have a license for the PLECS Coder.

### **Enable code generation**

Checking this option causes the subsystem to be added to the list of systems in the Coder Options dialog (see “Generating Code” on page 264). Checking this option also implicitly makes the subsystem atomic and disables the minimization of algebraic loops.

### **Discretization step size**

This parameter is enabled only if code generation is enabled. It specifies the base sample time for the generated code and is used to discretize the physical model equations (see “Physical Model Discretization” on page 33) and continuous state variables of control blocks.

**Simulation mode**

This parameter is enabled only if code generation is enabled. When this parameter is set to **Normal**, which is the default, the subsystem is simulated like a normal atomic subsystem. When the parameter is set to **Code-Gen**, the generated code is compiled and linked to PLECS to be executed instead of the subsystem during a simulation.

In connection with the “Traces” feature of the scopes (see “Adding Traces” on page 96), this allows you to easily verify the fidelity of the generated code against a normal simulation.

**Parameters**

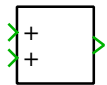
You can create a dialog box for your Subsystem by masking the block (see “Mask Parameters” on page 66 for more details).

## 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.

#### Output data type

*For code generation only.* The data type of the output signal. See “Data Types” (on page 261).

### 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.

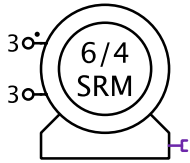
**Probe Signal** **Switch conductivity**  
Conduction state of the switch. The signal outputs 0 if the switch is open, and 1 if it is closed.

## 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.

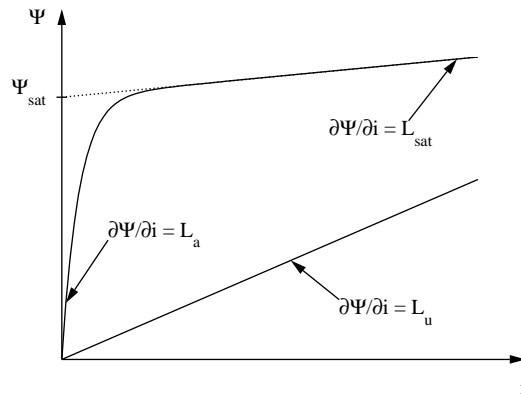
The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. In the component icon, the positive terminals of the stator windings are marked with a dot.

---

**Note** The Switched Reluctance Machine models can only be simulated with the Continuous State-Space Method.

---

The machine flux linkage is modeled as a non-linear 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 non-linear 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

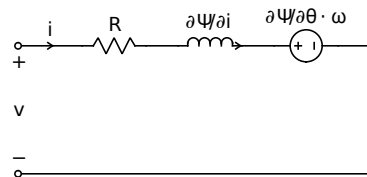
$$\Psi(i, \theta) = \Psi_u(i) + f(\theta) \cdot (\Psi_a(i) - \Psi_u(i))$$

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 ( $\Omega$ ).

#### **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  $\Psi_{sat}$  at which the stator saturates in the aligned position, in Vs.

#### **Inertia**

Combined rotor and load inertia  $J$  in  $\text{Nms}^2$ .

#### **Friction coefficient**

Viscous friction  $F$  in Nms.

#### **Initial rotor speed**

Initial mechanical speed  $\omega_{m,0}$  in radians per second ( $\text{s}^{-1}$ ).

#### **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).



**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 linkage**

The flux linkages in the individual phases of the machine in Vs.

**Rotational speed**

The rotational speed  $\omega_m$  of the rotor in radians per second ( $s^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  in radians.

**Electrical torque**

The electrical torque  $T_e$  of the machine in Nm.

**References**

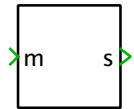
- D.A. Torrey, J.A. Lang, "Modelling a nonlinear variable-reluctance motor drive", IEE Proceedings, Vol. 137, Pt. B, No. 5, Sept. 1990.
- D.A. Torrey, X.-M. Niu, E.J. Unkauf, "Analytical modelling of variable-reluctance machine magnetisation characteristic", IEE Proceedings Electric Power Applications, Vol. 142, No. 1, Jan. 1995.

## Symmetrical PWM

**Purpose** Generate PWM signal using symmetrical triangular carrier

**Library** Control / Modulators

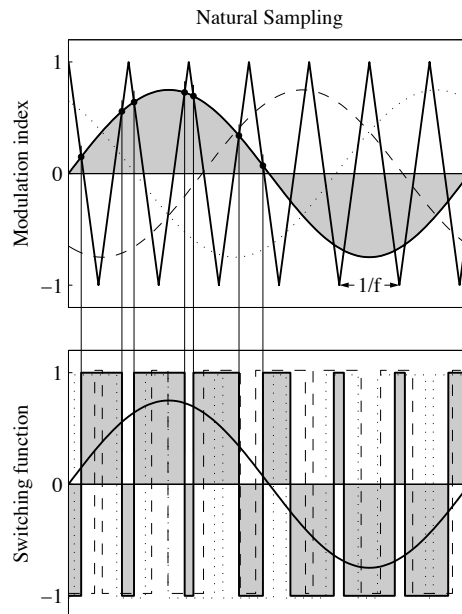
### Description



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.

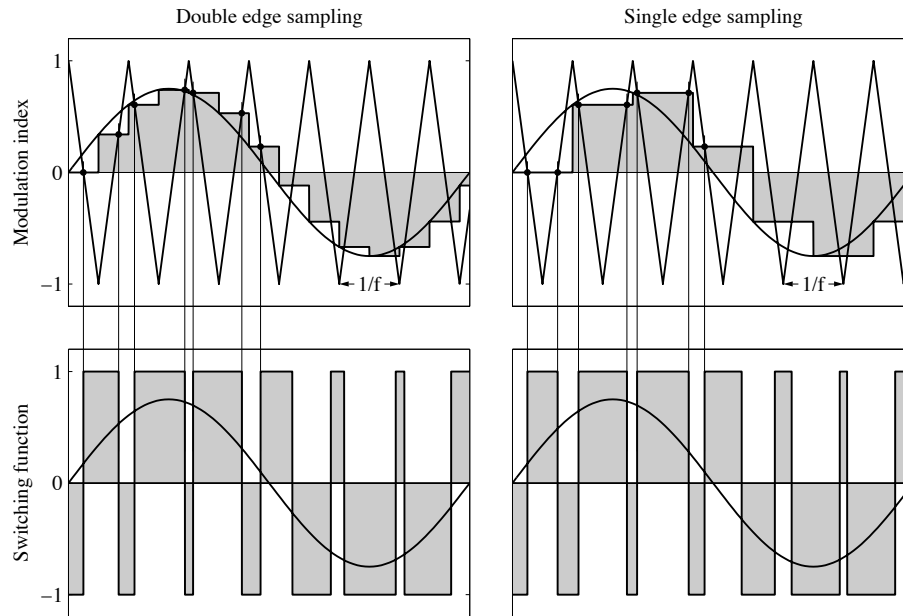
The block can be used to control the IGBT Converter (see page 426) or the ideal Converter (see page 411). In these cases the modulation index must have a width of 3 according to the number of inverter legs.

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

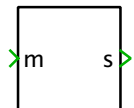
Values of the switching function in off-state and on-state. The default is  $[-1 \ 1]$ .

## Symmetrical PWM (3-Level)

**Purpose** Generate 3-level PWM signal using symmetrical triangular carriers

**Library** Control / Modulators

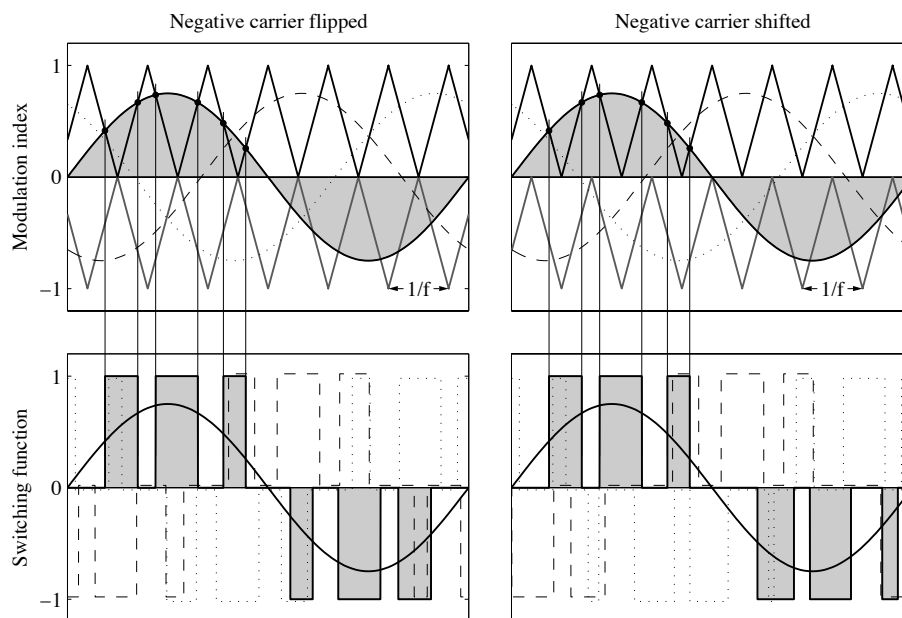
### Description



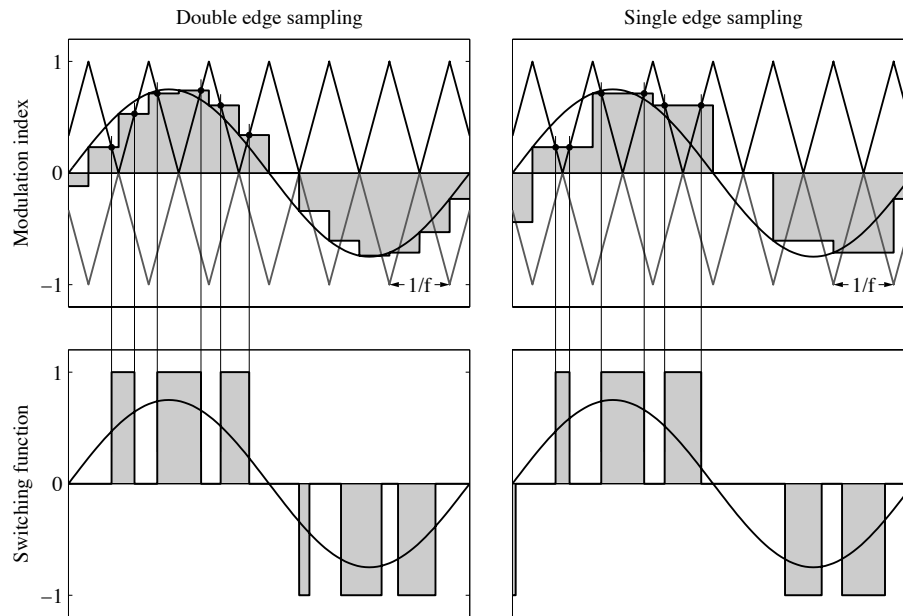
3-level PWM generator with two symmetrical triangular carriers. The input  $m$  is the modulation index. The switching function  $s$  outputs either 1, 0 or -1. If the modulation index is a vector the switching function is also a vector of the same width.

The block can be used to control the 3-Level IGBT Converter (see page 416) or the ideal 3-Level Converter (see page 410). 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

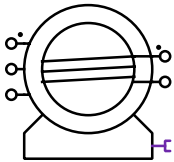
The range of the modulation index. The default is  $[-1 \ 1]$ .

## 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 **Subsystem** submenu of the **Edit** menu. If you want to make changes, you must first choose **Break library link** and then **Unprotect**, both from the same menu.

### Electrical System

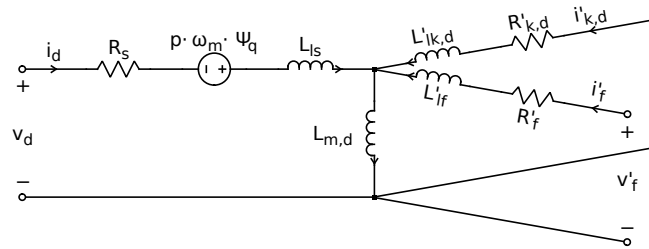
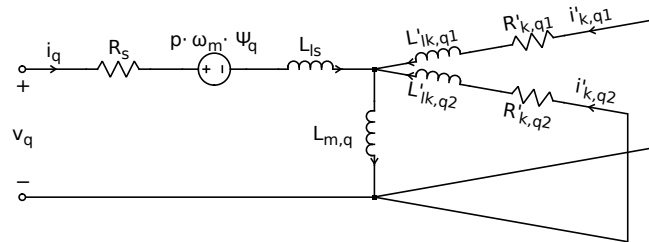
Stator flux linkages:

$$\Psi_d = L_{ls} i_d + L_{m,d} (i_d + i'_f + i'_{k,d})$$

$$\Psi_q = L_{ls} i_q + L_{m,q} (i_q + i'_g + i'_{k,q})$$

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.

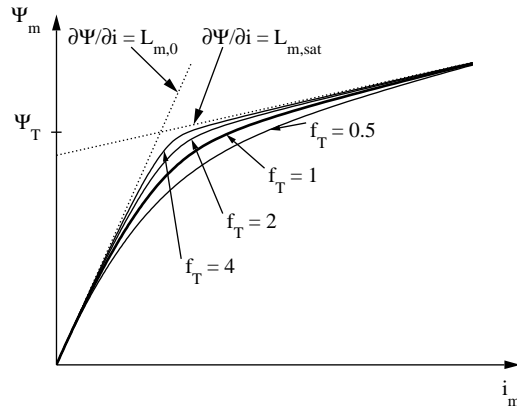
**d-axis****q-axis**

**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  $\Psi_m$  as illustrated in the  $\Psi_m/i_m$  diagram. For flux linkages  $\Psi_m$  far below the transition flux  $\Psi_T$ , 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  $L_{m,sat}$ .  $\Psi_T$  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  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\dot{\theta}_m = \omega_m$$



**Parameters**

Most parameters for the Salient Pole Synchronous Machine (see page 621) 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 ( $\Omega$ ) 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). The current in the damper winding is referred to the stator side.

**Probe Signals**

Most probe signals for the Salient Pole Synchronous Machine (see page 621) 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. Wasynczuk, 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.

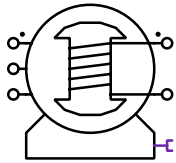
- E. Levi, "Modelling of magnetic saturation in smooth air-gap synchronous machines", IEEE Transactions on Energy Conversion, Vol. 12, No. 2, March 1997.
- E. Levi, "Impact of cross-saturation on accuracy of saturated synchronous machine models", IEEE Transactions on Energy Conversion, Vol. 15, No. 2, June 2000.

# Synchronous Machine (Salient Pole)

**Purpose** Salient pole synchronous machine with main-flux saturation

**Library** Electrical / Machines

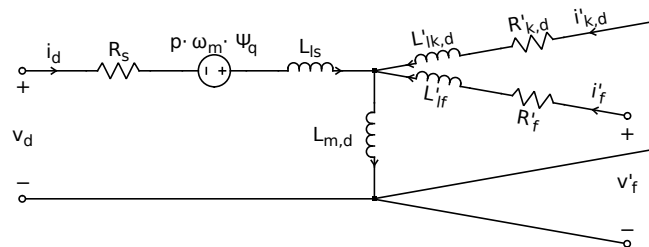
## Description



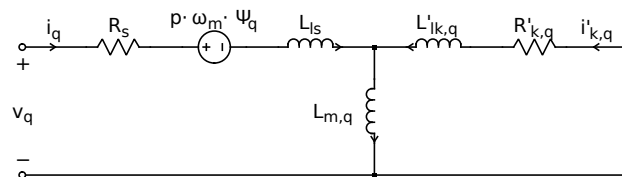
This synchronous machine has one damper winding each on the direct and the quadrature axis of the rotor. Main flux saturation is modeled by means of a continuous function.

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



## d-axis



## q-axis

Stator flux linkages:

$$\Psi_d = L_{ls} i_d + L_{m,d} (i_d + i'_f + i'_{k,d})$$

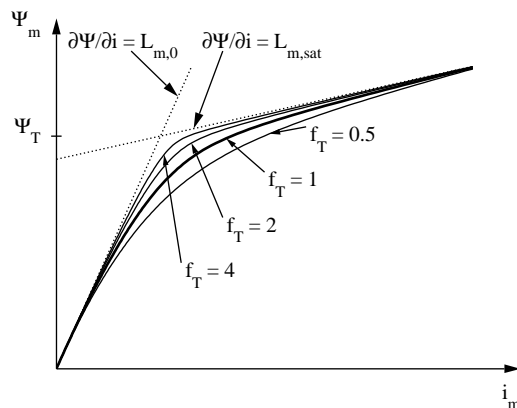
$$\Psi_q = L_{ls} i_q + L_{m,q} (i_q + i'_{k,q})$$

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  $\Psi_m$  as illustrated in the  $\Psi_m/i_m$  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  $\Psi_m$  in an isotropic machine is defined as

$$\Psi_m = \sqrt{\Psi_{m,d}^2 + \Psi_{m,q}^2/m^2}.$$

For flux linkages  $\Psi_m$  far below the transition flux  $\Psi_T$ , 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  $L_{m,\text{sat}}$ .  $\Psi_T$  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  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\dot{\theta}_m = \omega_m$$

**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 ( $\Omega$ ).

**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  $\Psi_T$ , 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 ( $\Omega$ ), 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 ( $\Omega$ ) 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  $\text{Nms}^2$ .

**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).

**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.

**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.

**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  $\Psi_d$  and  $\Psi_q$  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  $\omega_m$  of the rotor in radians per second ( $s^{-1}$ ).

**Rotor position**

The mechanical rotor angle  $\theta_m$  in radians.

**Electrical torque**

The electrical torque  $T_e$  of the machine in Nm.

**References**

- D. C. Aliprantis, O. Wasynczuk, 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.

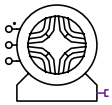


# Synchronous Reluctance Machine

**Purpose** Synchronous reluctance machine configurable with lookup tables

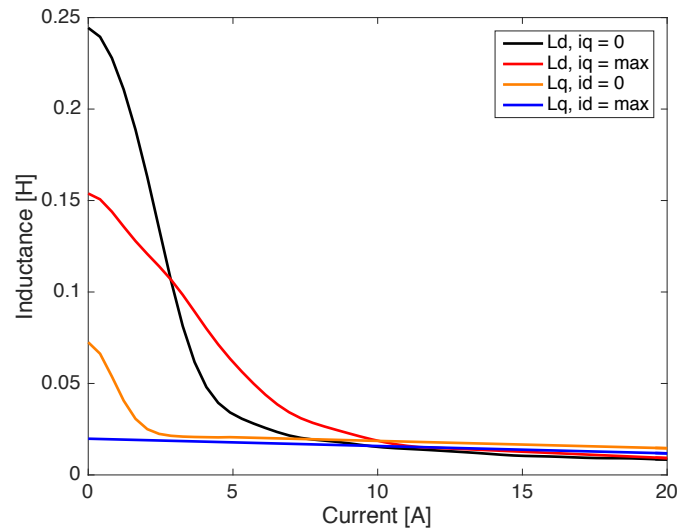
**Library** Electrical / Machines

## Description



This three-phase synchronous reluctance machine has a solid rotor without permanent magnets. Saliency, saturation and cross-coupling are modeled by means of corresponding inductance lookup tables.

Two sets of one-dimensional inductance tables must be provided, one for each axis (d, q), where the first curve corresponds to the case with no cross-saturation, and the second to the case with maximum cross-saturation.



From this information, complete flux linkage and incremental inductance tables are derived, using an interpolation method that ensures a conservative magnetic circuit.

The machine can operate as either a motor or generator. If the mechanical torque has the same sign as the rotational speed, the machine is operating in motor mode; otherwise it is in generator mode. In the component icon, phase a is marked with a dot.

### Electrical System

The model utilizes the Non-Excited Synchronous Machine (see page 507) component. Use this component directly to model permanent magnet-assisted synchronous reluctance machines, or if more complete flux/inductance data is available. The electrical system is realized by means of the Voltage Behind Reactance (VBR) formulation and is therefore appropriate to simulate switching dead-time and failure modes.

### Electro-Mechanical System

Electromagnetic torque:

$$T_e = \frac{3}{2} p (\varphi_d i_q - \varphi_q i_d)$$

### Mechanical System

Mechanical rotor speed  $\omega_m$ :

$$\dot{\omega}_m = \frac{1}{J} (T_e - F\omega_m - T_m)$$

$$\dot{\theta}_m = \omega_m$$

## Parameters

### General

#### Stator resistance

Armature or stator resistance  $R_s$  in  $\Omega$ .

#### Stator leakage inductance

Leakage inductance of stator windings in henries (H). Stator leakage must be set to a non-zero value.

#### Number of pole pairs

Number of pole pairs  $p$ .

#### 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).  $i_{c,0}$  is calculated assuming a neutral connection.

## Magnetizing Inductance

### Current lookup vector

d- and q-axis peak current vector serving as input to inductance lookup vectors. Must be a one dimensional vector with 3 or more elements, and monotonically increasing, i.e.  $[0 \dots i_{d,\max}]$  and  $[0 \dots i_{q,\max}]$ . The values are in amperes (A).

### Ld (iq = 0) lookup vector

d-axis inductance when there is no cross saturation ( $i_q = 0$ ). Must be the same size as the Current lookup vector. The values are in henries (H).

### Ld (iq = max) lookup vector

d-axis inductance when there is maximum cross saturation ( $i_q = i_{q,\max}$ ). If no cross-saturated data is available this can be left empty. Must be the same size as the Current lookup vector. The values are in henries (H).

### Lq (id = 0) lookup vector

q-axis inductance in when there is no cross saturation ( $i_d = 0$ ). Must be the same size as the Current lookup vector. The values are in henries (H).

### Lq (id = max) lookup vector

q-axis inductance when there is maximum cross saturation ( $i_d = i_{d,\max}$ ). If no cross-saturated data is available this can be left empty. Must be the same size as the Current lookup vector. The values are in henries (H).

### Generated table size

User-specified dimension to derive lookup tables for flux linkages and incremental inductances to be used in the underlying Non-Excited Synchronous Machine (see page 507) component.

If left empty, the specified data is used as-is.

Specifying a scalar value, n, will generate equally spaced, n-element d- and q-axis current vectors. The corresponding 2D lookup tables for flux linkage and incremental inductance are also generated. The dimensions of the generated tables must be 3 or more.

The size of the generated tables affect the model initialization and simulation speeds. A smaller size leads to faster model initialization and simulation speeds, but lower resolution in the generated tables. A larger size increases the resolution but adversely affects the model initialization and simulation speeds. Care must be taken when configuring this parameter.

**Current out of range**

Configure to ignore, warn, warn and pause simulation, or generate error and stop simulation if the d- or q-axis currents are outside the specified range.

**Co-energy plausibility check**

The change in co-energy ( $\Delta W$ ) between zero and maximum cross saturation is calculated for both the d-axis ( $\Delta W_d$ ) and q-axis ( $\Delta W_q$ ). Configure to check if  $\Delta W_{d,q}$  are within 5% or 10% of each other to validate the input data. This check can be disabled.

**Mechanical****Inertia**

Combined rotor and load inertia  $J$  in  $\text{Nms}^2$ .

**Friction coefficient**

Viscous friction  $F$  in  $\text{Nms}$ .

**Initial rotor speed**

Initial mechanical rotor speed  $\omega_{m,0}$  in radians per second ( $\text{s}^{-1}$ ).

**Initial rotor position**

Initial mechanical rotor angle  $\theta_{m,0}$  in radians.

**Probe Signals**

All probe signals for the Non-Excited Synchronous Machine (see page 507) are also available with this machine.

**References**

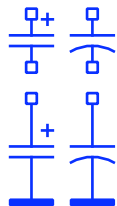
- A. Vagati, M. Pastorelli, F. Scapino, G. Franceschini, "Impact of cross saturation in synchronous reluctance motors of the transverse-laminated type", IEEE Transactions on Industry Applications, Vol. 36, No. 4, Aug 2000.
- A. Vagati, M. Pastorelli, G. Franceschini, "Effect of magnetic cross-coupling in synchronous reluctance motors", Article in PCIM conference proceedings, June 1997.

## 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 41) 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. If left blank or if the value is nan, PLECS will initialize the value based on a thermal “DC” analysis, see “Temperature Initialization” (on page 121).

### Probe Signal

#### Temperature

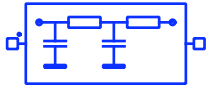
The temperature difference measured across the capacitance. A positive value is measured when the temperature at the terminal marked with “+” is greater than the temperature at the unmarked terminal.

## Thermal Chain

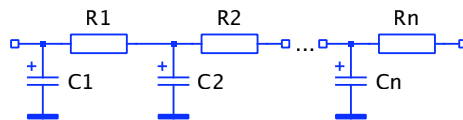
**Purpose** Thermal impedance implemented as RC chain

**Library** Thermal

### Description



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.



### Parameters

#### Thermal resistances

A vector containing the values of the thermal resistors  $R_1 \dots R_n$ , in K/W.

#### Thermal capacitances

A vector containing the values of the thermal capacitors  $C_1 \dots C_n$ , in J/K.

#### Initial temperature

A scalar value specifying the initial temperature of all thermal capacitors at simulation start, in kelvin (K). The default is 0. If left blank or if the value is nan, PLECS will initialize the value based on a thermal “DC” analysis, see “Temperature Initialization” (on page 121).

## Thermal Ground

**Purpose**                    Connect to common reference temperature

**Library**                    Thermal

**Description**              The Thermal Ground implements a connection to the thermal reference.




## Thermal Model Settings

**Purpose** Configure settings for an individual thermal model.

**Library** Thermal

**Description** The Thermal Model Settings block lets you configure parameter settings that influence the code generation for a particular thermal system, see also “Code Generation for Physical Systems” (on page 257).

Model Settings 

The block affects the thermal system that it is attached to by its terminal. At most one Model Settings block may be attached to an individual state-space system. All thermal models within an atomic subsystem are combined into a single common state-space system.

**Parameters** **Matrix coding style**

This setting allows you to specify the format used for storing the state-space matrices for a physical model. When set to `sparse`, only the non-zero matrix entries and their row and column indices are stored. When set to `full`, matrices are stored as full  $m \times n$  arrays. When set to `full (inlined)`, the matrices are additionally embedded in helper functions, which may enable the compiler to further optimize the matrix-vector-multiplications at the cost of increased code size.



## Thermal Port

**Purpose** Add thermal connector to subsystem

**Library** Thermal

### Description



Thermal ports are used to establish thermal connections between a schematic and the subschematic of a subsystem (see page 603). 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.

Terminals can be moved around the edges of the subsystem by holding down the **Shift** key or by using the middle mouse button.

### Thermal Ports in a Top-Level Schematic

In PLECS Blockset, if a Thermal Port is placed in a top-level schematic, the PLECS Circuit block in the Simulink model will show a corresponding thermal terminal, which may be connected with other thermal terminals of the same or a different PLECS Circuit block. The Thermal Port is also assigned a unique *physical port number*. Together with the parameter **Location on circuit block** the port number determines the position of the thermal terminal of the PLECS Circuit block.

For compatibility reasons you can also place an Thermal Port in a top-level schematic in PLECS Standalone. However, since there is no parent system to connect to, such a port will act like an isolated node.

---

**Note** Thermal Port blocks may not be used in schematics that contain Ambient Temperature blocks (see page 316).

---

### Parameter

#### Port number

If a Thermal Port is placed in a top-level schematic in PLECS Blockset, this parameter determines the position, at which the corresponding terminal appears on the PLECS Circuit block.

### **Location on circuit block**

If a Thermal Port is placed in a top-level schematic in PLECS Blockset, this parameter specifies the side of the PLECS Circuit block on which the corresponding terminal appears. By convention, `left` refers to the side on which also input terminals are shown and `right` refers to the side on which also output terminals are shown.

## 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 41) for information on how to change the graphical representation of thermal resistors.

**Parameter**

**Thermal resistance**

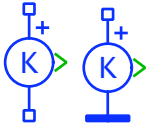
The resistance in K/W. All positive and negative values are accepted, including 0 and  $\text{inf}$  ( $\infty$ ). The default is 1.

## 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 583) or by dragging the component into the dialog box of a Probe block.

**Probe Signal**

**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 when the voltage between anode and cathode is positive and a non-zero gate signal is applied. The switch remains closed until the current becomes negative. 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 ( $\Omega$ ). 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 115). 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

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

## 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**

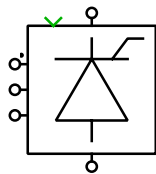
Instantaneous thermal switching losses in joules (J). Only defined if the component is placed on a heat sink.

## Thyristor Rectifier/Inverter

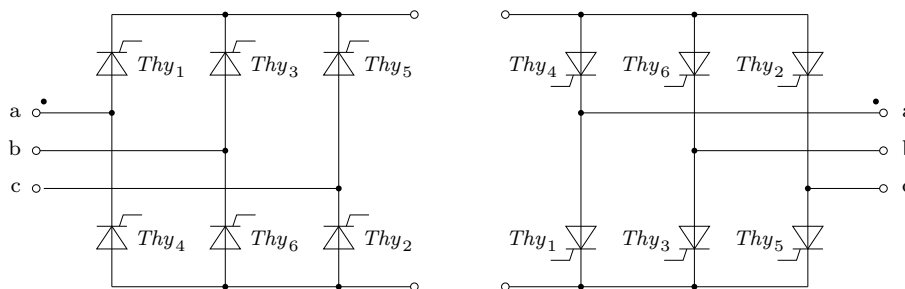
**Purpose** 3-phase thyristor rectifier/inverter

**Library** Electrical / Converters

### Description



Implements a three-phase rectifier or inverter based on the Thyristor model (see page 639). 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 639).

**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



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 363) and described there in detail.

---

### 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 ( $\Omega$ ). The default is 0.

#### Off-resistance

The resistance  $R_{off}$  of the blocking device, in ohms ( $\Omega$ ). The default is 1e6. This parameter may be set to inf unless multiple thyristors are connected in series.

#### 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

To File

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.

If you choose the option **literal**, the string that you enter is taken literally as the basename of the data file. So, if you enter e.g. MyFile, the file name will be MyFile.csv or MyFile.mat.

If you choose the option **evaluate**, the string that you enter is interpreted as a MATLAB/Octave expression that must yield a string, which in turn is taken as the basename of the data file. So, if you enter e.g. ['MyFile' num2str(index)] and the current workspace contains a variable index with a value of 2, the file name will be MyFile2.csv or MyFile2.mat.

#### 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

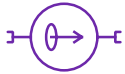
For positive values the input data will be written to the data file once in the given simulation interval. If the value is set to 0 the input data is written to the data file in each simulation step. See also the **Discrete-Periodic** sample time type in section "Sample Times" (on page 36).

## Torque (Constant)

**Purpose** Generate constant torque

**Library** Mechanical / Rotational / Sources

**Description** The Constant Torque generates a constant torque between its two flanges. The direction of a positive torque is indicated by the arrow.




---

**Note** A torque source may not be left unconnected or connected in series with a spring or any other torque source.

---

### Parameters

#### Second flange

Controls whether the second flange is accessible or connected to the rotational reference frame.

#### Torque

The magnitude of the torque, in Newton meters (Nm). The default value is 1.

### Probe Signals

#### Torque

The generated torque, in Newton meters (Nm).

#### Speed

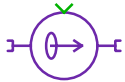
The speed of the flange that the arrow points to with respect to the other flange, in  $\frac{\text{rad}}{\text{s}}$ .

## Torque (Controlled)

**Purpose** Generate variable torque

**Library** Mechanical / Rotational / Sources

**Description**



The Controlled Torque generates a variable torque between its two flanges. The direction of a positive torque is indicated by the arrow. The momentary torque is determined by the signal fed into the input of the component.

---

**Note** A torque source may not be left unconnected or connected in series with a spring or any other torque source.

---

**Parameters**

**Second flange**

Controls whether the second flange is accessible or connected to the rotational reference frame.

**Allow state-space inlining**

**For expert use only!** When set to on and the input signal is a linear combination of mechanical measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

**Torque**

The generated torque, in Newton meters (Nm).

**Speed**

The speed of the flange that the arrow points to with respect to the other flange, in  $\frac{\text{rad}}{\text{s}}$ .

## Torque Sensor

**Purpose** Output measured torque as signal

**Library** Mechanical / Rotational / Sensors

### Description



The Torque Sensor measures the torque between its two flanges and provides it as a signal at the output of the component. A torque flow from the unmarked flange towards the flange marked with a dot is considered positive.

---

**Note** A torque sensor is ideally rigid. Hence, if multiple torque sensors are connected in parallel the torque measured by an individual sensor is undefined. This produces a run-time error.

---

### Parameter

#### Second flange

Controls whether the second flange is accessible or connected to the rotational reference frame.

### Probe Signal

#### Torque

The measured torque, in Newton meters (Nm).

## Torsion Spring

**Purpose** Ideal torsion spring

**Library** Mechanical / Rotational / Components

**Description** The Torsion Spring models an ideal linear spring in a rotational system described with the following equations:



$$\begin{aligned}\tau &= -c \cdot \Delta\theta \\ \Delta\theta &= \theta - \theta_0 \\ \frac{d}{dt}\theta &= \omega\end{aligned}$$

where  $\tau$  is the torque flow from the unmarked towards the marked flange,  $\theta$  is the angle of the marked flange with respect to the unmarked one, and  $\theta_0$  is the equilibrium flange displacement.

---

**Note** An torsion spring may not be connected in series with a torque source. Doing so would create a dependency between an input variable (the source torque) and a state variable (the spring torque) in the underlying state-space equations.

---

### Parameters

#### Spring constant

The spring rate or stiffness  $c$ , in  $\frac{\text{Nm}}{\text{rad}}$ .

#### Equilibrium (unstretched) displacement

The displacement  $\theta_0$  between the two flanges of the unloaded spring, in radians.

#### Initial deformation

The initial deformation (torsion)  $\Delta\theta_0$  of the spring, in radians.

### Probe Signals

#### Torque

The spring torque  $\tau$ , in Newton meters (Nm).

#### Deformation

The spring deformation  $\Delta\theta$ , in radians.

## Transfer Function

**Purpose** Model linear time-invariant system as transfer function

**Library** Control / Continuous

### Description

$$\frac{1}{s+1}$$

The Transfer Function models a linear time-invariant system that is expressed in the Laplace domain in terms of the argument  $s$ :

$$\frac{Y(s)}{U(s)} = \frac{n_n s^n + \dots + n_1 s + n_0}{d_n s^n + \dots + d_1 s + d_0}$$

The transfer function is displayed in the block if it is large enough, otherwise a default text is shown. To resize the block, select it, then drag one of its selection handles.

### 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  $s$ .

---

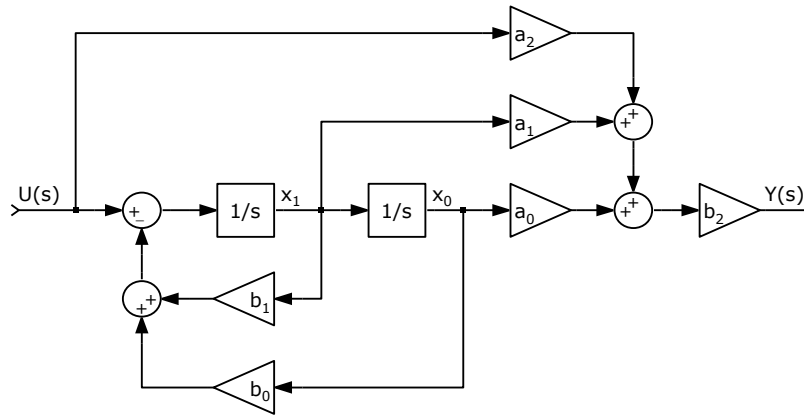
**Note** The order of the denominator (highest power of  $s$ ) must be greater than or equal to the order of the numerator.

---

#### 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 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.

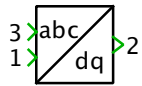


## Transformation 3ph->RRF

**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  $\varphi$  of the rotating reference frame.  $\varphi$  is given in radians.

$$\begin{bmatrix} y_d \\ y_q \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \cos(\varphi - 120^\circ) & -\sin(\varphi - 120^\circ) \\ \cos(\varphi + 120^\circ) & -\sin(\varphi + 120^\circ) \end{bmatrix}^T \cdot \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

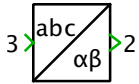
Any zero-sequence component in the three-phase signals is discarded.

## 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:



$$\begin{bmatrix} y_\alpha \\ y_\beta \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

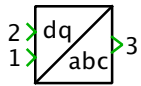
Any zero-sequence component in the three-phase signals is discarded.

## 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  $\varphi$  of the rotating reference frame.  $\varphi$  is given in radians.

$$\begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix} = \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \cos(\varphi - 120^\circ) & -\sin(\varphi - 120^\circ) \\ \cos(\varphi + 120^\circ) & -\sin(\varphi + 120^\circ) \end{bmatrix} \cdot \begin{bmatrix} x_d \\ x_q \end{bmatrix}$$

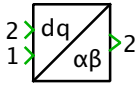
The resulting three-phase signal does not have any zero-sequence component.

## 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  $\varphi$  between the rotating and the stationary frame.  $\varphi$  is given in radians.



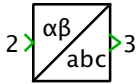
$$\begin{bmatrix} y_\alpha \\ y_\beta \end{bmatrix} = \begin{bmatrix} \cos \omega_1 t & -\sin \omega_1 t \\ \sin \omega_1 t & \cos \omega_1 t \end{bmatrix} \cdot \begin{bmatrix} x_d \\ x_q \end{bmatrix}$$

## 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  $[y_a \ y_b \ y_c]$ .



$$\begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}$$

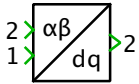
The resulting three-phase signal does not have any zero-sequence component.

## 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  $\varphi$  between the rotating and the stationary frame.  $\varphi$  is given in radians.



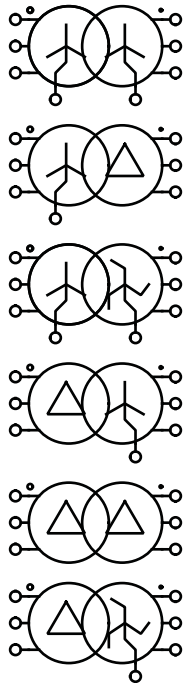
$$\begin{bmatrix} y_d \\ y_q \end{bmatrix} = \begin{bmatrix} \cos \omega_1 t & \sin \omega_1 t \\ -\sin \omega_1 t & \cos \omega_1 t \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}$$

## Transformers (3ph, 2 Windings)

**Purpose** 3-phase transformers in Yy, Yd, Yz, Dy, Dd and Dz connection

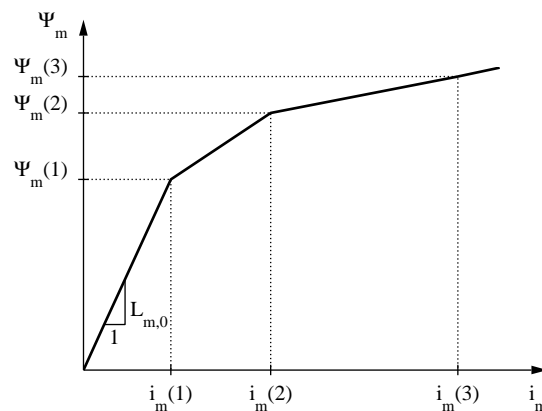
**Library** Electrical / Transformers

### 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  $60^\circ$ . For Yd and Dy connections the phase lag must be an odd integer multiple of  $30^\circ$ . 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 564). The magnetizing current  $i_m$  and flux  $\Psi_m$  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 ( $\Omega$ ).

### 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 ( $\Omega$ ) 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  $60^\circ$ .

### 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**

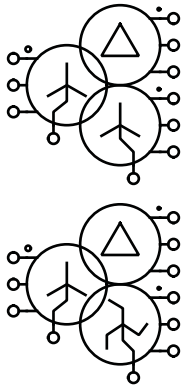
A vector containing the initial currents on the secondary side  $i_{2,a}$ ,  $i_{2,b}$  and, if the winding has a neutral point,  $i_{2,c}$ . The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is [0 0 0].

## Transformers (3ph, 3 Windings)

**Purpose** Three-phase transformers in Ydy and Ydz connection.

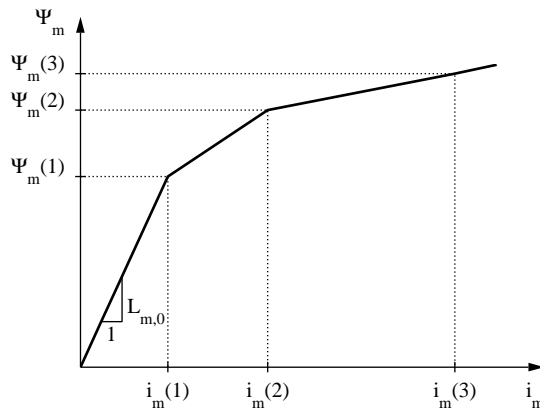
**Library** Electrical / Transformers

**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  $30^\circ$ . If the tertiary winding is in star connection the phase lag against the primary side must be an integer multiple of  $60^\circ$ . 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 564). The magnetizing current  $i_m$  and flux  $\Psi_m$  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 ( $\Omega$ ).

### 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 ( $\Omega$ ) 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  $60^\circ$ .

### 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].

# Translational Algebraic Component

**Purpose** Define an algebraic constraint in terms of force and speed

**Library** Mechanical / Translational / Components

## Description



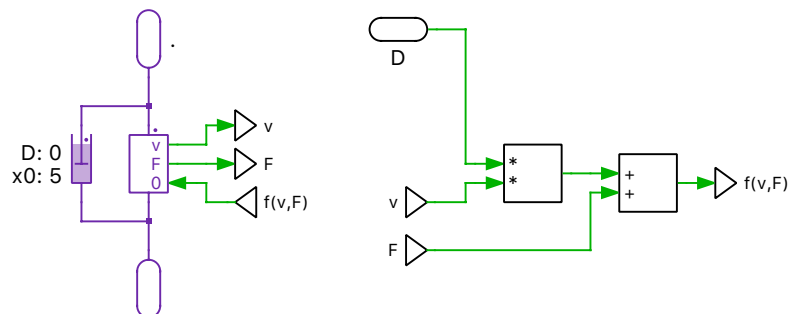
The Translational Algebraic Component enforces an arbitrary algebraic constraint involving force and speed.

The output signal “ $v$ ” measures the speed of the marked flange with respect to the unmarked one. The output signal “ $F$ ” measures the force flow from the unmarked towards the marked flange. The two output signals must affect the input signal “ $0$ ” by means of a direct feedthrough path. The component ensures that the input signal is zero at all times.

The direct feedthrough path defines a function  $f(v, F)$ , which in turn implicitly determines the speed-force characteristic of the component through the constraint  $f(v, F) = 0$ . For instance, the choice  $f(v, F) := F + D \cdot v$  causes the Translational Algebraic Component to act as a Translational Damper (see page 667) with damping constant  $D$ .

The Translational Algebraic Component offers no direct way to specify an initial displacement. In case you need to do so, place a Translational Damper with zero damping constant in parallel to the component and set the initial displacement property thereof.

By way of illustration, the following schematic shows a possible implementation of a translational damper with variable damping constant and prescribed initial displacement:



---

**Note** The Translational Algebraic Component creates an algebraic loop. See section “Block Sorting” (on page 29) for more information on algebraic loops.

---

## **Probe Signals**

### **Component force**

The force flow from the unmarked towards the marked flange.


### **Component speed**

The speed of the marked flange with respect to the unmarked one.

### **Component power**

The power consumed by the component.

## Translational Backlash

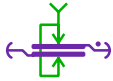
<b>Purpose</b>	Ideal translational backlash
<b>Library</b>	Mechanical / Translational / Components
<b>Description</b>	 <p>The Translational Backlash models an ideal symmetrical, two-sided Hard Stop (see page 670) in a translational system, which restricts the relative displacement of the two flanges between an upper and lower limit of <math>\pm \frac{b}{2}</math>. While the displacement is within the limits, no force is transmitted. When the displacement hits either limit, the flanges become rigidly connected until the transmitted force reverses.</p>
<b>Parameters</b>	<p><b>Total backlash</b> The total permitted displacement <math>b</math> between the flanges, in meters.</p> <p><b>Initial displacement</b> The initial displacement of the flanges, in meters. May be specified in order to provide proper initial conditions if absolute positions are measured anywhere in the system. Otherwise, this parameter can be left blank.</p>
<b>Probe Signals</b>	<p><b>Force</b> The transmitted force flowing from the unmarked to the marked flange, in Newton (N).</p> <p><b>Displacement</b> The displacement of the marked flange with respect to the unmarked flange, in meters (m).</p> <p><b>State</b> The internal state of the component: -1 in lower limit, 0 inside limits, +1 in upper limit.</p>

## Translational Clutch

**Purpose** Ideal translational clutch

**Library** Mechanical / Translational / Components

**Description**



The Translational Clutch models an ideal clutch in a translational system. When engaged, it makes an ideally rigid connection between the flanges; when disengaged, it transmits zero force. The clutch engages when the input signal becomes non-zero and disengages when the input signal becomes zero.

**Parameters**

**Initial state**

The initial state (engaged/disengaged) of the clutch.

**Initial displacement**

The initial displacement of the flanges, in meters. May be specified in order to provide proper initial conditions if absolute positions are measured anywhere in the system. Otherwise, this parameter can be left blank.



## Translational Damper

**Purpose** Ideal viscous translational damper

**Library** Mechanical / Translational / Components

**Description** The Translational Damper models an ideal linear damper in a translational system described with the following equations:



$$F = -D \cdot v$$

where  $F$  is the force flow from the unmarked towards the marked flange and  $v$  is the speed of the marked flange with respect to the unmarked one.

**Parameters**

**Damper constant**

The damping (viscous friction) constant  $D$ , in  $\frac{\text{Ns}}{\text{m}}$ .

**Initial displacement**

The initial displacement of the flanges, in meters. May be specified in order to provide proper initial conditions if absolute positions are measured anywhere in the system. Otherwise, this parameter can be left blank.

## Translational Friction

**Purpose** Ideal translational stick/slip friction

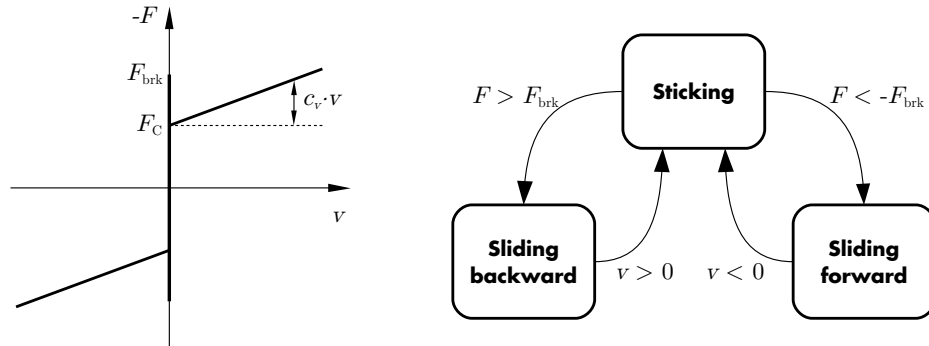
**Library** Mechanical / Translational / Components

**Description**



The Translational Friction models any combination of static, Coulomb and viscous friction between two flanges in a translational system. While the component is stuck, it exerts whatever force is necessary in order to maintain zero relative speed between the flanges, up to the limit of the breakaway force  $F_{brk}$ . When the breakaway force is exceeded, the flanges begin sliding against each other, and the component exerts a force that consists of the Coulomb friction force  $F_C$  and a speed-dependent viscous friction force  $c_v \cdot v$ .

The figure below shows the speed/force characteristic and the state chart of the component. Note that the friction force is opposed to the movement, hence the negative sign.



**Parameters**

**Breakaway friction force**

The maximum magnitude of the stiction force  $F_{brk}$ , in Newton. Must be greater than or equal to zero.

**Coulomb friction force**

The magnitude of the (constant) Coulomb friction force  $F_C$ , in Newton. Must be greater than or equal to zero and less than or equal to the breakaway friction force.

**Viscous friction coefficient**

The proportionality coefficient  $c_v$  that determines the speed dependent viscous friction force, in  $\frac{Ns}{m}$ .

**Probe Signals****Force**

The transmitted force  $F$  flowing from the unmarked to the marked flange, in Newton (N).

**Speed**

The speed  $v$  of the marked flange with respect to the unmarked flange, in  $\frac{\text{m}}{\text{s}}$ .

**State**

The internal state of the component: -1 sliding backward, 0 stuck, +1 sliding forward.

## Translational Hard Stop

**Purpose** Ideal translational hard stop

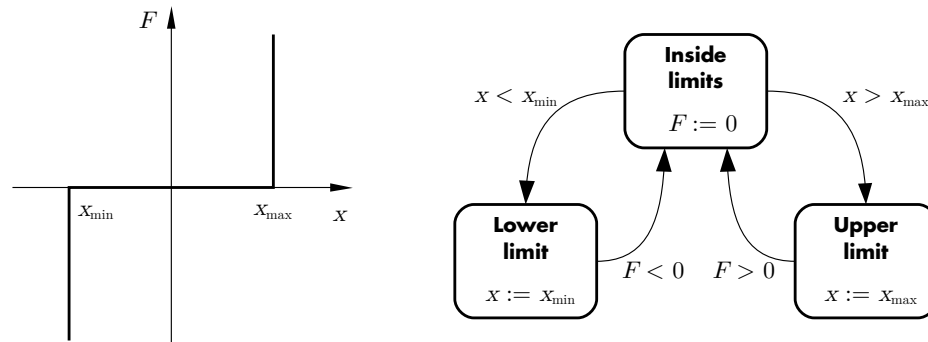
**Library** Mechanical / Translational / Components

**Description**



The Translational Hard Stop models an ideal one- or two-sided hard stop in a translational system, which restricts the relative displacement of the two flanges between an upper and lower limit. While the displacement is within the limits, no force is transmitted. When the displacement hits either limit, the displacement is clamped at the limit and the flanges become rigidly connected until the transmitted force reverses.

The figure below shows the displacement/force characteristic and the state chart of the component.



**Parameters**

**Upper limit**

The maximum displacement  $x_{max}$  between the flanges, in meters. Set to  $inf$  to disable this limit.

**Lower limit**

The minimum displacement  $x_{min}$  between the flanges, in meters. Set to  $-inf$  to disable this limit.

**Initial displacement**

The initial displacement of the flanges, in meters.

**Probe Signals**

**Force**

The transmitted force  $F$  flowing from the unmarked to the marked flange, in Newton (N).

**Displacement**

The displacement  $x$  of the marked flange with respect to the unmarked flange, in meters (m).

**State**

The internal state of the component: -1 in lower limit, 0 inside limits, +1 in upper limit.

## Translational Model Settings

**Purpose** Configure settings for an individual mechanical model.

**Library** Mechanical / Translational / Model Settings

### Description

Model Settings →

The Translational Model Settings block lets you configure parameter settings that influence the code generation for a particular mechanical system, see also “Code Generation for Physical Systems” (on page 257).

The block affects the mechanical system that it is attached to by its translational terminal. At most one Model Settings block may be attached to an individual state-space system. A mechanical model can be split into multiple state-space systems if the underlying model equations are fully decoupled. Note that a rotational system and a translational system that are coupled e.g. with a Rack and Pinion (see page 535) have coupled model equations. See also the options **Enable state-space splitting** and **Display state-space splitting** in the “Simulation Parameters” (on page 103).

### Parameters

#### Switching algorithm

This parameter allows you to choose between two algorithms to determine the clutch states in the generated code. See “Switching Algorithm” (on page 259) for details.

#### Matrix coding style

This setting allows you to specify the format used for storing the state-space matrices for a physical model. When set to `sparse`, only the non-zero matrix entries and their row and column indices are stored. When set to `full`, matrices are stored as full  $m \times n$  arrays. When set to `full (inlined)`, the matrices are additionally embedded in helper functions, which may enable the compiler to further optimize the matrix-vector-multiplications at the cost of increased code size.

#### Topologies

This parameter lets you specify a matrix containing the combinations of clutch states, for which code should be generated, see also “Reducing the Code Size” (on page 258). The columns represent the clutch elements of the physical model and each row represents one combination of clutch states (zero for an open clutch and non-zero for a closed clutch).

The default is an empty matrix `[]`, which means that *all* possible combinations are included.

## Translational Port

**Purpose** Add translational flange to subsystem

**Library** Mechanical / Translational / Components

### Description



Translational ports are used to establish translational mechanical connections between a schematic and the subschematic of a subsystem (see page 603). If you copy a Translational 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.

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.

### Translational Ports in a Top-Level Schematic

In PLECS Blockset, if a Translational Port is placed in a top-level schematic, the PLECS Circuit block in the Simulink model will show a corresponding translational terminal, which may be connected with other translational terminals of the same or a different PLECS Circuit block. The Translational Port is also assigned a unique *physical port number*. Together with the parameter **Location on circuit block** the port number determines the position of the translational terminal of the PLECS Circuit block.

For compatibility reasons you can also place an Translational Port in a top-level schematic in PLECS Standalone. However, since there is no parent system to connect to, such a port will act like an isolated node.

### Parameter

#### Port number

If a Translational Port is placed in a top-level schematic in PLECS Blockset, this parameter determines the position, at which the corresponding terminal appears on the PLECS Circuit block.

#### Location on circuit block

If a Translational Port is placed in a top-level schematic in PLECS Blockset, this parameter specifies the side of the PLECS Circuit block on which the corresponding terminal appears. By convention, `left` refers to the side on which also input terminals are shown, and `right` refers to the side on which also output terminals are shown.

## Translational Reference

**Purpose** Connect to common translational reference frame

**Library** Mechanical / Translational / Components

**Description** The Translational Reference implements a connection to the translational reference frame that has a fixed absolute position of zero.





## Translational Speed (Constant)

**Purpose** Maintain constant translational speed

**Library** Mechanical / Translational / Sources

**Description** The Constant Translational Speed maintains a constant linear speed between its two flanges regardless of the force required. The speed is considered positive at the flange marked with a “+”.




---

**Note** A speed source may not be short-circuited or connected in parallel with other speed sources, nor may a speed source directly drive a mass.

---

### Parameters

#### Second flange

Controls whether the second flange is accessible or connected to the translational reference frame.

#### Speed

The magnitude of the speed, in  $\frac{m}{s}$ . The default value is 1.

### Probe Signals

#### Force

The generated force flowing from the unmarked flange to the marked flange, in Newton (N).

#### Speed

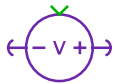
The speed, in  $\frac{m}{s}$ .

## Translational Speed (Controlled)

**Purpose** Maintain variable translational speed

**Library** Mechanical / Translational / Sources

### Description



The Controlled Translational Speed maintains a variable linear speed between its two flanges regardless of the force required. The speed is considered positive at the flange marked with a “+”. The momentary speed is determined by the signal fed into the input of the component.

---

**Note** A speed source may not be short-circuited or connected in parallel with other speed sources, nor may a speed source directly drive a mass.

---

### Parameters

#### Second flange

Controls whether the second flange is accessible or connected to the translational reference frame.

#### Allow state-space inlining

**For expert use only!** When set to on and the input signal is a linear combination of mechanical measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

### Probe Signals

#### Force

The generated force flowing from the unmarked flange to the marked flange, in Newton (N).

#### Speed

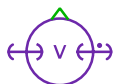
The speed, in  $\frac{m}{s}$ .

## Translational Speed Sensor

**Purpose** Output measured linear speed as signal

**Library** Mechanical / Translational / Sensors

**Description** The Translational Speed Sensor measures the linear speed of the flange marked with a dot with respect to the other flange.



---

**Note** Speed and position sensors are ideally compliant. Hence, if multiple speed or position sensors are connected in series the speed or position measured by an individual sensor is undefined. This produces a run-time error.

---

**Parameter** **Second flange**  
Controls whether the second flange is accessible or connected to the translational reference frame.

**Probe Signal** **Speed**  
The measured speed, in  $\frac{m}{s}$ .

## Translational Spring

**Purpose** Ideal translational spring

**Library** Mechanical / Translational / Components

**Description** The Translational Spring models an ideal linear spring in a translational system described with the following equations:



$$F = -c \cdot \Delta x$$

$$\Delta x = x - x_0$$

$$\frac{d}{dt}x = v$$

where  $F$  is the force flow from the unmarked towards the marked flange,  $x$  is the displacement of the marked flange with respect to the unmarked one, and  $x_0$  is the equilibrium displacement.

---

**Note** An translational spring may not be connected in series with a force source. Doing so would create a dependency between an input variable (the source force) and a state variable (the spring force) in the underlying state-space equations.

---

### Parameters

#### Spring constant

The spring rate or stiffness  $c$ , in  $\frac{\text{N}}{\text{m}}$ .

#### Equilibrium (unstretched) displacement

The displacement  $x_0$  between the two flanges of the unloaded spring, in meters.

#### Initial deformation

The initial deformation  $\Delta x_0$  of the spring, in meters.

### Probe Signals

#### Force

The spring force  $F$ , in Newton (N).

#### Deformation

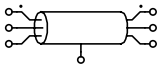
The spring deformation  $\Delta x$ , in meters (m).

## Transmission Line (3ph)

**Purpose** 3-phase transmission line

**Library** Electrical / Passive Components

### Description

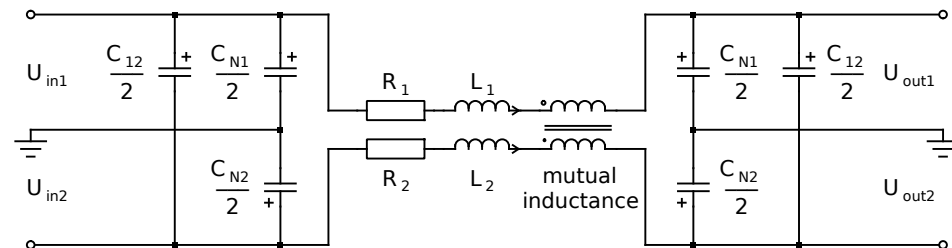


This component implements a three-phase transmission line. A transmission line is characterized by a uniform distribution of inductances, resistances and neutral capacitances along the line. In multi-wire lines, there are also uniformly distributed mutual inductances and coupling capacitances.

The user has the choice between two different implementations: one with series-connected pi sections of lumped elements and another one with distributed parameters based on traveling wave theory. A stiff solver is recommended for simulating models containing this component.

### Pi-Section Line

In many cases, the uniformly distributed parameters of a transmission line can be approximated by a series of pi sections consisting of lumped inductors, capacitors and resistors. The figure below illustrates a single pi section exemplified for a 2-phase line. Depending on the desired fidelity at higher frequencies, the number of series-connected pi sections can be configured.



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 neutral capacitance  $C_N$  as well as the coupling capacitances  $C_{ij}$  and mutual inductances  $L_{ij}$  of the discrete elements can then be calculated from their per-unit-length counterparts  $L'$ ,  $R'$ ,  $C'_N$ ,  $C'_{ij}$  and  $L'_{ij}$  using the following equations:

$$L = \frac{l}{n} L', \quad R = \frac{l}{n} R', \quad C_N = \frac{l}{n} C'_N$$

$$C_{ij} = \frac{l}{n} C'_{ij}, \quad L_{ij} = \frac{l}{n} L'_{ij}$$

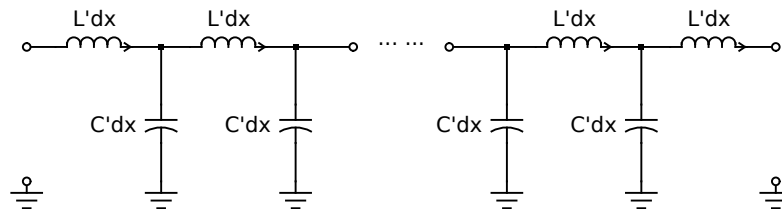
It is possible to specify the parameters for each phase individually in order to model asymmetric lines. In this case, the parameters must be provided in vector format. Otherwise, the parameter can be a scalar assigning the same value to all phases.

### Distributed Parameter Line

The implementation of a distributed parameter line is based on the traveling wave theory, which describes the time delay phenomenon. This approach is numerically more efficient due to the absence of numerous state variables and should be used in large models.

Modeling asymmetric lines is not supported, therefore all parameters need to be scalar.

### Single-Phase Lossless Line



Consider a lossless transmission line with inductance  $L'$  and capacitance  $C'$  per unit length. At a certain point  $x$  along the total length  $d$ , the relation between the line voltage and current can be described with partial differential equations:

$$-\frac{\partial e}{\partial x} = L' \cdot \frac{\partial i}{\partial t}$$

$$-\frac{\partial i}{\partial x} = C' \cdot \frac{\partial e}{\partial t}$$

Since a wave entering the sending end “s” of the line must remain unchanged when it arrives at the receiving end “r” (and vice versa), the following expression is derived:

$$i_s = \frac{1}{Z} \cdot e_s(t) - I_{sh}$$

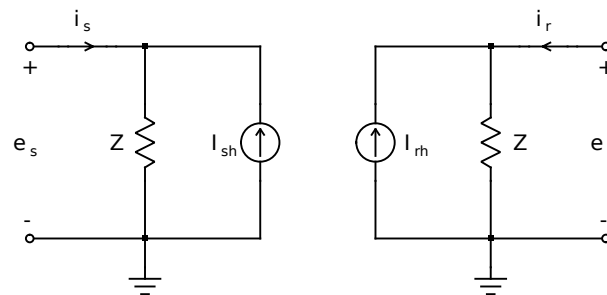
$$i_r = \frac{1}{Z} \cdot e_r(t) - I_{rh}$$

where

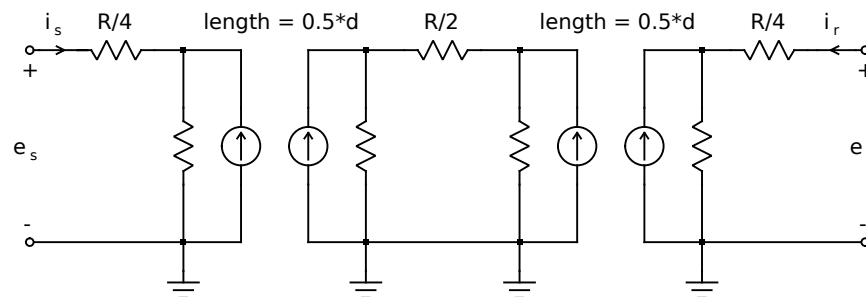
$$I_{sh} = \frac{1}{Z} \cdot e_r(t - \tau) + i_r(t - \tau)$$

$$I_{rh} = \frac{1}{Z} \cdot e_s(t - \tau) + i_s(t - \tau)$$

with surge impedance  $Z = \sqrt{\frac{L'}{C'}}$  and travel time  $\tau = d \cdot \sqrt{L'C'}$ . This model can be represented by a two-port equivalent circuit, where the electrical conditions at port "s" are transferred after a time delay  $\tau$  to port "r" via the controlled current source  $I_{rh}$ .



### Approximation of Series Resistance



Since the shunt conductance is usually negligible, the series resistance is responsible for the major part of the power losses. Such series resistance can be approximated by three lumped resistors, two of which with the value  $\frac{R}{4}$  are placed at both ends of the line while one with the value  $\frac{R}{2}$  is placed in the

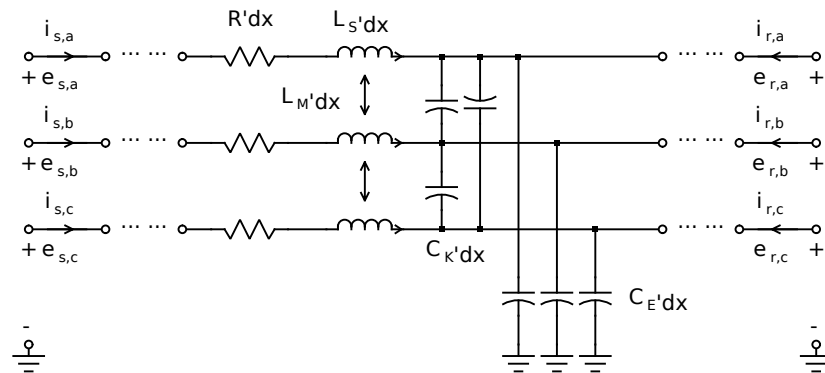
middle.  $R = R' \cdot d$  is the total series resistance of the line. After aggregation and substitution, the original expression of the two equivalent current source becomes:

$$I_{sh} = \frac{1+h}{2} \cdot \left( \frac{1}{Z_R} \cdot e_r(t-\tau) + h \cdot i_r(t-\tau) \right) + \frac{1-h}{2} \left( \frac{1}{Z_R} \cdot e_s(t-\tau) + h \cdot i_s(t-\tau) \right)$$

$$I_{rh} = \frac{1+h}{2} \cdot \left( \frac{1}{Z_R} \cdot e_s(t-\tau) + h \cdot i_s(t-\tau) \right) + \frac{1-h}{2} \left( \frac{1}{Z_R} \cdot e_r(t-\tau) + h \cdot i_r(t-\tau) \right)$$

with  $Z_R = Z + \frac{R}{4}$  and  $h = \frac{Z - \frac{R}{4}}{Z + \frac{R}{4}}$ .

### Three-Phase Line



The differential equations of a 3-phase system with vector variables  $\vec{e} = [e_a, e_b, e_c]^T$ ,  $\vec{i} = [i_a, i_b, i_c]^T$  can be expressed as:

$$-\frac{\partial \vec{e}}{\partial x} = \mathbf{L}' \cdot \frac{\partial \vec{i}}{\partial t} + \mathbf{R}' \cdot \vec{i}$$

$$-\frac{\partial \vec{i}}{\partial x} = \mathbf{C}' \cdot \frac{\partial \vec{e}}{\partial t}$$

Under the assumption of symmetrical phase parameters, the per unit length inductance, capacitance and resistance can be written in matrix form:

$$\mathbf{L}' = \begin{bmatrix} L'_S & L'_M & L'_M \\ L'_M & L'_S & L'_M \\ L'_M & L'_M & L'_S \end{bmatrix}$$



$$\mathbf{C}' = \begin{bmatrix} C'_E + 2 \cdot C'_K & -C'_K & -C'_K \\ -C'_K & C'_E + 2 \cdot C'_K & -C'_K \\ -C'_K & -C'_K & C'_E + 2 \cdot C'_K \end{bmatrix}$$

$$\mathbf{R}' = \begin{bmatrix} R' & 0 & 0 \\ 0 & R' & 0 \\ 0 & 0 & R' \end{bmatrix}$$

The presence of off-diagonal elements (mutual inductance and coupling capacitance) in the matrix make it difficult to solve the equation system. However, this can be overcome with the help of modal transformations. If the differential equations are multiplied by a transformation matrix  $\mathbf{T}$  on the left side

$$-(\mathbf{T} \cdot \frac{\partial \vec{e}}{\partial x}) = (\mathbf{T} \cdot \mathbf{L}' \cdot \mathbf{T}^{-1}) \cdot (\mathbf{T} \cdot \frac{\partial \vec{i}}{\partial t}) + (\mathbf{T} \cdot \mathbf{R}' \cdot \mathbf{T}^{-1}) \cdot (\mathbf{T} \cdot \vec{i})$$

$$-(\mathbf{T} \cdot \frac{\partial \vec{i}}{\partial x}) = (\mathbf{T} \cdot \mathbf{L}' \cdot \mathbf{T}^{-1}) \cdot (\mathbf{T} \cdot \frac{\partial \vec{e}}{\partial t})$$

with

$$\mathbf{T} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & 1 & -2 \end{bmatrix}$$

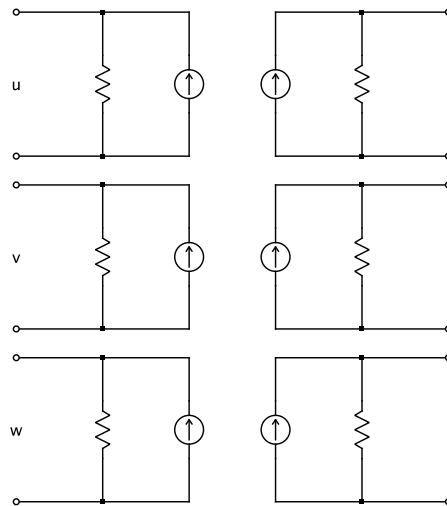
the off-diagonal elements of the inductance, capacitance and resistance matrix can be eliminated:

$$\mathbf{L}'_{\text{mod}} = \mathbf{T} \cdot \mathbf{L}' \cdot \mathbf{T}^{-1} = \begin{bmatrix} L'_u & 0 & 0 \\ 0 & L'_v & 0 \\ 0 & 0 & L'_w \end{bmatrix}$$

$$\mathbf{C}'_{\text{mod}} = \mathbf{T} \cdot \mathbf{C}' \cdot \mathbf{T}^{-1} = \begin{bmatrix} C'_u & 0 & 0 \\ 0 & C'_v & 0 \\ 0 & 0 & C'_w \end{bmatrix}$$

$$\mathbf{R}'_{\text{mod}} = \mathbf{T} \cdot \mathbf{R}' \cdot \mathbf{T}^{-1} = \begin{bmatrix} R' & 0 & 0 \\ 0 & R' & 0 \\ 0 & 0 & R' \end{bmatrix} = \mathbf{R}'$$

Thus the original system, in which the three phases are coupled, has been converted to three decoupled systems in the modal domain (denoted as  $u$ ,  $v$ ,  $w$ ). They can be treated separately in the same way as the single-phase system.



The simulation output in the modal domain should be eventually transformed back into the phase domain via the inverse of the matrix  $\mathbf{T}'$ .

## Parameters

### Self inductance per unit length

The series self inductance  $L'_S$  per unit length. If the length  $l$  is specified in meters (m) the unit of  $L'_S$  is henries per meter (H/m).

For a pi-section line, the self inductance can be specified individually per phase by providing a 3-element vector of the form  $L' = [L'_1 \ L'_2 \ L'_3]$ .

### Mutual inductance per unit length

The series mutual inductance  $L'_M$  per unit length. If the length  $l$  is specified in meters (m) the unit of  $L'_M$  is henries per meter (H/m).

In a pi-section line, the mutual inductances  $M'_{ij}$  between the i-th and j-th phase can be specified individually by providing a 3-element vector  $M' = [M'_{12} \ M'_{13} \ M'_{23}]$  containing the upper triangular coupling matrix.

### Resistance per unit length

The series resistance  $R'$  per unit length. If the length  $l$  is specified in meters (m) the unit of  $R'$  is ohms per meter ( $\Omega$ /m).

For a pi-section line, the parameter can be a vector of the form  $R' = [R'_1 \ R'_2 \ R'_3]$ .

### Neutral capacitance per unit length

The line-to-neutral capacitance  $C'_N$  per unit length. If the length  $l$  is specified in meters (m) the unit of  $C'_N$  is farads per meter (F/m).

For a pi-section line, this parameter can be a vector of the form  $C'_N = [C'_{N1} \ C'_{N2} \ C'_{N1}]$ .

### Coupling capacitance per unit length

The line-to-line capacitance  $C'_C$  per unit length. If the length  $l$  is specified in meters (m) the unit of  $C'_C$  is farads per meter (F/m).

In a pi-section line, the coupling capacitance  $C'_{ij}$  between the i-th and j-th phase can be specified individually by providing a 3-element vector  $C' = [C'_{12} \ C'_{13} \ C'_{23}]$  containing the upper triangular coupling matrix.

### Length

The length  $l$  of the line. The unit of  $l$  must match the units  $L'_S$ ,  $L'_M$ ,  $R'$ ,  $C'_N$  and  $C'_C$  are based on.

### Number of pi sections

Number of sections used to model the transmission line. The default is 3. This parameter only affects the pi-section implementation.

## Reference

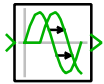
- H. Dommel: "Digital Computer Solution of Electromagnetic Transients in Single and Multiple Networks", IEEE Transactions on Power Apparatus and Systems, Vol. PAS88, No. 4, April, 1969

## Transport Delay

**Purpose** Delay continuous input signal by fixed time

**Library** Control / Delays

**Description** The Transport Delay outputs a signal that *approximates* a past value of the input signal:



$$y(t) \approx u(t - T_d)$$

For this purpose, the Transport Delay continuously records the input signals in an internal ring buffer. The output values are computed by looking up the samples nearest to  $t - T_d$  in the input buffer and performing a first order (linear) interpolation. The input signal can be a scalar or vector.

---

**Note** The Transport Delay should *not* be used to delay non-smooth signals such as rectangular or triangular signals because the solver is not guaranteed to make a simulation step at the precise instants required to accurately reproduce the discontinuities in the delayed signal.

- To generate phase-shifted rectangular or triangular signals, use the Pulse Generator (see page 533) or the Triangular Wave Generator (see page 689) and set the **Phase delay** parameter appropriately.
  - To delay arbitrary signals that only change at discrete instants, use the Pulse Delay (see page 531).
- 

### Parameters

#### Time delay $T_d$

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 639) 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:

#### 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 ( $\Omega$ ). 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 115). 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

This parameter is used only if the device has an internal thermal impedance and specifies the temperature of the thermal capacitance at the junction at simulation start. The temperatures of the other thermal capacitances are initialized based on a thermal “DC” analysis. If the parameter is left blank, all temperatures are initialized from the external temperature. See also “Temperature Initialization” (on page 121).

### Probe Signals

#### 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 Signal

#### Output

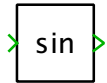
The block output signal.

## 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.



### 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  $[0 \dots 2\pi]$  or degrees  $[0 \dots 360]$ .

### Probe Signals

#### Input

The block input signal.

#### Output

The block output signal.

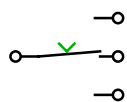


## Triple Switch

**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

#### 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.

### Probe Signal

#### 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.

## Trigger

**Purpose** Control execution of an atomic subsystem

**Library** System

### Description



The Trigger block is used in an atomic subsystem (see “Virtual and Atomic Subsystems” on page 603) to create a triggered subsystem. When you copy a Trigger block into the schematic of a subsystem, a corresponding trigger terminal will be created on the Subsystem block. In order to move this terminal around the edges of the Subsystem block, hold down the **Shift** key while dragging the terminal with the left mouse button or use the middle mouse button.

A triggered subsystem is executed when the trigger signal changes in the manner specified by the **Trigger type** parameter:

#### **rising**

The subsystem is executed when the trigger signal changes from 0 to a non-zero value.

#### **falling**

The subsystem is executed when the trigger signal changes from a non-zero value to 0.

#### **either**

The subsystem is executed when the trigger signal changes from 0 to a non-zero value or vice versa.

The trigger signal may be a vector signal. In this case the triggered subsystem is executed when *any* trigger signal changes in the specified manner.

If the sample time of the Subsystem block is not inherited, the trigger signal will be evaluated only at the instants specified by the sample time parameter.

---

**Note** A triggered subsystem can only contain components that have an inherited or constant sample time. In particular, it cannot contain any physical components.

---

### Parameters

#### **Width**

The width of the trigger signal. The default auto means that the width is inherited from connected blocks.

**Trigger type**

The direction of the edges of the trigger signal upon which the subsystem is executed, as described above.

**Show output port**

When this parameter is set to on, the Trigger block shows an output terminal with the same width as the trigger signal. The output signal will be 1 when the trigger signal has a rising edge, -1 when the trigger signal has a falling edge and 0 at all other times.

**Probe Signal****Output**

The output signal of the Trigger block as described for the parameter

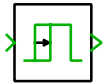
**Show output port.**

## 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:



- When the input signal changes from 0 to non-zero, the output signal will be set to 1 after the dead time has passed, provided that the input signal has remained non-zero.
- When the input signal becomes 0, the output is immediately set to 0.

**Parameter** **Dead time source**  
Specifies whether the dead time is determined by the **Dead time** parameter (**internal**) or by an external input signal (**external**).

**Dead time**  
Time by which the turn-on event is delayed. If set to 0, the delay is disabled, i.e. the output is set to 1 immediately when the input signal becomes non-zero.

**Dead time rounding (fixed-step)**  
If the dead time is determined by an external signal and the Turn-on Delay is used with a fixed-step solver, this parameter specifies how the dead time is rounded to an integer multiple of the fixed-step size.

**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

$$i = \frac{d}{dt}C \cdot v + C \cdot \frac{d}{dt}v$$

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) followed by their derivatives with respect to time, e.g. for two coupled phases:  $[C_{11} C_{12} C_{21} C_{22} \frac{d}{dt}C_{11} \frac{d}{dt}C_{12} \frac{d}{dt}C_{21} \frac{d}{dt}C_{22}]$ . 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.

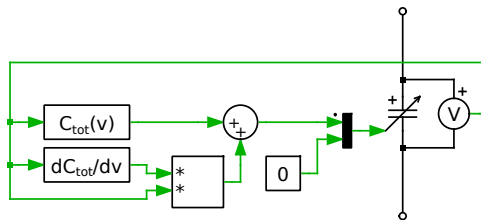
### Saturable Capacitor Modeling

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

$$\begin{aligned}
 i &= \frac{dQ}{dt} \\
 &= \frac{d}{dt} (C_{\text{tot}} \cdot v) \\
 &= C_{\text{tot}} \cdot \frac{dv}{dt} + \frac{dC_{\text{tot}}}{dt} \cdot v \\
 &= C_{\text{tot}} \cdot \frac{dv}{dt} + \frac{dC_{\text{tot}}}{dv} \cdot \frac{dv}{dt} \cdot v \\
 &= \left( C_{\text{tot}} + \frac{dC_{\text{tot}}}{dv} \cdot v \right) \cdot \frac{dv}{dt} ,
 \end{aligned}$$

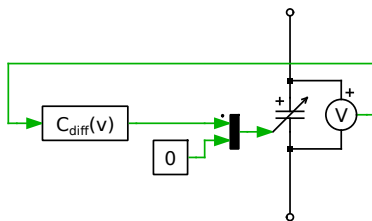
which can be implemented as follows:



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:



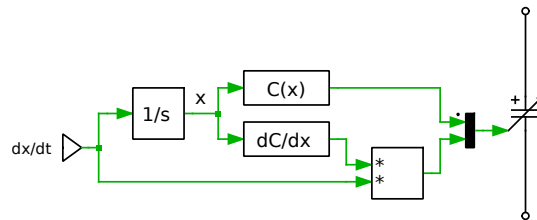
Note that in both cases the  $\frac{d}{dt}C$ -input of the Variable Capacitor is zero!

## 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

$$\begin{aligned} i &= C \cdot \frac{dv}{dt} + \frac{dC}{dt} \cdot v \\ &= C \cdot \frac{dv}{dt} + \frac{dC}{dx} \cdot \frac{dx}{dt} \cdot v \end{aligned}$$

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) followed by their derivatives with respect to time, e.g. for two coupled phases:  $[L_{11} \ L_{12} \ L_{21} \ L_{22} \ \frac{d}{dt}L_{11} \ \frac{d}{dt}L_{12} \ \frac{d}{dt}L_{21} \ \frac{d}{dt}L_{22}]$ . 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 608).

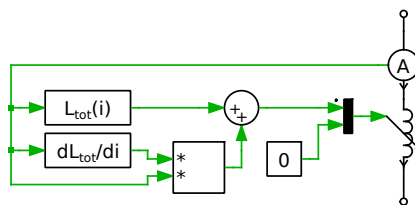
### Saturable Inductor Modeling

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 564).

With the *total* inductivity  $L_{\text{tot}}(i) = \Psi/i$  you have

$$\begin{aligned} v &= \frac{d\Psi}{dt} \\ &= \frac{d}{dt} (L_{\text{tot}} \cdot i) \\ &= L_{\text{tot}} \cdot \frac{di}{dt} + \frac{dL_{\text{tot}}}{dt} \cdot i \\ &= L_{\text{tot}} \cdot \frac{di}{dt} + \frac{dL_{\text{tot}}}{di} \cdot \frac{di}{dt} \cdot i \\ &= \left( L_{\text{tot}} + \frac{dL_{\text{tot}}}{di} \cdot i \right) \cdot \frac{di}{dt} , \end{aligned}$$

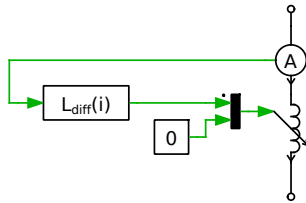
which can be implemented as follows:



With the *differential* inductivity  $L_{\text{diff}}(i) = d\Psi/di$  you have

$$\begin{aligned}
 v &= \frac{d\Psi}{dt} \\
 &= \frac{d\Psi}{di} \cdot \frac{di}{dt} \\
 &= L_{\text{diff}} \cdot \frac{di}{dt} \quad ,
 \end{aligned}$$

which can be implemented as follows:



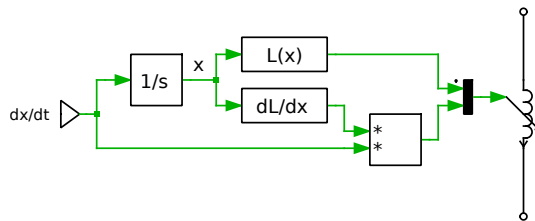
Note that in both cases the  $\frac{d}{dt}L$ -input of the Variable Inductor is zero!

### 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  $\Phi$  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

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

## Parameter

### 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

**Purpose** Resistance controlled by a signal

**Library** Electrical / Passive Components

**Description** This component provides an ideal resistor whose resistance is controlled by the input signal.



---

**Note** The Variable Resistor creates an algebraic loop. See section “Block Sorting” (on page 29) for more information on algebraic loops.

---

**Probe Signals** The small dot in the component icon marks the positive terminal.

**Resistor voltage**

The voltage measured across the resistor from the positive to the negative terminal.

**Resistor current**

The current flowing into the positive terminal.

**Resistor power**

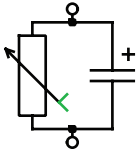
The power consumed by the resistor.

## 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 Signal

#### 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.

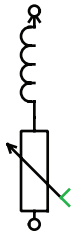
## Variable Resistor with Constant Inductor

**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 Signal

#### 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.

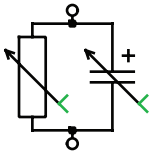


## Variable Resistor with Variable Capacitor

**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 695).

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) followed by their derivatives with respect to time, e.g. for two coupled phases:  $[C_{11} C_{12} C_{21} C_{22} \frac{d}{dt} C_{11} \frac{d}{dt} C_{12} \frac{d}{dt} C_{21} \frac{d}{dt} C_{22}]$ . 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 Signal****Capacitor voltage**

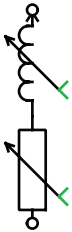
The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with “+” is greater than the potential at the unmarked terminal.

## Variable Resistor with Variable Inductor

**Purpose** Controlled resistance in series with controlled inductance

**Library** Electrical / Passive Components

### Description



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

$$v = \left( R + \frac{d}{dt} L \right) \cdot i + L \cdot \frac{d}{dt} i$$

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 698).

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) followed by their derivatives with respect to time, e.g. for two coupled phases:  $[L_{11} \ L_{12} \ L_{21} \ L_{22} \ \frac{d}{dt} L_{11} \ \frac{d}{dt} L_{12} \ \frac{d}{dt} L_{21} \ \frac{d}{dt} L_{22}]$ . 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.
- The control signal for the momentary inductance values must be continuous. Discontinuous changes will produce non-physical results.

### 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 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 is indicated by a small arrow in the component symbol. The default of the initial current is 0.

## **Probe Signal**

### **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.

---

### Parameters

#### Discretization behavior

Specifies whether a zero-order hold or a first-order hold is applied to the input signal when the model is discretized. For details, see “Physical Model Discretization” (on page 33).

The option **Non-causal zero-order hold** applies a zero-order hold with the input signal value from the *current* simulation step instead of the *previous* one. This option can be used to compensate for a known delay of the input signal.

#### Allow state-space inlining

**For expert use only!** When set to on and the input signal is a linear combination of electrical measurements, PLECS will eliminate the input variable from the state-space equations and substitute it with the corresponding output variables. The default is off.

### 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



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

$$v = A \cdot \sin(\omega \cdot t + \varphi)$$

where  $t$  is the simulation time.

If a variable-step solver is used, the solver step size is automatically limited to ensure that a smooth voltage waveform is produced.

---

**Note** A voltage source may not be short-circuited or connected in parallel to a capacitor or any other voltage 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 voltage, in volts (V). The default is 1.

#### Frequency

The angular frequency  $\omega$ , in  $\text{s}^{-1}$ . The default is  $2 \cdot \pi \cdot 50$  which corresponds to 50 Hz.

#### Phase

The phase shift  $\varphi$ , in radians. The default is 0.

### 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 (3-Phase)

**Purpose** Generate 3-phase sinusoidal voltage

**Library** Electrical / Sources

### Description



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

$$v_i = A_i \cdot \sin(2\pi \cdot f \cdot t + \varphi_i + \Delta\varphi_i)$$

where  $t$  is the simulation time and  $\varphi_0 = 0$ ,  $\varphi_1 = -2/3 \cdot \pi$  and  $\varphi_2 = 2/3 \cdot \pi$ .

---

**Note** A voltage source may not be short-circuited or connected in parallel to a capacitor or any other voltage source.

---

### Parameters

#### 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  $[A_0, A_1, A_2]$ .

#### Frequency

The frequency  $f$ , in Hertz (Hz).

#### Phase offset

The phase offset  $\Delta\varphi$ , in radians. The value can be given as a scalar or as a vector with three elements  $[\Delta\varphi_0, \Delta\varphi_1, \Delta\varphi_2]$ .

#### Neutral point

Show or hide the neutral point terminal.

### Probe Signals

#### Source voltage

The source voltages in volts (V) as a vectorized signal.

#### Source current

The currents flowing through the source, in amperes (A) as a vectorized signal.

#### Source power

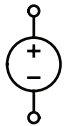
The combined instantaneous output power of the source, in watts (W).

## 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).

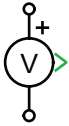


# Voltmeter

**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 583) 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 Signal

### Measured voltage

The measured voltage in volts (V).

## White Noise

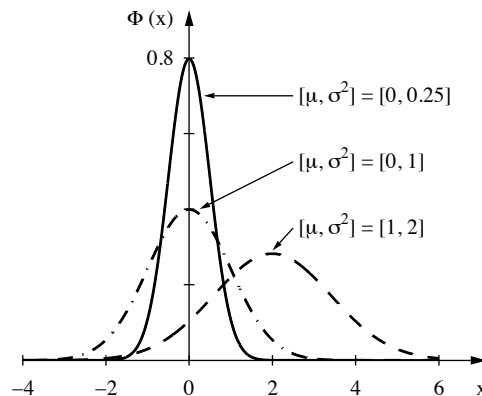
**Purpose** Generate normally distributed random numbers

**Library** Control / Sources

### Description



The White Noise block generates normally distributed (Gaussian) random numbers for modeling band-limited white noise. The mean and the standard deviation of the underlying distribution can be configured in the component dialog. The figure below illustrates the distribution for three different sets of parameters. The seed of the generator initializes the algorithm at the simula-



tion start. For the same seed, the sequence of random numbers is reproduced in every simulation run. If this behavior is undesired the system time can be used as a seed. To minimize correlation effects, it is recommended to use different seeds if multiple random generators are used in one model.

### Parameters

#### Mean $\mu$

The mean value of the distribution.

#### Standard deviation $\sigma$

The standard deviation of the distribution.

#### Seed

The seed used to initialize the White Noise generator.

#### Sample Time

The sampling period used for generating random output values.

**References**

Mersenne Twister: [http://en.wikipedia.org/wiki/Mersenne\\_twister](http://en.wikipedia.org/wiki/Mersenne_twister)

Leva, J. L., "A Fast Normal Random Number Generator", ACM Transactions on Mathematical Software, vol. 18, no. 4, pp. 449-453, 1992.

## 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 the 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.

The sign conventions used for the electrical and magnetic quantities are illustrated in the figure below for the two polarities of the winding. Notice the positions of the black dot and the brown arrow in the winding symbol.



**Parameters** **Number of turns**  
Specifies the number of winding turns.

**Polarity**  
Specifies the polarity of the winding. Choosing a negative polarity is equivalent to specifying a negative number of 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.

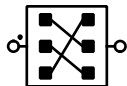
**Vector:** The length of the vector determines the number of individual wires. Each element specifies the width of the corresponding individual wire.

## 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

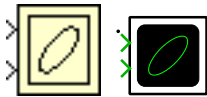
A vector with the indices of the input elements that the output bus should contain.

## 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 101).



### Parameters

#### Title

The name which is displayed above the plot.

#### Sample time

The block sample time used to sample the input signals. The default is -1 (inherited). Other valid settings are 0 (continuous) or a valid fixed-step discrete sample time pair (see “Sample Times” on page 36).

#### 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.

#### 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.

#### Plot style

This option lets you choose whether you would like to plot trajectories, vectors or a combination of both. If vectors are drawn, the aspect ratio (see below) is automatically fixed to 1:1.



**Keep aspect ratio (x:y)**

If this option is selected, the aspect ratio of the axes is kept constant. A ratio of  $x : y$  ensures that the length of  $x$  units on the x-axis matches the length of  $y$  units on the y-axis.

**Axis labels**

The axis labels that are displayed on the x- and y-axis.

**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.

## 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 361), 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

#### 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.

#### Zener resistance

The resistance  $R_z$ , in ohms ( $\Omega$ ), if the diode is reverse conducting.

#### Forward voltage

Additional dc voltage  $V_f$  in volts (V) between anode and cathode when the diode is forward conducting. The default is 0.

#### On-resistance

The resistance  $R_f$  of the forward conducting device, in ohms ( $\Omega$ ). The default is 0.

### Probe Signals

#### Diode voltage

The voltage measured between anode and cathode.

#### Diode current

The current through the diode flowing from anode to cathode.

#### Forward conductivity

Conduction state of the positive internal switch. The signal outputs 1 when the diode is conducting in forward direction, and 0 otherwise.

#### Reverse conductivity

Conduction state of the negative internal switch. The signal outputs 1 when the diode is conducting in reverse direction, and 0 otherwise.

## 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 36).

**Probe Signals** **Input**  
The input signal.

**Output**  
The output signal.



# Additional Simulink Blocks

This chapter lists the contents of the PLECS Extras library for PLECS Blockset in alphabetical order.

## AC Sweep

**Purpose** Perform AC sweep

**Library** PLECS Extras / Analysis Tools

### Description



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.

An ac sweep can be started either by clicking the button **Start analysis** or with the MATLAB command

```
placsweep(block);
```

where *block* 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.

For additional information see section “AC Analysis” (on page 164).

### 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

$$A_i = A_1 \cdot \sqrt{f_i/f_1}$$

**Show reference input**

If this parameter is set to on, the block will show an additional input port, and the signal that is connected to it is used to determine the spectrum  $U(s)$  for the calculation of the transfer function  $G(s) = Y(s)/U(s)$ . Otherwise,  $U(s)$  is determined by the internally generated perturbation signal.

**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:

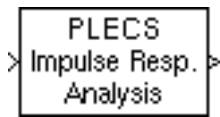
- Discrete Fourier Transform (see page 367)
- Discrete Mean Value (see page 372)
- Discrete RMS Value (see page 373)
- Discrete Total Harmonic Distortion (see page 375)

## Impulse Response Analysis

**Purpose** Perform impulse response analysis

**Library** PLECS Extras / Analysis Tools

### Description



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.

An analysis can be started either by clicking the button **Start analysis** or with the MATLAB command

```
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 that override the settings in the dialog box.

For additional information see section “Impulse Response Analysis” (on page 164).

### Parameters

#### System period length

The period length of the unperturbed system.

#### Simulation start time

The simulation start time for the impulse response analysis.

#### 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.

#### Perturbation

The amplitude of the perturbation signal.

**Compensation for discrete pulse**

Specifies whether and how the effect of the sampling should be compensated. See section “Compensation for Discrete Pulse” (on page 165) 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**

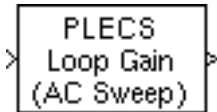
Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (`error`, `warning`, `none`).

## Loop Gain Analysis (AC Sweep)

**Purpose** Determine loop gain of closed control loop

**Library** PLECS Extras / Analysis Tools

### Description



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.

An analysis can be started either by clicking the button **Start analysis** or with the MATLAB command

```
placsweep(block);
```

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 164).

---

**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.

---

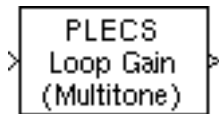
**Parameters** The parameters are identical to those of the AC Sweep block (see page 728).

## Loop Gain Analysis (Multitone)

**Purpose** Determine loop gain of closed control loop

**Library** PLECS Extras / Analysis Tools

### Description



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.

An analysis can be started either by clicking the button **Start analysis** or with the MATLAB command

```
plmultitone(block);
```

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 “Multitone Analysis” (on page 166).

---

**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 Multitone Analysis block (see page 737).

## Modulators

Please refer to the documentation on the following components:

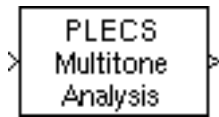
- 2-Pulse Generator (see page 309)
- 3-Phase Overmodulation (see page 312)
- 6-Pulse Generator (see page 313)
- Blanking Time (see page 327)
- Blanking Time (3-Level) (see page 328)
- Sawtooth PWM (see page 569)
- Sawtooth PWM (3-Level) (see page 571)
- Symmetrical PWM (see page 612)
- Symmetrical PWM (3-Level) (see page 614)
- Peak Current Controller (see page 515)

## Multitone Analysis

**Purpose** Perform a multitone analysis

**Library** PLECS Extras / Analysis Tools

### Description



The Multitone 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 multitone signal containing 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.

A multitone analysis can be started either by clicking the button **Start analysis** or with the MATLAB command

```
plmultitone(block);
```

where *block* is the Simulink handle or the full block path of the Multitone 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 “Multitone Analysis” (on page 166).

### Parameters

#### Frequency range

A vector containing the lowest and highest perturbation frequency.

#### Amplitude

The amplitude of the perturbation signal.

#### Show reference input

If this parameter is set to on, the block will show an additional input port, and the signal that is connected to it is used to determine the spectrum  $U(s)$  for the calculation of the transfer function  $G(s) = Y(s)/U(s)$ . Otherwise,  $U(s)$  is determined by the internally generated perturbation signal.

#### Init. simulation period

The duration of an initial simulation performed before the response is measured. It is assumed that during this period, the system reaches its steady state. The total simulation duration will be the sum of this parameter and one period of the base frequency signal.

**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 `plmultitone` 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`).

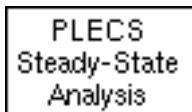


## Steady-State Analysis

**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.

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 161).

### 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 `plsteadystate` 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`).

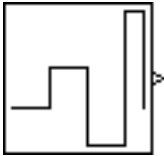
## Timer

**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.



### 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 651)
- Transformation 3ph->SRF (see page 652)
- Transformation RRF->3ph (see page 653)
- Transformation RRF->SRF (see page 654)
- Transformation SRF->3ph (see page 655)
- Transformation SRF->RRF (see page 656)



plexim  
electrical engineering software

---