

PLECS

THE SIMULATION PLATFORM FOR POWER ELECTRONIC SYSTEMS

User Manual Version 5.0

How to Contact Plexim:

☎	+41 44 533 51 00	Phone
✉	Plexim GmbH Technoparkstrasse 1 8005 Zurich Switzerland	Mail
@	`info@plexim.com`	Email
	`http://www.plexim.com`	Web

PLECS User Manual

© 2002–2026 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 and Nanostep are a registered trademarks 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

Contents **iii**

Before You Begin **1**

Installing PLECS Standalone 2
Installation on Microsoft Windows 2
Installation on macOS 2
Installation on Linux 2
Licensing 2
Installing PLECS Blockset 4
Installation on Microsoft Windows 4
Installation on macOS 4
Installation on Linux 5
Licensing 5
Configuring the MATLAB Search Path 6
Configuring PLECS 6
Installing Different Versions of PLECS Blockset in Parallel 6
Uninstalling PLECS Blockset 7
License File Location 8
Network Licensing 9
What's New in Version 5.0 10
Major New Features 10
Further Enhancements 10

1 Getting Started **13**
Getting Started with PLECS Standalone 14
A Simple Passive Network 14
Buck Converter 18
Getting Started with PLECS Blockset 22
A Simple Passive Network 22
Buck Converter 27

2 How PLECS Works **31**
Modeling Dynamic Systems 32
System Equations 32
Block Diagrams 33
Physical Models 33
Simulating Dynamic Systems 34
Model Initialization 34
Model Execution 37
Fixed-Step Simulation 39
Sampled Data Systems 43
Sample Times 43
Sample Time Inheritance 44
Continuous Sample Time Conflicts 45
Multirate Systems 46
Data Types 48
Fixed-Point Data Types 48
Blocks with an Implicit Output Data Type 49
Specifying an Output Data Type 49
Data Type Inheritance 50
Data Type Overflows 51
Conventions & Definitions 52
Three-Phase Sequence Convention 52
Reference Frame Transformations 52

3 Using PLECS **55**
Using PLECS Standalone 56
Creating a New Model 56
Importing a Schematic From PLECS Blockset 56
Using PLECS Blockset 57
Creating a New Circuit 57
Importing a Schematic From PLECS Standalone 58
Opening a PLECS Standalone Model 58
Customizing the Circuit Block 58
Configuring PLECS 60
General 60
Schematic 61
Window 62
Library 63
Thermal 64
Scope 64
Update 65
Coder 65
Using the Library Browser 66
Components 67
Copying a Component into a Schematic 67
Moving a Component 67
Specifying Component Parameters 67
Displaying Parameters in the Schematic 69
Changing Parameters of Multiple Components 69
Changing Parameters During a Simulation 69
Changing Component Names 69
Placing the Component Label 70
Changing the Orientation of Components 70
Disabling Components 70

Getting Component Help 71
Libraries 72
Creating a New Library in PLECS Standalone 72
Creating a New Library in PLECS Blockset 72
Creating a Library Reference 72
Updating a Library Reference 73
Breaking a Library Reference 73
Connections 74
Signal Connections and Physical Connections 74
Creating Connections 74
Creating Branches 74
Automatic Creation of Connections 74
Editing Connections 76
Vectorization 77
Vectorized Sources 77
Signal De-/Multiplexer 78
Vectorizing Physical Components 78
Annotations 81
Text Annotations 81
Box Annotations 81
Line Annotations 81
Subsystems 82
Creating a Subsystem by Adding the Subsystem Block 82
Creating a Subsystem by Grouping Existing Blocks 82
Arranging Subsystem Terminals 82
Resizing a Subsystem Block 83
Masking Subsystems 84
Mask Icon 84
PLECS Colors 92
PLECS Icon Library 92

Mask Dialog 98
Mask Workspace 103
Mask Probe Signals 104
Mask Documentation 105
Unprotecting Masked Subsystems 107
Getting Started with Lua 107
Assertions 113
Assertion Blocks 113
Component Assertions 114
Locating Assertions 115
PLECS Probe 116
Copying a Probe 117
Using the PLECS Scope 119
Getting Started 119
Legend 119
Zoom Operations 119
Scrolling 120
Y-Axis Auto-Scaling 121
Changing Curve Properties 121
Spreading Signals 121
Cursors 122
Fourier Analysis 123
Saving a View 123
Adding Traces 123
Saving and Loading Trace Data 124
Scope Parameters 124
Printing and Exporting 124
Using the Fourier Analysis 126
Calculation Parameters 126
Display Parameters 127

Zoom, Export and Print . . . 128
Calculation of the Fourier coefficients . . . 128
Using the XY Plot . . . 129
Time Range Window . . . 129
Zoom, Save View, Export and Print . . . 130
Circuit Browser . . . 131
Viewing Options . . . 132
Controlling Access to Circuits and Subsystems . . . 134
Encrypting Circuits and Subsystems . . . 134
Simulation Parameters . . . 135
PLECS Standalone Parameters . . . 135
PLECS Blockset Parameters . . . 143
Error Messages . . . 150
Could not solve the algebraic loop . . . 150
Encountered 1001 consecutive zero-crossings . . . 153
State discontinuity after switching . . . 155
Derivatives are infinite or NaN . . . 159
Negative losses . . . 163
State-source dependency of components . . . 164
Out of memory . . . 167
Exporting Schematics . . . 168
Web-Based Simulation Designer . . . 169

4 Thermal Modeling 171

Thermal Concepts . . . 172
Heat Sink Concept . . . 172
Thermal Loss Dissipation . . . 173
Heat Sinks and Subsystems . . . 177
Temperature Initialization . . . 177
Thermal Description Parameter . . . 179
Thermal Modeling with Parallel Devices . . . 181

Thermal Library 183
Library Structure 183
Global and Local Data Sheets 183
Creating New Data Sheets 184
Browsing the Thermal Library 184
Thermal Editor 185
Thermal Description for a Single Device 186
Editing Switching Losses 187
Editing Conduction Losses 187
Editing the Thermal Equivalent Circuit 188
Adding Custom Variables 191
Adding Custom Lookup Tables 193
Editing Lookup Tables 194
Thermal Package Description 195
Importing Data from Graphical Datasheets 199
Semiconductor Loss Specification 205

5 Magnetic Modeling **209**
Equivalent circuits for magnetic components 210
Coupled inductors 210
Reluctance-resistance analogy 211
Permeance-capacitance analogy 212
Magnetic Circuit Domain in PLECS 214
Modeling Non-Linear Magnetic Material 214
Saturation Curves for Soft-Magnetic Material 216
References 216

6 Mechanical Modeling **217**
Flanges and Connections 219
Force/Torque Flows and Sign Conventions 220
Positions and Angles 221

Initial Conditions . . . 221
Angle Wrapping . . . 222
Ideal Clutches . . . 223
Inelastic Collisions . . . 223

7 Analysis Tools 227
Steady-State Analysis . . . 228
Algorithm . . . 228
Fast Jacobian Calculation for Thermal States . . . 229
Non-Periodic Case . . . 229
Limitations . . . 230
Reference . . . 230
AC Sweep . . . 231
Impulse Response Analysis . . . 232
Algorithm . . . 232
Reference . . . 232
Multitone Analysis . . . 233
Algorithm . . . 233
Remarks . . . 233
References . . . 234
Frequency Response . . . 235
AC Analysis (Linearized Circuit) . . . 236
Usage in PLECS Standalone . . . 237
Steady-State Analysis . . . 238
AC Sweep . . . 239
Impulse Response Analysis . . . 240
Multitone Analysis . . . 241
Frequency Response . . . 242
AC Analysis (Linearized Circuit) . . . 245
Extraction of State-Space Matrices . . . 246
Application Example . . . 247

Usage in PLECS Blockset 250
Steady-State Analysis 250
AC Sweep / Loop Gain Analysis 252
Impulse Response Analysis 254
Multitone / Loop Gain Analysis 257
Extraction of State-Space Matrices 258
Application Example 259
State-Space Averaging 263
Reference 268

8 C-Scripts 269
How C-Scripts Work 270
C-Script Functions 270
Modeling Discontinuities 272
Sample Time 274
User Parameters 276
Runtime Checks 277
C-Script Examples 279
A Simple Function – Times Two 279
Discrete States – Sampled Delay 279
Continuous States – Integrator 280
Event Handling – Wrapping Integrator 281
Piecewise Smooth Functions – Saturation 283
Multiple Sample Times – Turn-on Delay 284
Load External Files 287
C-Script Macros 289
Deprecated Macros 294

9 State Machines 295
Working with State Machines 296
Working with States 297

Working with Transitions . . . 298
Working with Junctions . . . 299
Working with Annotations . . . 300
State Machine Configuration . . . 300
State Machine Execution . . . 305
Transition Evaluation . . . 305
Trigger Types . . . 306
Trigger Lifetime . . . 307
Execution of Hierarchical State Machines . . . 307
State Machine Examples . . . 310
Oven Control . . . 310
Demo Models . . . 312

10 Simulation Scripts **313**
Simulation Scripts in PLECS Standalone . . . 314
Overview of PLECS Scripting Extensions . . . 315
Example Script . . . 321
RPC Interface in PLECS Standalone . . . 323
Usage Examples . . . 323
Overview of RPC Commands . . . 324
Example Script . . . 327
Scripted Simulation and Analysis Options . . . 329
Command Line Interface in PLECS Blockset . . . 337
Reading and Setting Parameters of Components . . . 337
Handling of Information, Warnings and Errors in Initialization Commands . . . 337
Handling of Traces in Scopes . . . 338
Retrieving Scope Cursor Data . . . 338
Exporting Scope Data . . . 339
Exporting Scope Bitmaps . . . 339
Other CLI Commands . . . 340

Examples 340

11 Code Generation **341**

Code Generation for Physical Systems 342
Maximum Number of Switches 342
Handling Naturally Commutated Devices 342
Ideal and Non-Ideal Switch Models in Electric Circuits 344
Unsupported Components 346
Code Generation with PLECS Standalone 347
Generating Code 348
Task Transitions in Multi-Tasking Mode 353
Simulating a Subsystem in CodeGen Mode 354
Code Generation with PLECS Blockset 355
Standalone Code Generation 355
Integration with Simulink Coder 355
Simulink Coder Options 355
Code Generation Targets 356
Real-Time Code Execution 360
Real-Time Simulations 360
Real-time Code Execution on the RT Box 361

12 Functional Mockup Interface **365**

FMI Standard 366
FMI Version 366
Common Concepts for FMU Export 367
PLECS Model Exchange FMU 368
Limitations 368
PLECS Co-Simulation FMU 370
FMU with Discrete Inputs 370
Limitations 370
Exporting an FMU 371

General Options . . . 371
Terminal Order . . . 372
Documentation . . . 372

13 Diff Tool **375**
How to Compute a Diff . . . 376
Computing a Diff in PLECS Standalone . . . 376
Computing a Diff in PLECS Blockset . . . 377
How to Interpret the Results . . . 379
The Diff Browser . . . 379
Tool Buttons . . . 381
The Diff Views . . . 382
How Diff Works . . . 383
References . . . 383
Limitations . . . 384
Model Conversion . . . 384
Non-Intuitive Results . . . 384

14 SPICE **387**
How PLECS Spice Works . . . 389
Modified Nodal Analysis . . . 389
Differential Algebraic Equation Solver . . . 393
Mixed-Signal Hybrid Simulation . . . 394
Netlists . . . 394
Compact Models . . . 396
Using PLECS Spice . . . 397
Netlist Component . . . 397
Netlist File Reference Component . . . 403
Best Practices . . . 404
SPICE Simulation Parameters . . . 406
PLECS Components Compatible with SPICE . . . 409

Netlist Cheat Sheets . . . 411
Differences to Text-Based SPICE Simulators . . . 414
Netlist Syntax Reference . . . 429
Fundamentals . . . 429
List of Statements . . . 435

15 Components by Category 475
System . . . 476
Assertions . . . 478
Control . . . 479
Sources . . . 479
Math . . . 479
Continuous . . . 480
Delays . . . 480
Discontinuous . . . 480
Discrete . . . 481
Filters . . . 481
Functions & Tables . . . 482
Logical . . . 482
Modulators . . . 483
Transformations . . . 484
State Machine . . . 484
Small Signal Analysis . . . 485
Electrical . . . 486
Connectivity . . . 486
Sources . . . 486
Meters . . . 486
Passive Components . . . 487
Power Semiconductors . . . 487
Power Modules . . . 488
Switches . . . 491

Transformers . . . 491
Machines . . . 491
Converters . . . 492
Nanostep . . . 493
Electronics . . . 495
Model Settings . . . 495
SPICE Components . . . 495
Thermal . . . 497
Connectivity . . . 497
Sources . . . 497
Meters . . . 497
Components . . . 497
Model Settings . . . 498
Magnetic . . . 499
Connectivity . . . 499
Sources . . . 499
Meters . . . 499
Components . . . 499
Mechanical . . . 501
Translational . . . 501
Rotational . . . 502
Additional Simulink Blocks . . . 505

16 Component Reference **507**
1D Look-Up Table . . . 508
2-Pulse Generator . . . 510
2D Look-Up Table . . . 511
3-Level Half Bridge (ANPC) . . . 512
3-Level Half Bridge (NPC) . . . 514
3-Level Half Bridge (T-Type) . . . 516
3-Phase Current Source Inverter . . . 519

3-Phase Dual Active Bridge 521
3-Phase Index-Based Modulation 524
3-Phase NPC Voltage Source Inverter 531
3-Phase Overmodulation 535
3-Phase T-Type Voltage Source Inverter 537
3-Phase Voltage Source Inverter 541
3-Phase Voltage Source Inverter 544
3D Look-Up Table 546
4-Phase Voltage Source Inverter 548
5-Level Half Bridge (ANPC) 551
6-Pulse Generator 553
Abs 554
Air Gap 555
Algebraic Constraint 556
Ambient Temperature 557
Ammeter 558
Angle Sensor 559
Assert Dynamic Lower Limit 560
Assert Dynamic Range 561
Assert Dynamic Upper Limit 562
Assertion 563
Assert Lower Limit 564
Assert Range 565
Assert Upper Limit 566
Bidirectional Phase-Shifted Full-Bridge Converter 567
Blanking Time 571
Blanking Time (3-Level) 572
Boost Converter 573
Boost PFC Converter 576
Boost PFC Converter with LC Filter 579

Breaker 583
Brushless DC Machine 584
Brushless DC Machine (Simple) 587
Buck Converter 591
C-Script 594
Capacitor 597
Chopper (High-Side Switch) 599
Chopper (High-Side Switch with Reverse Diode) 601
Chopper (Low-Side Switch) 603
Chopper (Low-Side Switch with Reverse Diode) 605
CLLLC Converter 607
CLLLC Converter with NPC Half-Bridge 611
Clock 616
Combinatorial Logic 617
Comparator 618
Compare to Constant 619
Configurable Subsystem 620
Constant 621
Constant Heat Flow 622
Constant Temperature 623
Continuous PID Controller 624
Controlled Heat Flow 630
Controlled Temperature 631
Current Source (Controlled) 632
Current Source AC 633
Current Source DC 634
Data Type 635
DC Machine 636
Dead Zone 638
Delay 639

D Flip-flop 640
Diode 641
Diode Pair 644
Diode Rectifier (3ph) 645
Diode with Reverse Recovery 646
Discrete Integrator 649
Discrete Mean Value 653
Discrete PID Controller 654
Discrete State Space 658
Discrete Transfer Function 660
Display 663
DLL 664
Double Switch 668
Dynamic Signal Selector 669
Edge Detection 670
Electrical Algebraic Component 671
Electrical Ground 673
Electrical Label 674
Electrical Model Settings 675
Electrical Port 676
Enable 678
Flux Rate Meter 679
Flyback Converter with Snubber 680
Flying Capacitor Half Bridge 684
FMU 686
Force (Constant) 687
Force (Controlled) 688
Force Sensor 689
Fourier Series 690
Fourier Transform 691

Frequency-doubling LLC Converter 692
From File 696
Full-Bridge Inverter 699
Full-Bridge Inverter with LCL Filter 702
Full-Bridge LLC Converter 706
Full-Bridge LLC Converter with Synchronous Rectification 710
Full Bridges (Series Connected) 713
Function 715
Gain 716
Gear 717
GTO 718
GTO (Reverse Conducting) 720
Half-Bridge Inverter 722
Half-Bridge LLC Converter 725
Half-Bridge LLC Converter with Synchronous Rectification 729
Half Bridge 733
Half Bridges (Low-/High-Side Connected) 735
Heat Flow Meter 737
Heat Sink 738
Hit Crossing 739
Hysteretic Core 740
Ideal 3-Level Converter (3ph) 742
Ideal Converter (3ph) 743
Ideal Transformer 744
IGBT 746
IGBT 3-Level Converter (3ph) 748
IGBT Converter (3ph) 750
IGBT with Diode 752
IGBT with Limited di/dt 754
IGCT (Reverse Blocking) 758

IGCT (Reverse Conducting) 760
Induction Machine (Open Stator Windings) 762
Induction Machine (Slip Ring) 765
Induction Machine (Squirrel Cage) 770
Induction Machine with Saturation 773
Inductor 779
Inertia 781
Initial Condition 782
Integrator 783
JK Flip-flop 785
Leakage Flux Path 787
Linear Core 788
Linear Transformer (2 Windings) 789
Linear Transformer (3 Windings) 791
Logical Operator 793
Magnetic Multiplexer 794
Magnetic Permeance 795
Magnetic Port 796
Magnetic Resistance 798
Magnetic Selector 799
Manual Double Switch 800
Manual Signal Switch 801
Manual Switch 802
Manual Triple Switch 803
Mass 804
Math Function 805
Matrix Bidirectional Phase-Shifted Full-Bridge Converter 806
Matrix CLLLC Converter 810
Matrix Phase-Shifted Full-Bridge Converter 814
Memory 818

Meter (3-Phase) 819
Minimum / Maximum 820
MMF Meter 821
MMF Source (Constant) 822
MMF Source (Controlled) 823
Model Reference 824
Monoflop 826
MOSFET 828
MOSFET Converter (3ph) 831
MOSFET with Diode 832
MOSFET with Limited di/dt 835
Moving Average 837
Multiport Signal Switch 838
Mutual Inductance (2 Windings) 839
Mutual Inductance (3 Windings) 841
Mutual Inductor 843
Netlist 845
Netlist File Reference 847
Non-Excited Synchronous Machine 850
Offset 855
Op-Amp 856
Op-Amp with Limited Output 857
Pause / Stop 858
Peak Current Controller 859
Periodic Average 860
Periodic Impulse Average 861
Permanent Magnet Synchronous Machine 862
Permanent Magnet Synchronous Machine (Open Winding) 866
Phase-Shifted Full-Bridge Converter 870
Pi-Section Line 874

Piece-wise Linear Resistor 876
Planetary Gear Set 878
PLL (Single-Phase) 880
PLL (Three-Phase) 885
Polar to Rectangular 890
Position Sensor 891
Product 892
Pulse Delay 893
Pulse Generator 895
Quantizer 896
Rack and Pinion 897
Ramp 898
Random Numbers 899
Rate Limiter 901
Rectangular to Polar 902
Relational Operator 903
Relay 904
Resistor 905
RMS Value 906
Rotational Algebraic Component 907
Rotational Backlash 909
Rotational Clutch 910
Rotational Damper 911
Rotational Friction 912
Rotational Friction (Controlled) 914
Rotational Hard Stop 915
Rotational Model Settings 917
Rotational Multiplexer 918
Rotational Port 919
Rotational Reference 920

Rotational Selector 921
Rotational Speed (Constant) 922
Rotational Speed (Controlled) 923
Rotational Speed Sensor 924
Rounding 925
Saturable Capacitor 926
Saturable Core 928
Saturable Inductor 931
Saturable Transformers 934
Saturation 936
Sawtooth PWM 937
Sawtooth PWM (3-Level) 939
Scalar Expander 941
Scope 942
Set/Reset Switch 944
Signal Demultiplexer 946
Signal From 947
Signal Goto 948
Signal Inport 949
Signal Multiplexer 951
Signal Outport 952
Signal Selector 954
Signal Switch 955
Signum 956
Simple Dual Active Bridge 957
Sine Wave 960
Small Signal Gain 961
Small Signal Perturbation 962
Small Signal Response 963
Space Vector PWM 964

Space Vector PWM (3-Level) 968
SR Flip-flop 972
State Machine 973
State Space 974
Step 976
Subsystem 977
Sum 980
Switch 981
Switched Reluctance Machine 982
Switch Loss Calculator 986
Symmetrical PWM 988
Symmetrical PWM (3-Level) 992
Synchronous Machine (Round Rotor) 995
Synchronous Machine (Salient Pole) 1001
Synchronous Reluctance Machine 1007
Task Frame 1011
Task Transition 1012
Thermal Capacitor 1013
Thermal Chain 1014
Thermal Ground 1015
Thermal Model Settings 1016
Thermal Multiplexer 1017
Thermal Package Impedance 1018
Thermal Port 1019
Thermal Resistor 1021
Thermal Selector 1022
Thermometer 1023
Thyristor 1024
Thyristor Pair 1026
Thyristor Rectifier/Inverter 1027

Thyristor with Reverse Recovery 1028
To File 1030
Torque (Constant) 1032
Torque (Controlled) 1033
Torque Sensor 1034
Torsion Spring 1035
Total Harmonic Distortion 1036
Transfer Function 1037
Transformation 3ph->RRF 1040
Transformation 3ph->SRF 1041
Transformation RRF->3ph 1042
Transformation RRF->SRF 1043
Transformation SRF->3ph 1044
Transformation SRF->RRF 1045
Transformers (3ph, 2 Windings) 1046
Transformers (3ph, 3 Windings) 1049
Translational Algebraic Component 1052
Translational Backlash 1054
Translational Clutch 1055
Translational Damper 1056
Translational Friction 1057
Translational Friction (Controlled) 1059
Translational Hard Stop 1060
Translational Model Settings 1062
Translational Multiplexer 1063
Translational Port 1064
Translational Reference 1065
Translational Selector 1066
Translational Speed (Constant) 1067
Translational Speed (Controlled) 1068

Translational Speed Sensor 1069
Translational Spring 1070
Transmission Line (3ph) 1071
Transport Delay 1078
TRIAC 1080
Triangular Wave Generator 1082
Trigger 1083
Trigonometric Function 1085
Triple Active Bridge 1086
Triple Switch 1090
Turn-on Delay 1091
Variable Capacitor 1092
Variable Frequency PWM 1096
Variable Inductor 1099
Variable Magnetic Permeance 1103
Variable Phase PWM 1105
Variable Resistor 1109
Variable Resistor with Constant Capacitor 1110
Variable Resistor with Constant Inductor 1111
Variable Resistor with Variable Capacitor 1112
Variable Resistor with Variable Inductor 1114
Vienna Rectifier 1116
Voltage Source (Controlled) 1119
Voltage Source AC 1120
Voltage Source AC (3-Phase) 1121
Voltage Source DC 1122
Voltmeter 1123
White Noise 1124
Winding 1126
Wire Multiplexer 1128

Wire Selector 1129
XY Plot 1130
Zener Diode 1132
Zero Order Hold 1133

17 Simulink Blocks **1135**
Adapted Blocks 1136
Simulink-Only Blocks 1137
AC Sweep 1138
Impulse Response Analysis 1141
Loop Gain Analysis (AC Sweep) 1143
Loop Gain Analysis (Multitone) 1144
Multitone Analysis 1145
Steady-State Analysis 1147
Timer 1149

Before You Begin

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 extract the package `plecs-standalone-x-y-z_linux64.tar.gz` by entering the command

   ```
   tar xf 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 **Install license...** if you already have a PLECS license. A file dialog opens where you can select your license file.
- Choose **Get trial or student license...** to request a time-limited trial or student license.
- Choose **Run in demo mode** to use PLECS in a restricted demo mode that lets you build models and run simulations. Saving models or data is disabled in this mode.

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.

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 keep an existing license file, copy a new license file from your harddisk or request a trial or 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 keep an existing license file, copy a new license file from your harddisk or request a trial or 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 extract the package `plecs-blockset-x-y-z_linux64.tar.gz` by entering the command

```
tar xf 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 keep an existing license file, copy a new license file from your harddisk or request a trial or 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 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. Saving Simulink models containing PLECS blocks is disabled in this mode.

  ↪ *Added in PLECS 4.5.*
- 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.

Without a valid license you will still be able to open or save Simulink models containing PLECS blocks. However, you cannot modify a circuit schematic 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” (p. 60).

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.

License File Location

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

Table 1: License File Search Paths

Platform	Search Paths
Windows	`C:\Users\<USER>\AppData\Local\Plexim\PLECS\licenses` `C:\ProgramData\Plexim\PLECS\licenses`
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 5.0

Major New Features

- PLECS Spice is a separately licensed add-on for PLECS Standalone that allows you to import SPICE netlists into a PLECS model and connect them to other components in a schematic. Read more in the chapter "SPICE" (p. 387).
- The external mode of the PLECS Coder supports the Nanostep Scope introduced in the RT Box Target Support Package version 4.0, which enables you to monitor internal voltages and currents in Nanostep blocks with Nanostep time resolution.
- The Nanostep library contains new blocks for modeling on-board chargers, such as the Bidirectional PSFB Converter and various single-stage topologies with matrix converter frontends. These blocks support real-time simulations with step sizes as small as $4\,\mathrm{ns}$ on PLECS RT Boxes with firmware version 4.0 or newer. For a full list see "Nanostep" (p. 493) in the chapter "Components by Category" (p. 475).
- The Functional Mockup Interface of PLECS allows you to export and import Model Exchange and Co-Simulation Functional Mockup Units conforming to the FMI standard versions 2.0 and 3.0. See "Functional Mockup Interface" (p. 365) for more information.
- The Frequency Response Analysis is a robust general-purpose small-signal analysis type that works with plain time-domain simulations. See "Frequency Response" (p. 235) for details.

Further Enhancements

- The PLECS Standalone solvers by default use an analytical method to calculate the Jacobian matrix of a model for enhanced performance during the simulation of stiff systems. See "Variable-Step Solver Options" (p. 136).
- PLECS validates parameter expressions as you enter them into dialog boxes and flags both syntax errors and semantic errors (such as entering a negative number where only positive values are allowed).
- PLECS shows the evaluated parameter values as tooltips both in dialog boxes and for parameters shown on the schematic.
- You can visualize vectorized connections and optionally show their widths in a schematic, see the section "Schematic" (p. 61) in "Configuring PLECS" (p. 60).
- The Scope, Multiplexer and Demultiplexer blocks can be stretched vertically to facilitate the creation of cleaner schematics.

- The **Curve Properties** editor of the scope allows you to configure markers for visualizing the time steps of a simulation, see "Changing Curve Properties" (p. 121).
- Pressing the **Enter** key in the schematic editor opens the parameter dialog of a selected component, see "Specifying Component Parameters" (p. 67).
- If a schematic is larger than the window, you can press the middle mouse button (or the scroll wheel) while dragging the mouse for panning.

1

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 Standalone. If you are using PLECS Blockset for Simulink, please continue with section “Getting Started with PLECS Blockset” (p. 22).

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. See also: “Using PLECS Standalone” (p. 56).

A Simple Passive Network

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.

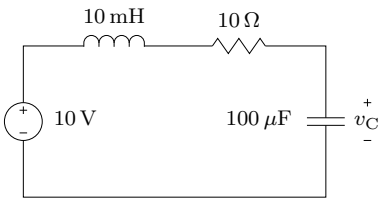


Fig. 1.1: Simple RLC network

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** (on macOS: **Cmd**) key while dragging them.

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

the component you need in the search bar. For example, typing `sc` shows you the scope, `res` all available resistors etc.

After you have copied all components the schematic window should look like Fig. 1.2. If not, move the components with the left mouse button. To rotate selected components press **Ctrl+R** (on macOS: **Cmd+R**), to flip them horizontally press **Ctrl+F** (on macOS: **Cmd+F**). All these functions can also be accessed via the menu bar.

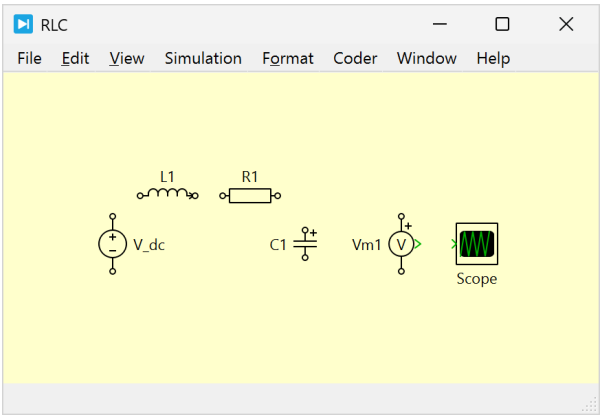


Fig. 1.2: PLECS schematic

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** (on macOS: **Cmd**) key you can create a connection from there to the desired destination.

Component Properties

Each component is identified by a unique name, which is chosen automatically. You may change it as you wish by double-clicking on it in the schematic. The name is intended only for documentation purposes and does not affect the simulation. Of greater importance are the parameters that determine, for example, the inductance of an inductor, the capacity of an capacitor, or the voltage of a DC voltage source. A double-click on the component icon opens a dialog box in which you can set these parameters. Fig. 1.3 shows the dialog box for an inductor.

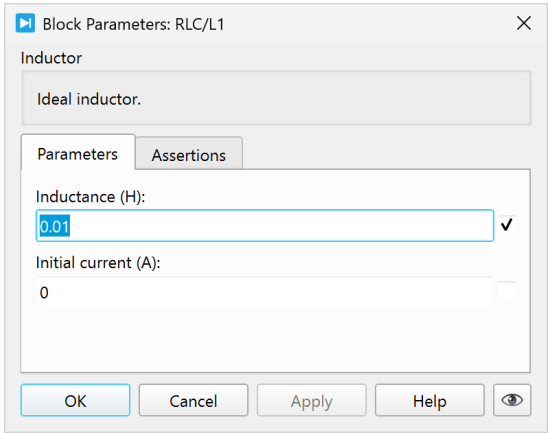


Fig. 1.3: Inductor dialog box

If you want selected parameters to be displayed in the schematic, 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. 1.1 on p. 14. 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. 1.4. To start the simulation, press **Ctrl+T** (on macOS: **Cmd+T**) or select "Start" from the "Simulation" menu. In order to see the more interesting part of the simulation, you need to set the time span 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** (on macOS: **Cmd+E**).

You should now get the simulation results shown in below.

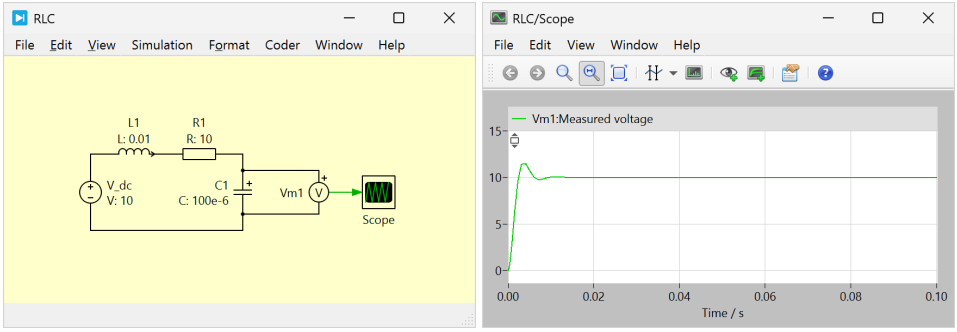


Fig. 1.4: Complete model and simulation result

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. 1.5 we

used a pulse generator with a period of $0.04\,\mathrm{s}$ and an amplitude of 10 to control the voltage source.

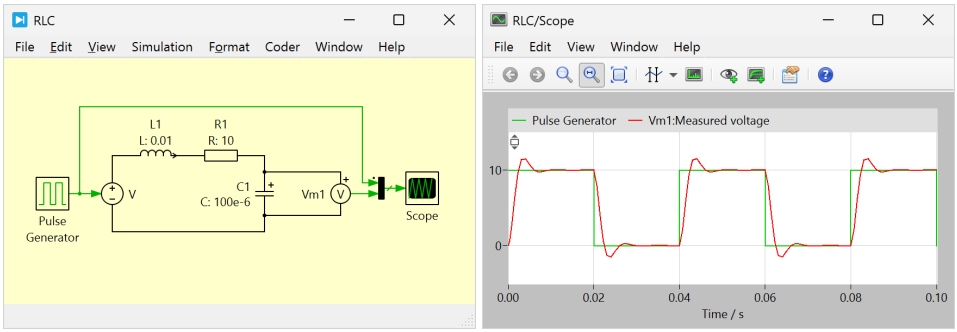


Fig. 1.5: RLC network with a pulsed voltage source

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.

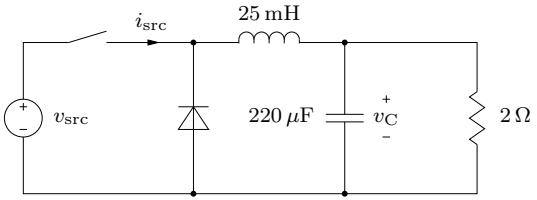


Fig. 1.6: Schematic of buck converter

Switches

In the buck converter outlined in Fig. 1.6 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.

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. 1.7.

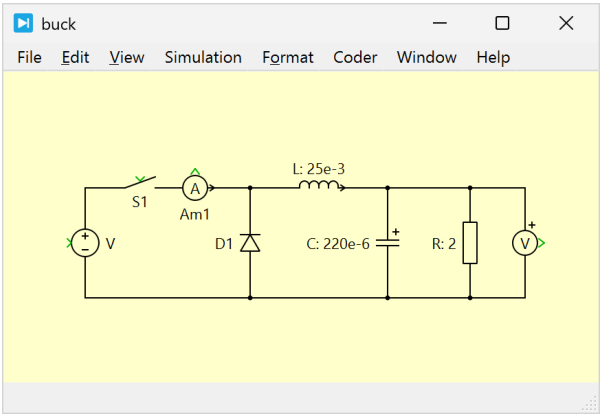


Fig. 1.7: Electrical part of buck converter

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. 1.7, 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** (on macOS: **Cmd+A**)). Now create a new subsystem by selecting “Create Subsystem” from the “Edit” menu or by pressing **Ctrl+G** (on macOS: **Cmd+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 dragging it with the middle mouse button to the borders or the corners of the icon.

Alternatively, use the left mouse button while holding the **Shift** key. Now your system should look similar to Fig. 1.8.

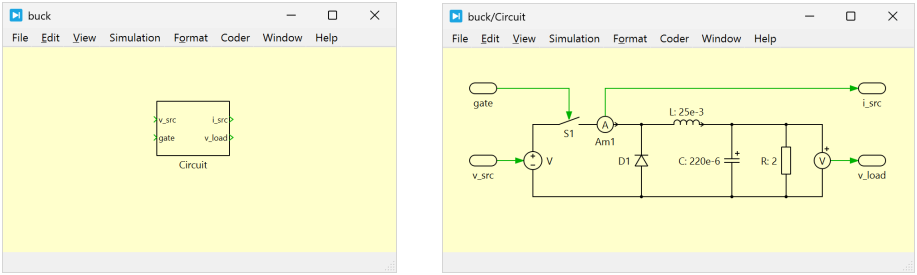


Fig. 1.8: Electrical Subsystem

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 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 $\pm 0.2\,\mathrm{V}$ band around $6\,\mathrm{V}$. To make things a bit more interesting we apply a step change from $12\,\mathrm{V}$ down to $8\,\mathrm{V}$ to the input voltage during the simulation.

Your simulation should now reproduce the results shown in Fig. 1.9.

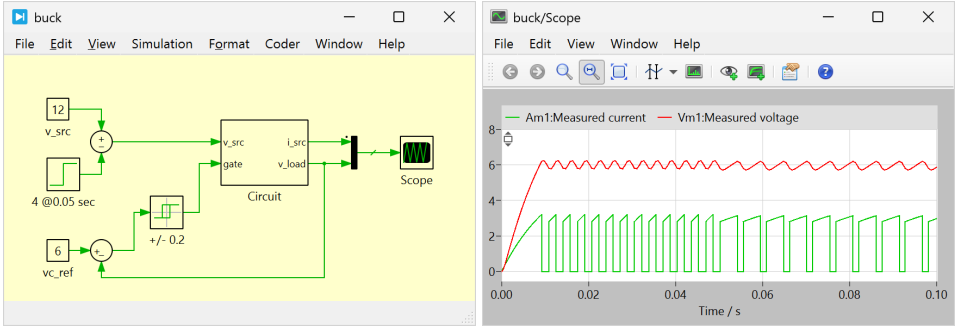


Fig. 1.9: Simulation of buck converter with hysteresis control

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.

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. See also: "Using PLECS Blockset" (p. 57).

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. 1.10. A capacitor is charged by a DC voltage source via an RL-branch and its voltage is monitored with a voltmeter.

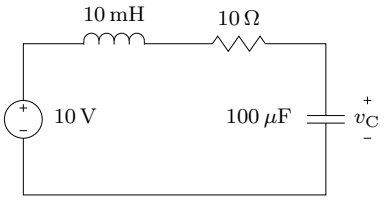


Fig. 1.10: Simple RLC network

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. 1.11.

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** (on macOS: **Cmd**) key while dragging the mouse. The components that you

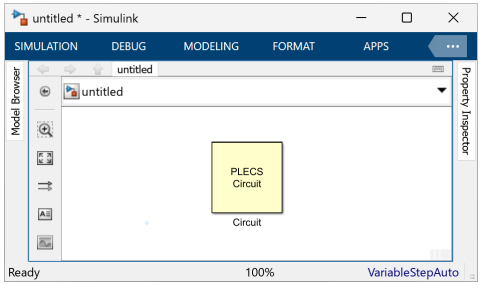


Fig. 1.11: Simulink model

need for the RLC network can be found in 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. 1.12. If not, move the components with the left mouse button. To rotate selected components press **Ctrl+R** (on macOS: **Cmd+R**), to flip them horizontally press **Ctrl+F** (on macOS: **Cmd+F**). All these functions can also be accessed via the menu bar.

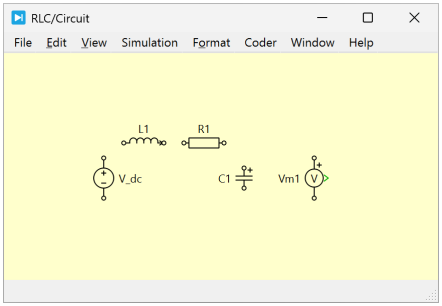


Fig. 1.12: PLECS schematic

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** (on macOS: **Cmd**) key you can create a connection from there to the desired destination.

Component Properties

Each component is identified by a unique name, which is chosen automatically. You may change it as you wish by double-clicking on it in the schematic. The name is intended only for documentation purposes and does not affect the simulation. Of greater importance are the parameters that determine, for example, the inductance of an inductor, the capacity of an capacitor, or the voltage of a DC voltage source. A double-click on the component icon opens a dialog box in which you can set these parameters. Fig. 1.13 shows the dialog box for an inductor.

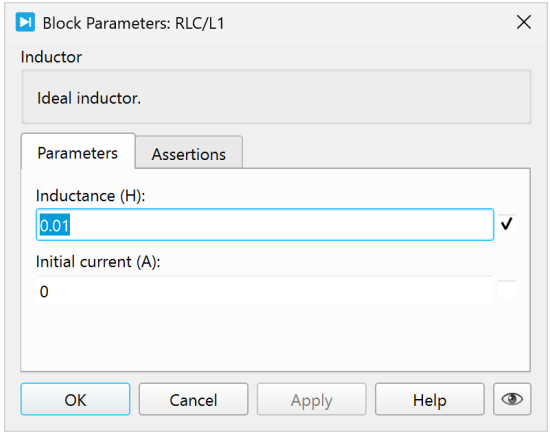


Fig. 1.13: Inductor dialog box

If you want selected parameters to be displayed in the schematic, 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. 1.10 on p. 22. In the dialog boxes of the inductor and the capacitor you can additionally set the initial current resp. the initial voltage. Please leave both values at zero.

Signals

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

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

Connect the output of the voltmeter with the input of the port block. In Simulink, connect a Scope to the output of the PLECS block and start the simulation. In order to see 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. 1.14 and Fig. 1.15 on your screen.

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".

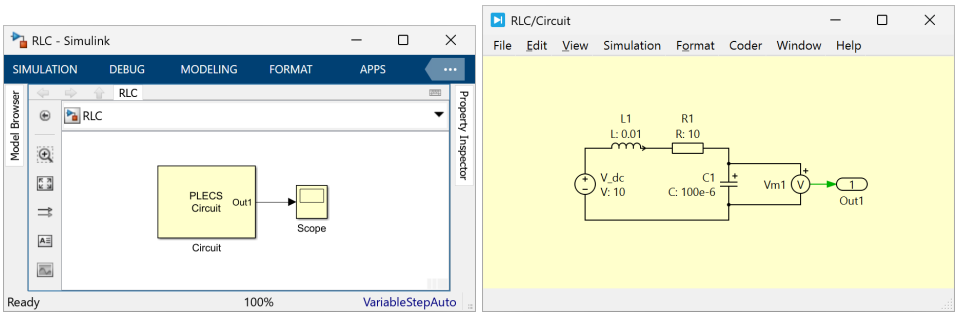


Fig. 1.14: Complete model

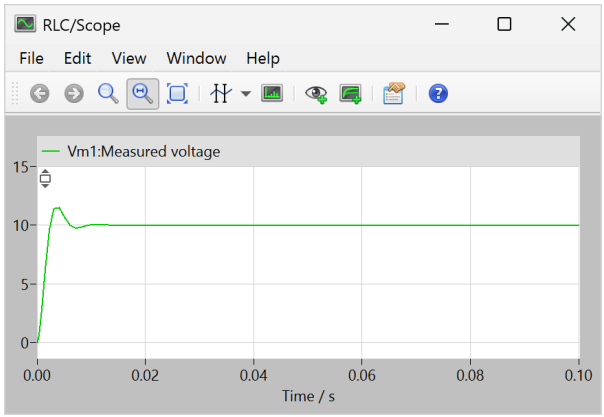


Fig. 1.15: Simulation result

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. 1.16 we used a pulse generator with a period of $0.04\,\mathrm{s}$ and an amplitude of 10.

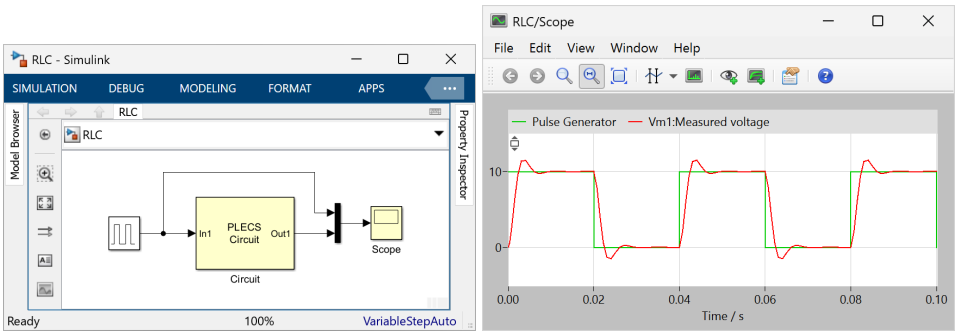


Fig. 1.16: RLC network with a pulsed voltage source

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. 1.17).

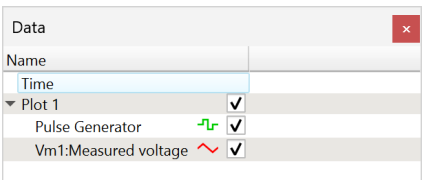


Fig. 1.17: Data window of the PLECS Scope

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.

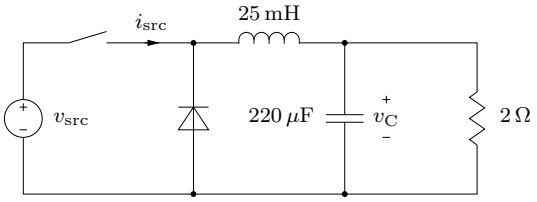


Fig. 1.18: Schematic of buck converter

Switches

In the buck converter outlined in Fig. 1.18 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.

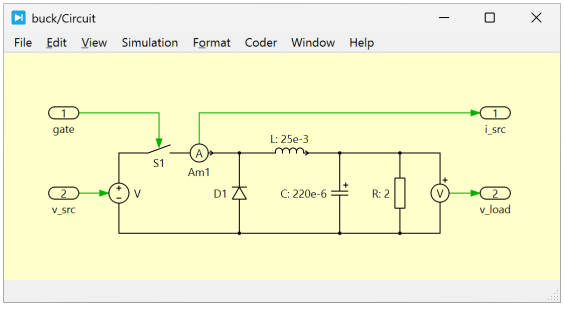


Fig. 1.19: Electrical part of buck converter

By now you should be able to model the electrical part of the buck converter as

shown in Fig. 1.19. For the buck converter we will implement a hysteresis type control that keeps the capacitor voltage roughly in a $\pm 0.2\,\mathrm{V}$ band around $6\,\mathrm{V}$. To make things a bit more interesting we apply a step change from $12\,\mathrm{V}$ down to $8\,\mathrm{V}$ to the input voltage during the simulation.

Your simulation should now reproduce the results shown in Fig. 1.20.

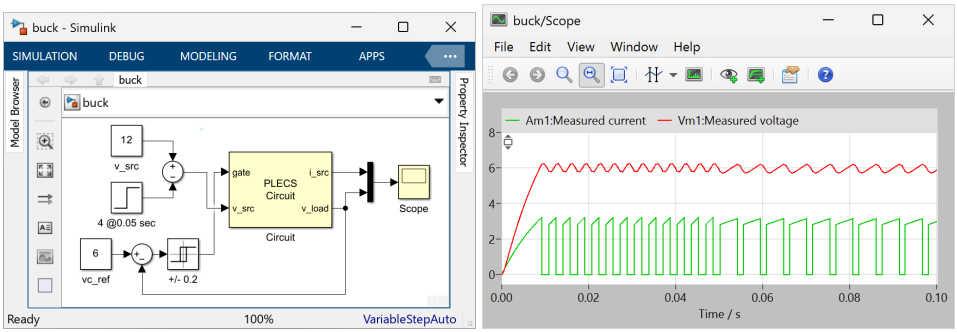


Fig. 1.20: Simulation of buck converter with hysteresis control

Demo Models

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

2

How PLECS Works

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

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

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

Modeling Dynamic Systems

A system can be thought of as a black box as depicted in Fig. 2.1. 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.

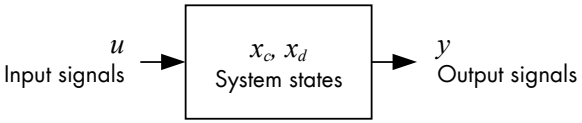


Fig. 2.1: Example of a dynamic system

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 in Fig. 2.2, which shows a low pass filter.

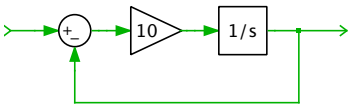


Fig. 2.2: Block diagram implementation of a low pass filter

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.

From the **Simulation** menu, you can run a complete simulation, via **Start**, or the initialization phase only, via **Evaluate Parameters**.

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 piece-wise linear state-space equations:

$$\dot{\mathbf{x}} = \mathbf{A}_\sigma \mathbf{x} + \mathbf{B}_\sigma \mathbf{u}$$
$$\mathbf{y} = \mathbf{C}_\sigma \mathbf{x} + \mathbf{D}_\sigma \mathbf{u}$$

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

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

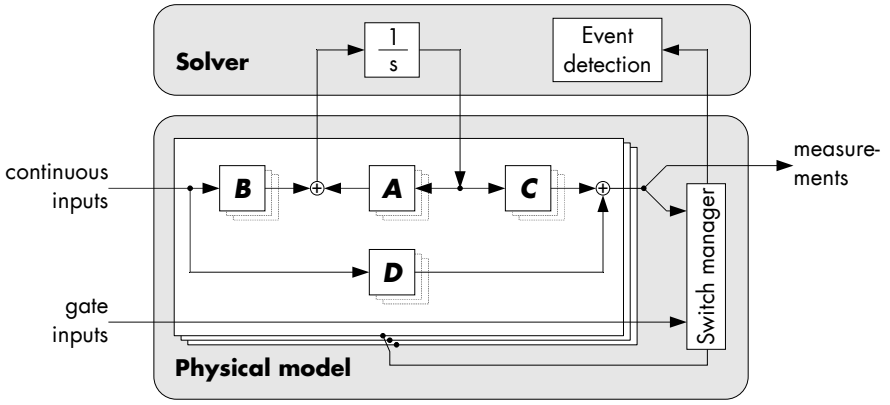


Fig. 2.3: Switched state-space implementation

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 σ is encountered repeatedly in this process, PLECS is unable to determine stable conditions and aborts the simulation.

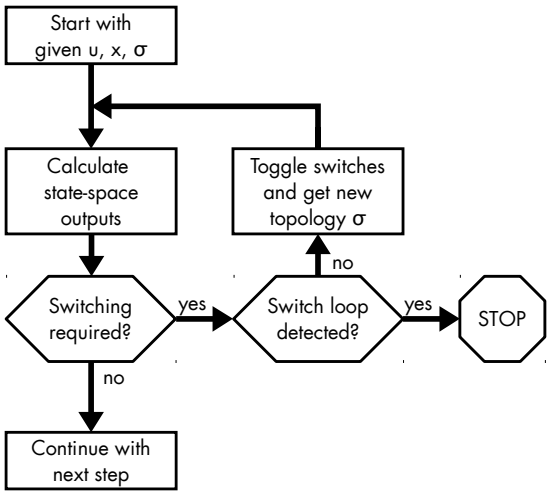


Fig. 2.4: Switch Manager flowchart

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

↪ *Added in PLECS 4.0.*

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

↪ *Added in PLECS 4.1.*

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” (p. 135) for a list of parameters that influence the solution of algebraic loops.

The block Initial Condition (p. 782) can be used to provide a guess to the equation solver at the start of a simulation.

Model Execution

Fig. 2.5 illustrates the workflow of the actual simulation.

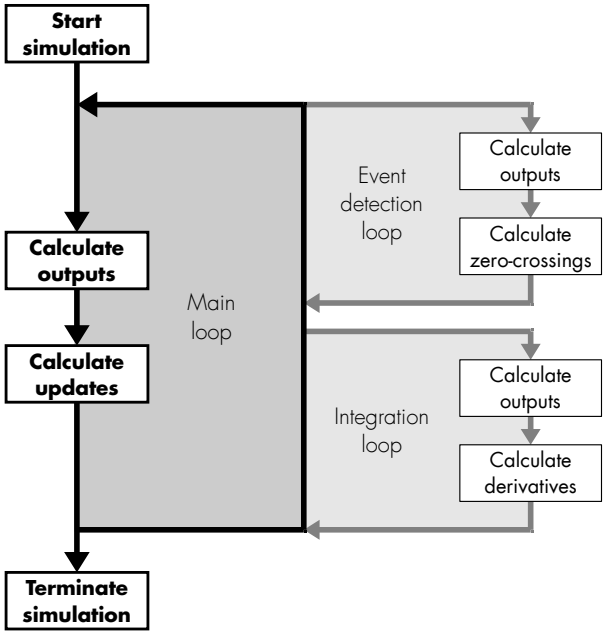


Fig. 2.5: Simulation loop

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.

1 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.

2 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.

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:

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

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 Current Source (Controlled) (p. 632) and the Voltage Source (Controlled) (p. 1119) 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 $\mathbf{A}_\mathrm{d}$, $\mathbf{B}_{\mathrm{d}1}$ and $\mathbf{B}_{\mathrm{d}2}$ are calculated from the continuous matrices $\mathbf{A}$ and $\mathbf{B}$ using a fifth-order accurate fully-implicit three-stage Runge-Kutta formula (Radau IIA). 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 $\mathbf{B}_{\mathrm{d}1}$ equals $\mathbf{B}_{\mathrm{d}2}$. 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 (p. 135).

↪ *Changed in PLECS 4.1:* Introduced BI45 as alternative to Tustin.

↪ *Changed in PLECS 4.4:* Replaced BI45 by Radau IIA.

Note that this applies only to the discretization of physical domains. The state variables 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).

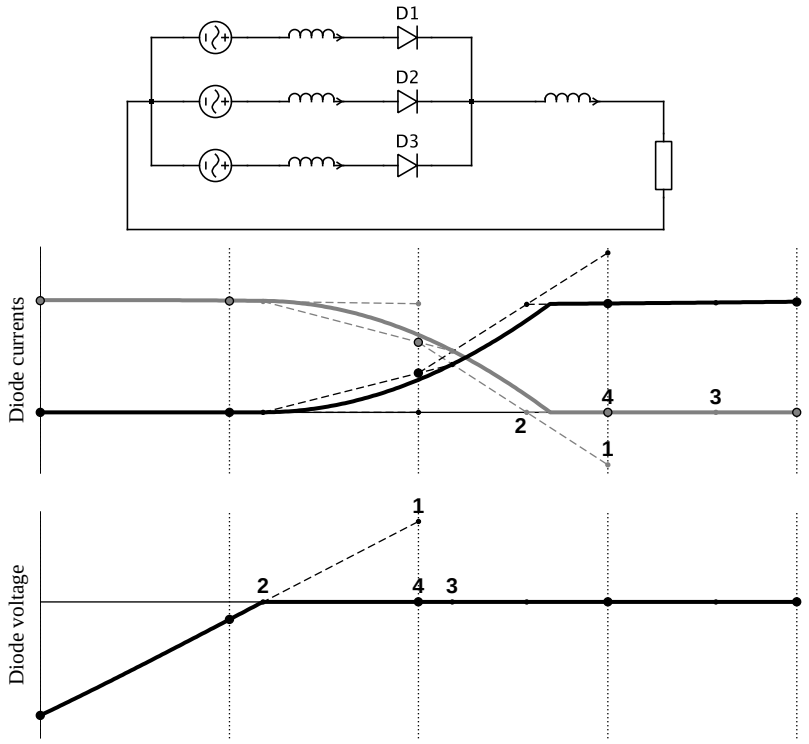


Fig. 2.6: Interpolation of non-sampled switching events

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. Some blocks may have more than one sample time, and they may associate assign different sample times to different input or output terminals. 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 (p. 818), 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 (p. 716), Sum (p. 980) and Product (p. 892).

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 (p. 621). 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 (p. 594) 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.

Table 2.1: Sample Time Parameter Values

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`	
Inherited	`[-1, 0]` or `-1`	
Constant	`[inf, 0]` or `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" (p. 36)). 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.

Continuous Sample Time Conflicts

↪ *Added in PLECS 4.8.*

A continuous sample time conflict arises when a continuous signal is fed into a block that expects a discrete input or when a discrete signal is fed into a block that expects a continuous input. Such a situation typically indicates a modeling error that produces undesirable results.

Example 1

A user attempts to break an algebraic loop by inserting a Memory block (p. 818). This is problematic if the loop involves a continuous-time signal because Memory (p. 818) is a *discrete* block and converts the continuous signal into a piece-wise constant signal. It also introduces a variable-step delay when used with a variable-step solver. In order to mitigate the negative effects of this, the user may be tempted to limit the maximum step size of the variable-step solver or switch to the fixed-step solver with a small time step, which leads to very long simulation times.

The proper way to break a continuous algebraic loop is by inserting a continuous low pass filter.

Example 2

A user attempts to produce phase-shifted gate signals for an interleaved converter with a Transport Delay block (p. 1078). This is problematic because Transport Delay (p. 1078) is a *continuous* block and produces the delayed output signal by *interpolating* between past samples of the input signal. A rectangular input signal is thus converted into a trapezoidal signal where the slope depends on the solver step size. Again, they user may be tempted to limit the maximum step size of the variable-step solver or switch to the fixed-step solver with a small time step.

The proper block to delay a rectangular or any other discrete signal is the Pulse Delay (p. 893).

PLECS will detect such continuous sample time conflicts and flag an appropriate diagnostic warning or error. This is controlled with the solver option **Continuous sample time conflict** on the **Diagnostics** tab of the Simulation Parameters dialog (p. 135).

Multirate Systems

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

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

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

Troubleshooting

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

1. 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 $30\,\mathrm{kHz}$ enter `1/30e3` instead of `3.3333e-5`.

2. 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.

3 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.

Data Types

↪ *Added in PLECS 4.2:* Introduced for code generation.

↪ *Added in PLECS 4.4:* Introduced for normal simulations.

PLECS can use different data types to store the value of a signal. Boolean, integer and floating-point data types are listed in the table below. Additionally fixed-point data types are supported (see next section).

Table 2.2: 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
`target`	A pseudo data type that resolves to single-precision or double-precision floating point depending on the simulation target. For normal simulations and when using the PLECS Blockset Coder, this type resolves to `double`. When using the PLECS Standalone Coder, the type is determined by the parameter **Floating point format** in the **Coder Options** dialog (see “Generating Code” (p. 348)).

Fixed-Point Data Types

↪ *Added in PLECS 4.8.*

PLECS supports signed and unsigned fixed-point numbers with a word length of 8, 16 or 32 bits. The scaling and interpretation of the fixed-point value is defined by the number of fractional bits. The results of arithmetic operations are

rounded to the closest representable number in direction of negative infinity (two's complement truncation).

When you select a **Fixed-point number** data type, additional control elements appear. The signedness and word length are set by selecting the base data type from the combo box. The number of fractional bits can be specified with the spin box. The resulting range and precision of the current settings are displayed underneath.

A fixed-point data type can also be passed with a reference variable by specifying its properties as a structure. E.g. `struct('Signed', 1, 'WordLength', 32, 'FractionLength', 16)` specifies a signed 32-bit fixed-point data type with 16 fractional bits.

Setting the global model parameter **Use floating-point data type for fixed-point signals** (see "Simulation Parameters" (p. 135)) overwrites all fixed-point data types with the target floating-point data type. This can be used to easily compare the accuracy of fixed-point operations with floating-point operations.

Blocks with an Implicit Output Data Type

Certain blocks have an implicit output data type. For instance, all physical meters use the `target` output type, and a logical operator always outputs a Boolean signal. Other blocks with implicit output data types are listed below:

- Comparator (p. 618), `bool`
- Compare to Constant (p. 619), `bool`
- Edge Detection (p. 670), `bool`
- FMU (p. 686), according to the FMU description
- Hit Crossing (p. 739), `bool`
- Logical Operator (p. 793), `bool`
- Monoflop (p. 826), `bool`
- Relational Operator (p. 903), `bool`
- Signum (p. 956), `int32_t`
- Trigger (p. 1083), `bool`

Specifying an Output Data Type

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

- Constant (p. 621)

- Data Type (p. 635)
- Enable (p. 678)
- Gain (p. 716)
- Offset (p. 855)
- Product (p. 892)
- Pulse Generator (p. 895)
- Relay (p. 904)
- Rounding (p. 925)
- Signal Inport (p. 949)
- Step (p. 976)
- Sum (p. 980)

Some of these blocks also let you choose to inherit the output data type from the input signal(s).

Data Type Inheritance

The following blocks implicitly inherit the output data type from the input signals:

- Abs (p. 554), the unsigned input data type is set as output data type
- Delay (p. 639)
- Dynamic Signal Selector (p. 669)
- Function (p. 715)
- Initial Condition (p. 782)
- Manual Signal Switch (p. 801)
- Memory (p. 818)
- Minimum / Maximum (p. 820)
- Multiport Signal Switch (p. 838)
- Saturation (p. 936)
- Signal Switch (p. 955)
- Zero Order Hold (p. 1133)

If the input signals have heterogeneous data types, an error message is generated, unless there is at least one floating-point data type and no fixed-point data type. In this case the floating-point data type is inherited. To explicitly set the data type of one of the above blocks, insert a Data Type block (p. 635) in front of the block.

Earlier versions of PLECS implemented more relaxed data type inheritance rules by choosing the data type with the greatest order (as defined in the table above). These rules can be applied to legacy models by changing the global model parameter **Datatype inheritance conflict** (see "Simulation Parameters" (p. 135)).

Data Type Overflows

↪ *Added in PLECS 4.8.*

Data type overflows are handled individually per block, according to the following parameter:

Data type overflow handling

This parameter applies to simulations and generated code. The options are:

- **Unchecked**
  Overflows are ignored, the resulting value is dependent on the platform and the compiler used. The most efficient code is generated with this option.
- **Saturate**
  Overflows/underflows are clamped to the maximum/minimum representable value of the data type.
- **Assert with error**
  The simulation or program execution is aborted with an error if an overflow is detected.

With the global model parameter **Datatype overflow** (see "Simulation Parameters" (p. 135)), warning/error messages on data type overflow can be enabled independent from the individual block settings. This parameter only affects simulations and not code generation.

Conventions & Definitions

Three-Phase Sequence Convention

For three-phase electrical systems (e.g., Voltage Source AC (3-Phase) (p. 1121)), PLECS adheres to the positive sequence and counter-clockwise rotation conventions of the phases. Thus, phase b lags phase a by $120°$, and phase c leads phase a by $120°$.

$$x_a = X_a \cos(\omega t + \varphi)$$

$$x_b = X_b \cos(\omega t + \varphi - \frac{2\pi}{3})$$

$$x_c = X_c \cos(\omega t + \varphi + \frac{2\pi}{3})$$

where, x_i represents the instantaneous voltage/current, X_i represents the peak value of the voltage/current, ω is the angular frequency in radians per second, t is the simulation time, and φ is the phase shift in radians.

The phasor diagram of a three-phase electrical system is given in Fig. 2.7

Reference Frame Transformations

With regards to the reference frame transformation components:

- Transformation 3ph->RRF (p. 1040)
- Transformation 3ph->SRF (p. 1041)
- Transformation RRF->3ph (p. 1042)
- Transformation RRF->SRF (p. 1043)
- Transformation SRF->3ph (p. 1044)
- Transformation SRF->RRF (p. 1045)

the stationary $[x_\alpha \ x_\beta]$ and the rotating $[x_\mathrm{d} \ x_\mathrm{q}]$ reference frame axes are aligned and rotating according to Fig. 2.8 on p. 54, with respect to the three-phase system.

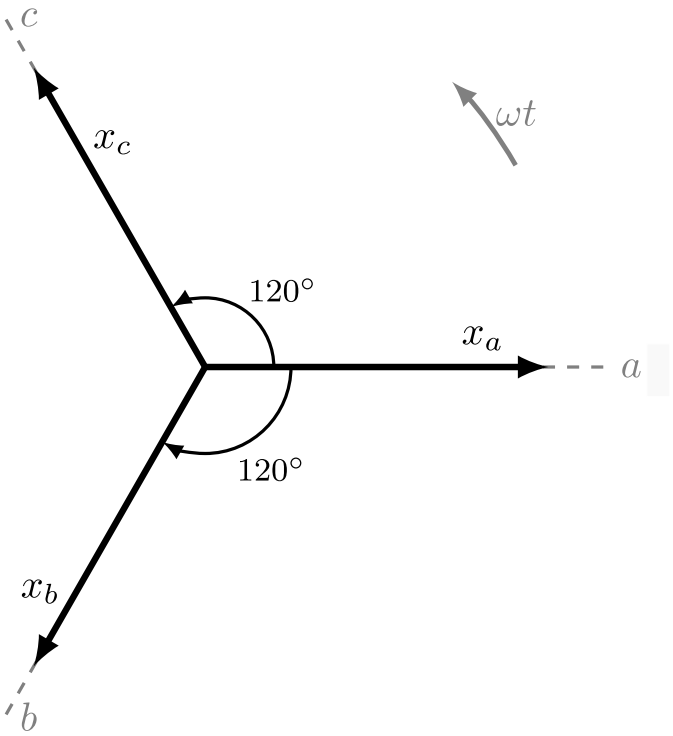


Fig. 2.7: Phasor diagram of a three-phase system.

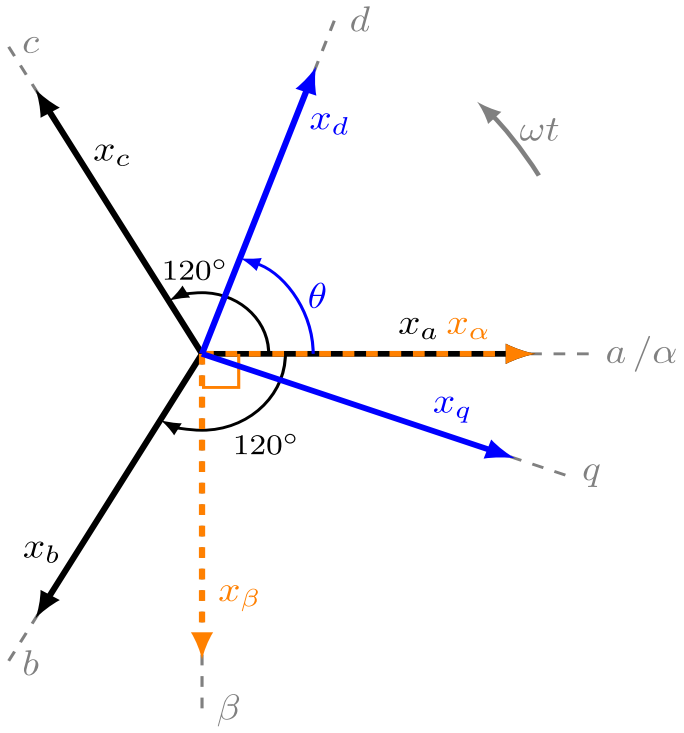


Fig. 2.8: Graphical representation of stationary and rotating reference frame.

3

Using PLECS

The main user interface of PLECS is the graphical schematic editor in which you create block diagrams and physical models. This chapter describes the aspects of the schematic editor that are common to PLECS Standalone and PLECS Blockset but also the differences between the two that are highlighted in the first two sections.

Using PLECS Standalone

Creating a New Model

To create a new model, choose **New Model** from the **File** menu. It is good practice to save the new model before you make any changes in order to enable the auto-save functionality.

Importing a Schematic From PLECS Blockset

To facilitate data exchange between PLECS Standalone and PLECS Blockset, you can import the schematics of PLECS Circuit blocks inside a Simulink model. To do so, choose **Import form Blockset...** from the **File** menu.

> **Note**
>
> PLECS cannot import the Simulink blocks within a Simulink model because there is no exact match between Simulink blocks and PLECS components.

Using PLECS Blockset

The main element of PLECS Blockset is the **PLECS Circuit** block in the PLECS Library. This block constitutes the interface between the PLECS simulation engine and the Simulink solver.

To open the PLECS library, type `plecslib` at the MATLAB command prompt or click on the **Library Browser** icon in a Simulink window and click on **PLECS** in the **Simulink Library Browser** window.

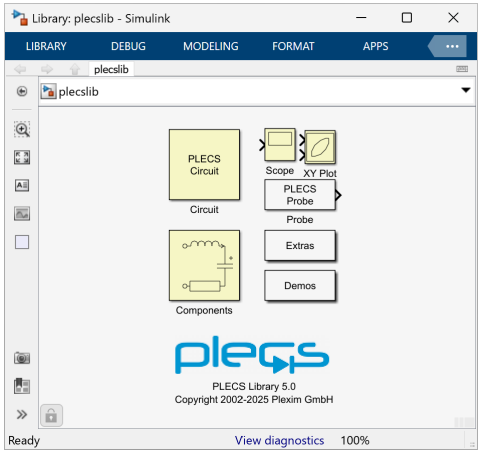


Fig. 3.1: The PLECS library in Simulink

Creating a New Circuit

To create a new circuit, copy the **PLECS Circuit** block from the PLECS library into your Simulink model, then double-click the block to open the schematic editor.

> **Note**
>
> You cannot place Simulink blocks in a PLECS schematic or PLECS components in a Simulink model because the two programs do not share the same Graphical User Interface.

Importing a Schematic From PLECS Standalone

To facilitate data exchange between PLECS Blockset and PLECS Standalone, you can import the schematics of a PLECS Standalone model into a PLECS Blockset schematic. To do so, choose **Import from Standalone...** from the **File** menu.

Opening a PLECS Standalone Model

You can also directly open a PLECS Standalone model in the PLECS Blockset schematic editor. This is useful if you want to work with the Model Reference (p. 824) block.

To open a PLECS Standalone model, choose **Open...** from the **File** menu of a schematic editor or the PLECS Library Browser and change the file type filter in the file dialog to **PLECS Standalone model files (*.plecs)**. Note that the schematic editor of a PLECS Standalone model does not have a **Simulate** menu in PLECS Blockset because you cannot run a simulation without a Simulink model.

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” (p. 67).

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

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

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

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

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

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

Configuring PLECS

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

Note that various tooltips give additional information upon hovering.

General

Language

The language used by PLECS can be specified or set to `Auto`, such that PLECS uses the language settings of your computer. Available languages are `English` and `Japanese`. To activate the new language settings, PLECS must be restarted.

Appearance

This setting controls the color scheme used by PLECS. You can choose between the classic `light` scheme and the new `dark` scheme that is designed to work well in low-light conditions. On Windows and macOS, you can also let PLECS dynamically adapt to the system default of your computer by choosing the `auto` option. When PLECS is used in `dark` mode, the darkness of the schematic background can be scaled by the option **Contrast in dark schematic**.

↪ *Added in PLECS 4.8.*

System Settings

- PLECS saves your opened models regularly as an internal **Autosave**, both at the modifiable frequency and when a simulation is started. If PLECS quits unexpectedly, these autosaves can be restored: For models that have been saved before, an autosave can be loaded upon opening the model. For new unsaved models, an autosave can be loaded upon starting PLECS.

  ↪ *Added in PLECS 4.9.*

- In PLECS Standalone, the **Thread limit** specifies the maximum number of threads that may be used by an individual interactive analysis or an individual parallel simulation or analysis command. This can be further reduced using simulation or analysis options.

  ↪ *Added in PLECS 4.8.*

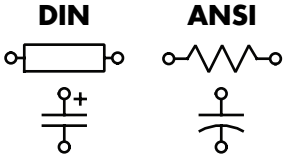
- In PLECS Standalone, the **RPC interface port** can be enabled or disabled for external scripting. When enabled, PLECS listens on the specified TCP port for incoming RPC connections. See chapter "RPC Interface in PLECS Standalone" (p. 323) for details on using the RPC interface.
- In PLECS Standalone, **Enable FMU Co-Simulation endpoint** controls whether PLECS can be used in a tool-coupling co-simulation (see section "PLECS Co-Simulation FMU" (p. 370)).

  ↪ *Added in PLECS 5.0.*

Schematic

Symbol Format

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



Grid

When set to `On`, a grid is displayed in the background of schematic windows for easier placement of components and their connections.

Vectorization

This setting sets the default vizualization of connection widths > 1. The default can be `Off`, `Markers` or `Numbers`. You can toggle whether the width numbers are shown any time while holding the **Tab** key. Note that the connection widths may not be up to date in your schematic, unless you run **Evaluate Parameters** or **Start** from the **Simulation** menu.

↪ *Added in PLECS 5.0.*

Zoom

This setting allows you to disable continuous zooming into the schematic via your mouse wheel. Discrete zooming via the **View** menu or the native keyboard shortcuts is always possible.

Component Snap

This feature allows you to snap components to useful positions while you

drag them across the schematic. It aligns component centers as well as terminals of matching domains and signal directions. It indicates the alignment during the drag by a dotted line.

↪ *Added in PLECS 4.9.*

This feature can be configured to match individual preferences:

- You can individually enable the **Elements** to be snapped to.
- You can decide whether snapping is enabled by **Default**.
- While holding a configurable **Modifier** key, snapping can be temporarily activated or deactivated.
- The **Jump** defines the maximum distance a dragged component will jump to align. The lower this value, the smaller the jumps.
- When limiting the **Reach**, only elements that lie within the specified range around the dragged one will be snapped to.

Window

Circuit Browser

This setting specifies the default filtering mode used for all circuit browsers (see section “Circuit Browser” (p. 131)).

Model Comparison

When comparing two models in a Diff Result window, by default the left view shows the “before” model and the right view the “after” model. With this setting, you can swap the views. Note that this only changes their location in the window and has no influence on the roles of the “before” and “after” models (see section “Diff Tool” (p. 375)).

↪ *Added in PLECS 4.7.*

Show on Startup

When starting PLECS, the **Library Browser** and **Welcome Screen** are shown if enabled.

↪ *Added in PLECS 4.8.*

OpenGL

On Windows, PLECS can use OpenGL for GPU-accelerated rendering, e.g. for the PLECS Documentation in the PLECS Help Viewer.

Crash Report

If PLECS crashes, a crash report dialog is shown, offering the possibility to send a report to Plexim to help investigate the cause of the crash. Some peripherals like the PLECS Help Viewer run in a separate process. You can disable the crash report dialog for this process.

↪ *Added in PLECS 4.7.*

Library

For details on creating custom libraries in PLECS, see “Libraries” (p. 72).

Search Path

In PLECS Standalone, the library search path can be modified in the **Search Path** table.

In PLECS Blockset, the custom libraries must be located on the MATLAB search path, which can be set from the MATLAB file menu.

User Libraries

To add custom libraries to the library browser, add these libraries to the **User Libraries** table. All custom libraries must be located on the library search path.

Links

If the checkbox **Show library link indicators** is checked, PLECS will display a small hollow curved arrow (⇱) in the lower left corner of each component that links back to a library. A right-click on this link indicator opens a context menu that allows you to break the library link or show the original component in the library browser.

↪ *Added in PLECS 4.5.*

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, see “Libraries” (p. 72).

Note

In PLECS Spice, the library search path is also used to find netlist files for the Netlist File Reference (p. 847).

Thermal

For more information about thermal modeling in PLECS, see “Thermal Modeling” (p. 171).

Thermal Description Search Path

This table contains the root directories of the thermal library. See section “Thermal Library” (p. 183) for more details.

Scope

For more information about the PLECS scope, see “Using the PLECS Scope” (p. 119).

Scope Background

This setting determines whether the PLECS scopes are drawn with a `Black` or `White` background. Choose the option `Auto` to dynamically adjust the scope background to the light or dark appearance of PLECS.

Scope Palette

- The **Palette** 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** table lists the base properties used for the curves in a scope plot. You can specify color, line style, line width, marker style, and marker size 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” (p. 123)). 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`, `Line width`, or the `Marker` appearance. The selected property will then not be available in the signal list, but in the **Traces** list. Again, if a plot has more traces than the number of entries in this list, PLECS will restart at the beginning.

  ↪ *Changed in PLECS 5.0:* Added customizable markers.

Scope Windows

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

Update

PLECS Software Updates

PLECS can be configured to **automatically 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

↪ *Added in PLECS 4.0.*

For more information about the PLECS Coder, see “Code Generation” (p. 341).

Target Support Packages Path

This setting 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.

Installed targets

This table 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 **Rescan** button.

Using the Library Browser

In PLECS Standalone it is opened automatically when the program is started. In PLECS Blockset the library browser is opened by a double-click on the Components block in the PLECS library. It can always be re-opened by choosing **Library Browser** in the **Window** menu or using the shortcut **Ctrl+L** (on macOS: **Cmd+L**).

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.

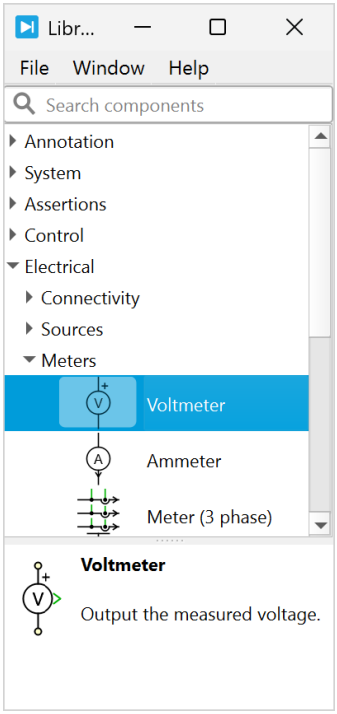


Fig. 3.2: The Library Browser

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

Components

Copying a Component into a Schematic

You can copy a new component into a schematic in different ways:

- Drag a component from the Library Browser into the schematic.
- Click into the schematic and start typing the type name of the desired component. This will open a mini browser in a popup window. To insert a component, use the up/down cursor keys and press **Enter**, or click and drag the component to the desired location. To close the mini browser without inserting a component, press the **Esc** key.

  ↪ *Added in PLECS 4.4.*
- Press and hold the **Ctrl** (on macOS: **Cmd**) key, then drag an existing component from the same or a different schematic.

Moving a Component

To move a component, click it with the left mouse button and drag it to the desired location. If the component has connected terminals, the connections are automatically rerouted to the new location so that the terminals remain connected. To cut any existing connections when moving a component, hold the **Shift** key while dragging the component.

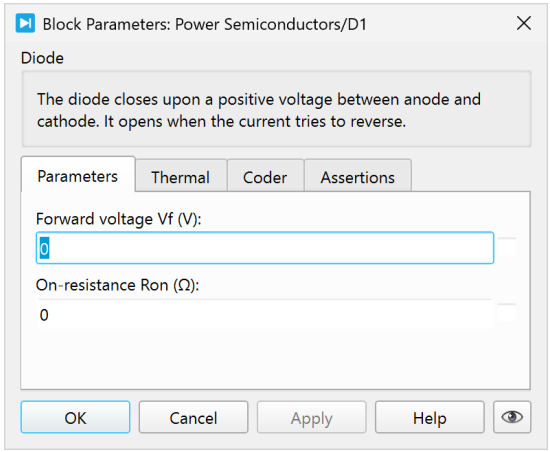
Specifying Component Parameters

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. You can also select the component and press the **Enter** key.

↪ *Changed in PLECS 5.0:* Added shortcut.

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 (p. 843) or the parameter **Width** of the Wire Multiplexer (p. 1128). 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 workspace variables. Expressions are evaluated as a whole in one workspace. By default, the evaluation workspace is the base Octave or MATLAB workspace. Notice that in PLECS Standalone every model has its own base Octave workspace that is populated by the model initialization commands that can be configured in the Simulation Parameters dialog, see “Model Initialization Commands” (p. 143).

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 Dialog” (p. 98).

In PLECS Blockset 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 instead of the MATLAB base 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” (p. 58).

Displaying Parameters in the Schematic

You can cause PLECS to display any component parameter beneath the component name in the schematic. To specify which parameter should be displayed in the schematic, open the dialog box and check the check box next to the parameter edit field. 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.

Parameters that have different values for the selected components 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

To edit a component name, double-click it in the schematic. Press **Enter** or click anywhere outside the label to finish editing. Press **Shift+Enter** to enter a line break for component names that span multiple lines. To show or hide a component name, toggle **Show name** in the **Format** menu.

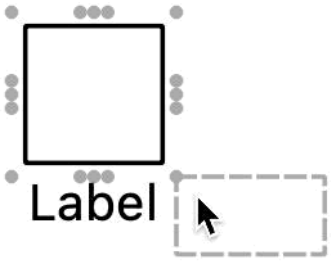
All component names in the same schematic must be unique and must contain at least one non-space character. Trailing spaces are removed from the names.

Placing the Component Label

↪ *Added in PLECS 4.7.*

The label of a component can be placed at any of the following 16 positions along the component frame: at the four corners, at the center of the four edges and at two off-center positions of each of the four edges. The label of a Subsystem block can additionally be placed at the center of the component frame.

To change the placement of a label, drag it with the middle mouse button to a new location. Alternatively, use the left mouse button while holding the **Shift** key. While you hold down the mouse button, small dots mark the possible positions, and a dashed rectangle indicates the new label position. When you release the mouse button, the label is moved. The horizontal and vertical alignment of the label text is automatically adjusted to the label position.



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

Disabling Components

↪ *Added in PLECS 4.7.*

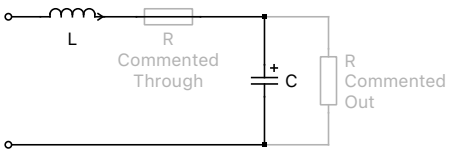
If you would like to disable one or more components temporarily so that they do not affect the behavior of a model, you can do so by *commenting* them. The term

"commenting" is inspired by the programming pattern of disabling lines of code by commenting them. There are two possibilities to comment a component:

Comment Out
: Commenting out a component has the same effect on the model as deleting the component, leaving the connections that lead to and from the component unconnected.

Comment Through
: For certain types of components it is useful to "close the gap" when they are commented. For instance, you may want to replace a commented resistor with a short circuit rather than an open circuit.

A commented component is drawn with dimmed colors, and the connections leading to and from it are also dimmed. If a component is commented through, solid lines indicate the terminals that are connected with each other, and the corresponding external connections are not dimmed. This is demonstrated in the figure below.



To comment or uncomment a component interactively, select it and toggle **Comment out** or **Comment through** in the **Edit** menu or the component's context menu. To comment or uncomment a component programmatically, use the command:

```
plecs('set', 'componentPath', 'CommentStatus', 'status')
```

where `status` is one of `CommentedOut`, `CommentedThrough` or `Active`.

Getting Component Help

Press the **Help** button in the dialog box to view the documentation for 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 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" (p. 60) 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 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" (p. 60) for details).

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 model. Afterwards, any changes that you make to a referenced library component are propagated to the referencing components when you start a simulation or when you manually update the simulation model with **Synchronize all external links...** from the **Edit** menu.

Breaking a Library Reference

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

In order to break the link between a reference and its library component, select the reference component, then choose **Break library link** from the **Edit > Subsystem** menu or the block'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 external 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** (on macOS: **Cmd**) key.

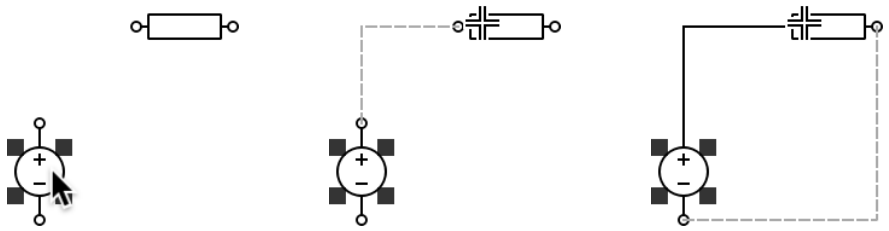
Alternatively, you can also create a branch by clicking on an unconnected terminal and dragging the mouse pointer to a matching connection or node.

Automatic Creation of Connections

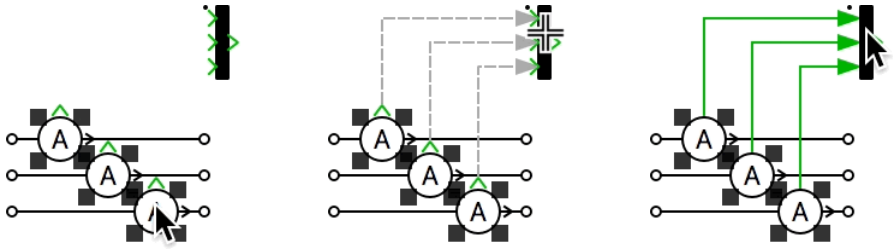
↪ *Added in PLECS 4.2.*

If you select a component and hold the **Alt** (on macOS: **Option**) 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** (on macOS: **Option**) 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 (p. 951). First, select the many components, then move the mouse pointer to the destination while holding down the **Alt** (on macOS: **Option**) 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.

Vectorization

Some models are based on a repetitive structure. Such structures can be implemented in a compact form with the concept of "Vectorization". For this purpose there is a De-/Multiplexer and Signal Selector component in the controls domain and a Wire Multiplexer and Wire Selector block in each each physical domain. These blocks allow to realize series and parallel connections of many components without actually having to place all components in the schematic. Besides this, source components can also output vector or bus signals. This way the circuit structure becomes parameter dependent.

Vectorized Sources

Source components can be vectorized by specifying a vector for one of the parameters. If a vector is specified for multiple parameters they all need to have the same length. For example the **Phase** parameter in the Sine Wave block (p. 960) allows a vector `[0, -120, 120]`.

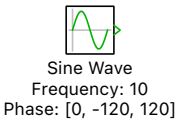


Fig. 3.5: Sine Wave block with vector output

Example Model

- See the example model **"Vector Sine Wave"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

In case of a physical domain the vectorized source components will lead to completely vectorized circuits, as in Fig. 3.6.

Example Model

- See the example model **"Vector Circuits"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

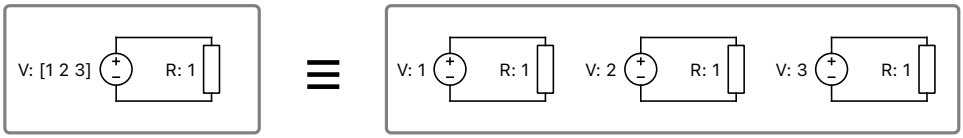


Fig. 3.6: Equivalence of a vector circuit with three scalar circuits

Signal De-/Multiplexer

In a basic application a signal multiplexer and demultiplexer are used to form and split bus signals. In Fig. 3.7 the sine wave block outputs a bus signal of width 3.

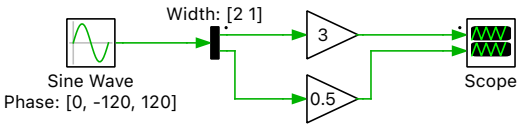


Fig. 3.7: Vector Sine Wave output demultiplexed

The Demultiplexer component is used to split the bus signal into a bus signal of width 2 and a scalar signal. The upper Gain multiplies each component of the bus signal with 3 and the lower Gain block simply multiplies the scalar signal with 0.5.

Example Model

- See the example model **"De-Multiplexer"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

Vectorizing Physical Components

Fig. 3.8 shows how a chain of RC elements can be built using wire multiplexers.

In a first step the original circuit is built with two wire multiplexer components of width 4. The first wire that enters the multiplexer leaves the second multiplexer on the first terminal (indicated with a dot). The same applies for the input signals 2 to 4. This is shown in Fig. 3.9

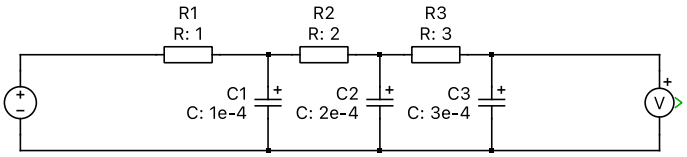


Fig. 3.8: Non-vectorized RC chain

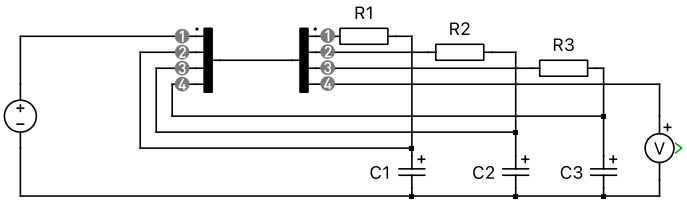


Fig. 3.9: RC chain built with a wire multiplexer

Because multiplexers can take vectors as **Width** parameter, the wires 2 to 4 can be combined. In Fig. 3.10 the two multiplexers use the value `[1 3]` and `[3 1]` for the **Width** parameter.

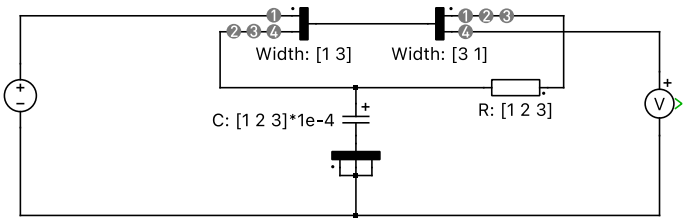


Fig. 3.10: Vectorized version of the RC chain

The vectorized implementation is a condensed form of the original circuit with only one resistor and capacitor component. By using a variable for the **Width** parameter (`[1 n]` and `[n 1]`), the length of the RC chain can be set from the model initialization commands. The values for the capacitances and resistances have to be a scalar or a vector with length of the RC chain.

Example Model

- The example model **"Step-by-Step Vectorization of RC Chain"** features an RC chain.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Example Model

- The example model **"Basic Vectorization Schemes"** features parallel and series connections of components in general.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Note

The input signal to a vectorized switching device of width `n` can be a scalar or a vector. If it is a scalar all `n` switching devices will have the same (gate) input signal. If the input signal is a vector of length `n` every element of the vector is the (gate) input signal to the corresponding switching device.

Annotations

You can annotate schematics with text labels and graphical elements, i.e. boxes, lines and arrows. Annotations are purely for documentation purposes and have no influence on the model behavior.

Text Annotations

To create a text annotation, double-click 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 a text annotation by selecting and dragging it with the mouse. To edit an existing annotation, double-click it. While you are editing an annotation, a toolbar is shown next to it allowing you to change the style, size, color and alignment of the text.

Box Annotations

↪ *Added in PLECS 4.6.*

To create a box annotation, double-click in an unoccupied area of a schematic and drag the mouse while holding the mouse button. To move a box annotation, click on its border and drag it to the desired location. To change the size of a box annotation, select it by clicking on its border, then drag one of its selection handles. To change the appearance of a box annotation, double-click on its border. A toolbar will be shown next to the box allowing you to change the border width, corner radius, border color and fill color of the box.

Line Annotations

↪ *Added in PLECS 4.6.*

To create a line annotation, press and hold the **Shift** key while double-clicking in an unoccupied area of the schematic and dragging the mouse. To move a line annotation, click on the line and drag it to the desired location. To change the orientation of a line annotation, select it by clicking on the line, then drag one of its selection handles. To change the appearance of a line annotation, double-click on the line. A toolbar will be shown next to the line allowing you to change the width, style and color of the line and the size and location of arrow heads.

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” (p. 84).

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 (p. 949), Signal Outport (p. 952) or Electrical Port (p. 676)) 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:

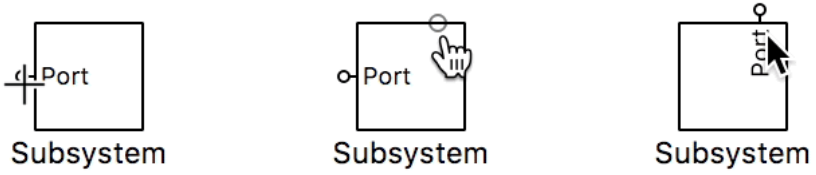
1. Select the blocks and connections that you want to include in the subsystem within a bounding box.
2. Choose **Create subsystem** from the **Edit** menu. PLECS replaces the selected blocks with a Subsystem block.

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. Alternatively, use the left mouse button while holding the **Shift** key. While you hold down the mouse button, a circle shows the free slot nearest to the mouse pointer. When you release the mouse button, the terminal is moved.

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

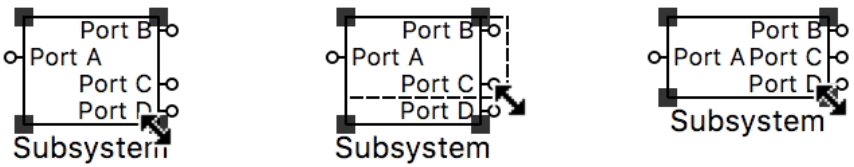


Notice how the shape of the cursor changes to crosshairs as you move it into the capture radius of the terminal. When you press and hold down the center mouse button, the cursor shape changes to a pointing hand.

Resizing a Subsystem Block

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

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



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

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 **Edit > Subsystem** 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.

Example Model

- The example model **"Subsystem Masks"** accompanies the explanations in this section.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

Mask Icon

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

Mask Icon Drawing Language

↪ *Added in PLECS 4.2.*

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 or if you would like to make use of the "PLECS Icon Library" (p. 92). 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.

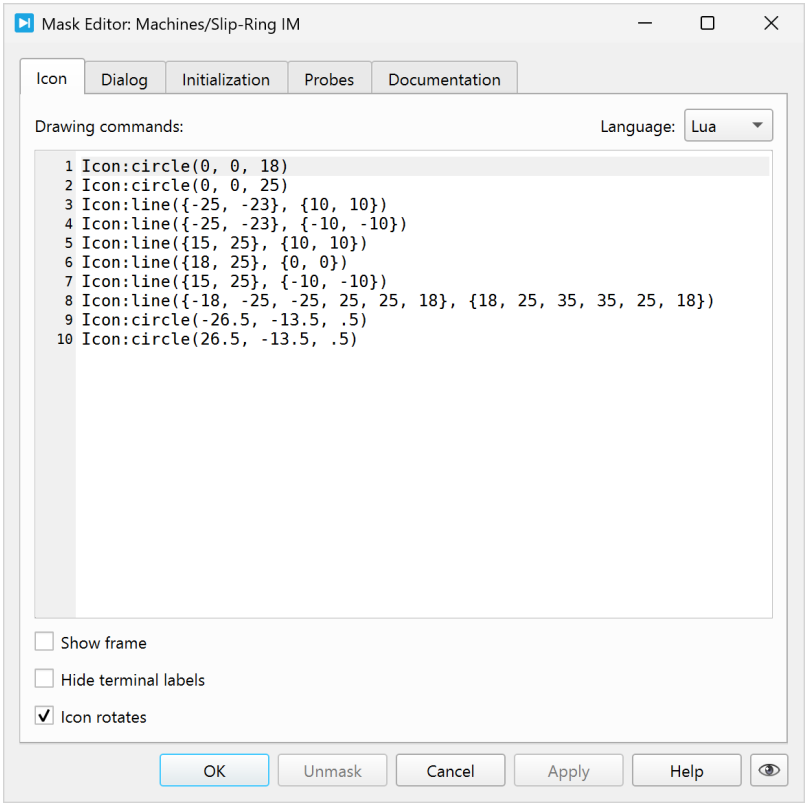


Fig. 3.13: Mask Editor Icon Tab

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.

Table 3.1: 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 string specifying the color name (preferred). All appearance-sensitive "PLECS Colors" (p. 92) are allowed. • A vector `{r, g, b}` of three integers in the range from 0 to 255 specifying the color in RBG format. This color applies in `light` mode and is automatically transformed in `dark` mode (similar hue, inverted lightness). • A table `{light = {r, g, b}, dark = {r2, g2, b2}}` assigning custom colors in RGB format to each appearance. **Note** The table's `dark` mode color defaults to the `light` mode color, such that `{light = {r, g, b}}` results in a fixed color for all appearances.

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.

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 counter-clockwise. The 0 degree angle is at 3 o'clock. Clockwise arcs can be drawn using a negative *span* angle.

Color

The commands

```
Icon:color('colorname')
Icon:color(r, g, b)
Icon:color({r, g, b})
Icon:color({light = {r, g, b}, dark = {r', g', b'}})
```

change the current drawing color. The new color can be defined by:

- A string *colorname* specifying the color name (preferred). All appearance-sensitive "PLECS Colors" (p. 92) are allowed.

- Three integers *r*, *g* and *b* or a vector `{r, g, b}` of three integers in the range from 0 to 255 specifying the color in RBG format. This color applies in `light` mode and is automatically transformed in `dark` mode (similar hue, inverted lightness).
- A table `{light = {r, g, b}, dark = {r', g', b'}}` assigning custom colors in RGB format to each appearance.

> **Note**
>
> The table's `dark` mode color defaults to the `light` mode color, such that `{light = {r, g, b}}` results in a fixed color for all appearances.

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`). The recommended path separator is the forward slash (`/`). If you want to use the backslash (`\`) instead, note that the backslash is also used as an escape character, so you must enter *two consecutive* backslashes for a single path separator (e.g. `C:\\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

↪ *Added in PLECS 4.2.*

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 Combo Box parameter is a string containing the integer value of the chosen option or the expression entered by the user.

Querying Preferences

The command

```
Preferences:get('Appearance')
```

returns the current app appearance as a string, either `"light"` or `"dark"`. This can be used for appearance-sensitive drawing commands beyond the normal color transformation.

The command

```
Preferences:get('DrawANSI')
```

returns true if the **Symbol format** setting is set to `ANSI` instead of `DIN`. This can be used for drawing commands which differ between these formatting conventions.

Legacy Command Syntax (PLECS 4.1 and older)

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

Table 3.2: Legacy Command Syntax

Command	Syntax
Text	`text('text')` `text(x, y, 'text')`

continues on next page

Table 3.2 – continued from previous page

Command	Syntax
Line	`line(xvec, yvec)`
Patch	`patch(xvec, yvec)`
Circle	`circle(x, y, r)`
Color	`color(r, g, b)`
Image	`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 unset 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.

PLECS Colors

↪ *Added in PLECS 4.8.*

The PLECS palette contains all available appearance-sensitive colors which all PLECS components use for drawing their block/mask icon. They can be used in the “Mask Icon Drawing Commands” (p. 84).

Table 3.3: PLECS Palette

Color name	Description
`"text"`	default text color
`"electrical"`	electrical domain color
`"signal"`	signal connection color
`"axis"`	axis color for icons
`"thermal"`	thermal domain color
`"thermalBg"`	thermal domain background
`"magnetic"`	magnetic domain color
`"magneticBg"`	magnetic domain background
`"rotational"`	rotational domain color
`"rotationalBg"`	rotational domain background
`"translational"`	translational domain color
`"translationalBg"`	translational domain background
`"event"`	event connection color
`"normalFg"`	normal foreground
`"normalBg"`	schematic background (configurable in dark mode)

PLECS Icon Library

PLECS comes with a preinstalled Lua library for drawing basic electrical components. A selection of the available icons is displayed in Fig. 3.14.

To gain access to this icon library, start by including the PLECS Icon Library

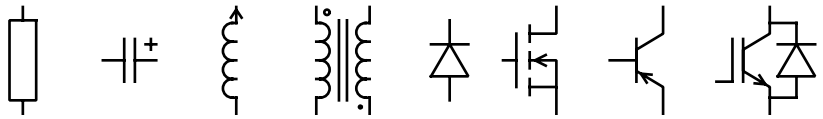


Fig. 3.14: A selection of the built-in component icons of the PLECS Icon Library.

via `local IconLib = require('PlecsIconLib')`. To reproduce the above image, you can copy the following code:

```
local IconLib = require('PlecsIconLib')
-- Passive elements
IconLib.resistor(-140, 0)
IconLib.capacitor(-100, 0, {rotate=270, flip=true, showPolarity=true})
IconLib.inductor(-60, 0, {showArrow=true})
IconLib.transformer(-20, 0, {showCore=true, showPolarity=true, polarity="+-
↪"})

-- Semiconductor elements
IconLib.diode(20, 0)
IconLib.mosfet(60, 0, {deviceType="n"})
IconLib.bjt(100, 0, {deviceType="pnp"})
IconLib.igbtDiode(140, 0, {shiftGate=true})
```

In the above code, each line draws exactly one icon. Each function follows the same syntax rules:

```
IconLib.component(x0, y0, params)
```

Here, `component` stands for the component icon to be drawn, `x0` and `y0` are the coordinates of the origin of the icon (see Fig. 3.15 on p. 95), and `params = {key1=val1, key2=val2}` is a table containing additional settings to customize the rendering of the icon (see "Additional Parameters" (p. 95).).

The PLECS Icon Library supports drawing the following icons:

Component	Function Call
Node	`IconLib.node(x0, y0)`
Resistor (p. 905)	`IconLib.resistor(x0, y0, params)`
Capacitor (p. 597)	`IconLib.capacitor(x0, y0, params)`
Inductor (p. 779)	`IconLib.inductor(x0, y0, params)`
Ideal Transformer (p. 744)	`IconLib.transformer(x0, y0, params)`
Diode (p. 641)	`IconLib.diode(x0, y0, params)`
Thyristor (p. 1024)	`IconLib.thyristor(x0, y0, params)`
MOSFET (p. 828)	`IconLib.mosfet(x0, y0, params)`
MOSFET with Diode (p. 832)	`IconLib.mosfetDiode(x0, y0, params)`
IGBT (p. 746)	`IconLib.igbt(x0, y0, params)`
IGBT with Diode (p. 752)	`IconLib.igbtDiode(x0, y0, params)`
Bipolar Junction Transistor	`IconLib.bjt(x0, y0, params)`
Junction Field Effect Transistor	`IconLib.jfet(x0, y0, params)`

Note

Legacy variants with function names that start with a capital letter are deprecated and should not be used anymore.

Icon Geometry

The PLECS Icon library uses the coordinate system introduced in section "Mask Icon Coordinates" (p. 90). Icons rotate around their origin in clockwise direction and are, by default, drawn identically to the corresponding PLECS component. The geometric properties of the supported icons are summarized in the following figure:

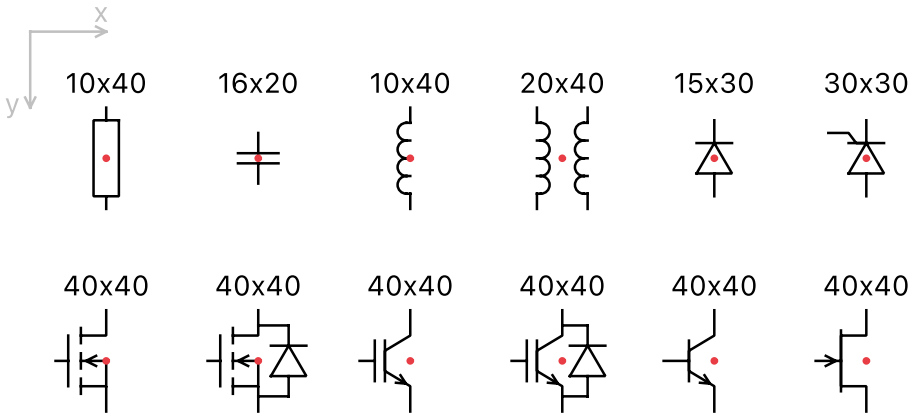


Fig. 3.15: Icon origin and size. The origin of each icon component is indicated with a red dot. The maximal size in upright position (i.e., `rotate=0` or `rotate=180`) is displayed above each icon.

Additional Parameters

Additional parameters are passed via a table of `key = value` entries. The following table summarizes the available options.

Keyword	Data Type	Description	Elements
`rotate`	Number	Specifies the rotation angle (clockwise rotation). Must be an integer multiple of 90.	*all elements*
`flip`	Boolean	Flips the unrotated icon about the x-axis.	*all elements*
`wireLength`	Number Table of Numbers	Specifies the additional length of the terminal wires. Can be either a number or a table of numbers (one number per terminal). The default value of `0` reproduces the standard icons used in the PLECS library browser.	*all elements*
`showArrow`	Boolean	Adds an arrow decoration to indicate the direction of positive current flow.	inductor
`showPolarity`	Boolean	Adds an annotation for the polarity of the device.	capacitor transformer
`polarity`	String	Sets the polarity of the transformer. Available options are `"+"`, `"-"`, `"+-"` and `"-+"`. Note that the annotation is only shown if `showPolarity` is enabled.	transformer
`showCore`	Boolean	Adds a magnetic core decoration.	transformer
`deviceType`	String	Defines the transistor type. For a MOSFET or a JFET, use `"n"` and `"p"` to indicate the channel type. For a BJT, the available options are `"npn"` and `"pnp"`. By default, `"n"` and `"npn"` are assumed, respectively.	mosfet mosfetDiode bjt jfet
`shiftGate`	Boolean	Shifts the gate terminal towards the source terminal (MOSFET, JFET) or the emitter terminal (IGBT). By default, the gate terminal is centered.	mosfet mosfetDiode igbt igbtDiode jfet

Rotating and Flipping

You can flip and rotate basic component icons to create your own composite images. Icons are always flipped first and then rotated. We use the x-axis as mirroring axis in the unrotated coordinate system. As a consequence, this means that the mirroring axis rotates together with the icon. You can find an illustration of the combined effect of rotating and flipping icons in Fig. 3.16. The corresponding Lua code is listed below.

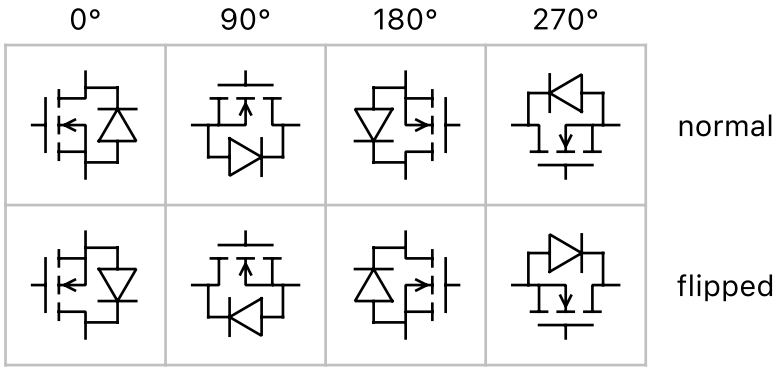


Fig. 3.16: Illustration of the effect of the `rotate` and `flip` keywords for a MOSFET with Diode.

```lua
local IconLib = require('PlecsIconLib')

-- 0° Rotation
IconLib.mosfetDiode(-90, -30, {rotate=0, flip=false})
IconLib.mosfetDiode(-90, 30, {rotate=0, flip=true})

-- 90° Rotation
IconLib.mosfetDiode(-30, -30, {rotate=90, flip=false})
IconLib.mosfetDiode(-30, 30, {rotate=90, flip=true})

-- 180° Rotation
IconLib.mosfetDiode(30, -30, {rotate=180, flip=false})
IconLib.mosfetDiode(30, 30, {rotate=180, flip=true})

-- 270° Rotation
IconLib.mosfetDiode(90, -30, {rotate=270, flip=false})
IconLib.mosfetDiode(90, 30, {rotate=270, flip=true})
```

(continues on next page)

(continued from previous page)

```
-- Add annotations
Icon:text(-90, -70, "0°")
Icon:text(-30, -70, "90°")
Icon:text(30, -70, "180°")
Icon:text(90, -70, "270°")
Icon:text(150, -30, "normal")
Icon:text(150, 30, "flipped")

-- Draw the frame
Icon:color('axis')
Icon:line({-60, -60}, {-60, 60})
Icon:line({0, 0}, {-60, 60})
Icon:line({60, 60}, {-60, 60})
Icon:line({-120, -120}, {-60, 60})
Icon:line({120, 120}, {-60, 60})
Icon:line({-120, 120}, {0, 0})
Icon:line({-120, 120}, {-60, -60})
Icon:line({-120, 120}, {60, 60})
```

Additional Wire Length

At times, you may want to separate two elements by a distance larger than the combined wire length. You can extend the length of the terminal wires via the `wireLength` parameter. You may either specify a single number to extend (or decrease) every terminal wire by the same amount. Or, you can control the length of each terminal individually by providing a list of numbers. The length of that list must match the number of terminals. The order used in this list is defined for the component in the original coordinate system, according to Fig. 3.17.

Mask Dialog

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

Prompts and Associated Variables

Mask parameters are defined by a **Prompt**, a **Variable** 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. The possible parameter types are described in the table below.

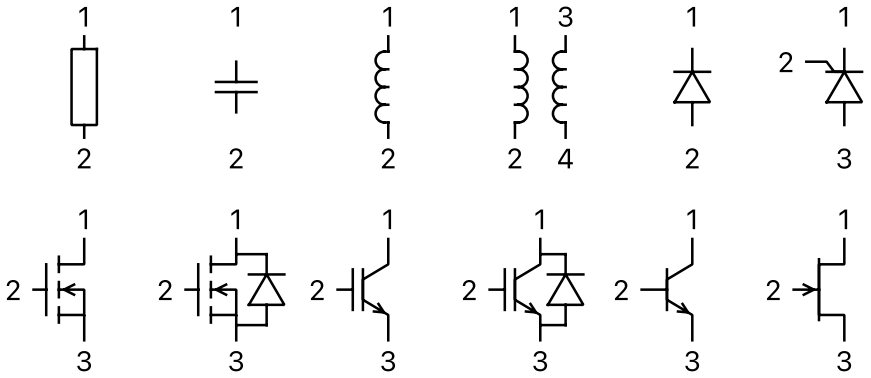


Fig. 3.17: Identification of the terminal oder for the `wireLength` parameter.

Type	Description
`Edit`	Shows a line edit field. The entered text is interpreted as a MATLAB/Octave expression and is evaluated when a simulation is started.
`String`	Shows a line edit field and a selector that controls whether the entered text is interpreted as a literal string or as a MATLAB/Octave expression that *evaluates* to a string.
`Combo Box`	Shows a pop-up menu that allows the user to select an item from a list of possible options. Use the **Combo box values** editor below the parameter list to enter the list of possible options and their associated integer values that are assigned to the parameter variable. The values must be unique integers.
`String-Valued Combo Box`	Shows a pop-up menu that allows the user to select an item from a list of possible options. Use the **String-valued combo box values** editor below the parameter list to enter the list of possible options and their associated string values that are assigned to the parameter variable.
`Thermal`	Shows a thermal parameter editor that allows the user to specify a thermal description (see "Thermal Description Parameter" (p. 179) for details). If you enter a text in the **Device type filter** editor below the parameter list, the thermal parameter editor will show only thermal descriptions that have the matching device type. Otherwise, all thermal descriptions will be shown.

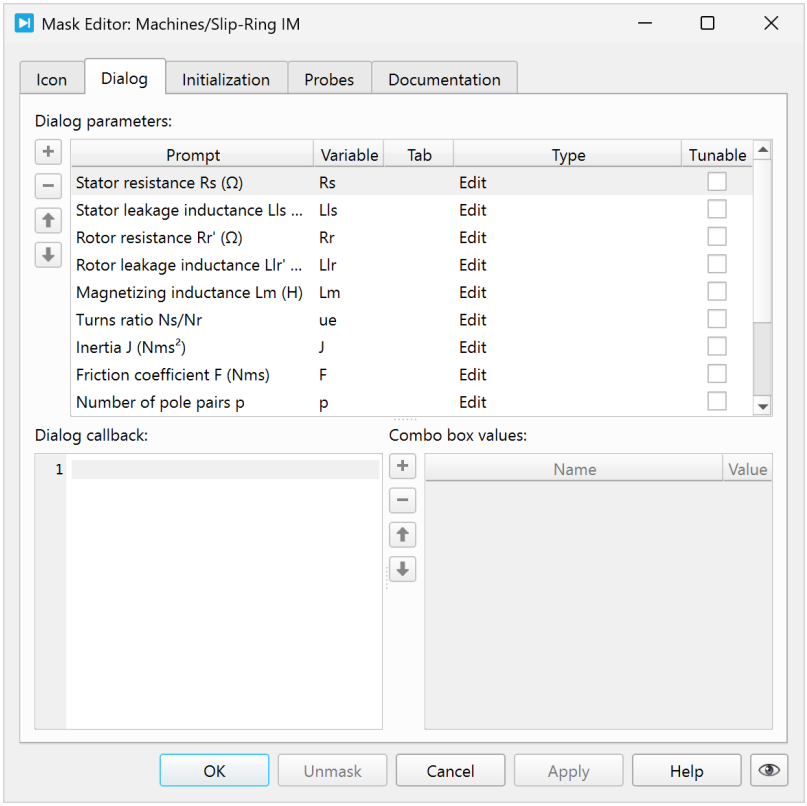


Fig. 3.18: Mask Editor Dialog Tab

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.

> **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.

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

↪ *Added in PLECS 4.2.*

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 Combo Box parameter is a string containing the integer value 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.

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

The following example only shows the parameter named 'parameter' when the first configuration of a Configurable Subsystem (p. 620) is selected.

```
Dialog:set('parameter', 'Visible', Dialog:get('Configuration') == '1');
```

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

↪ *Added in PLECS 4.3.*

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

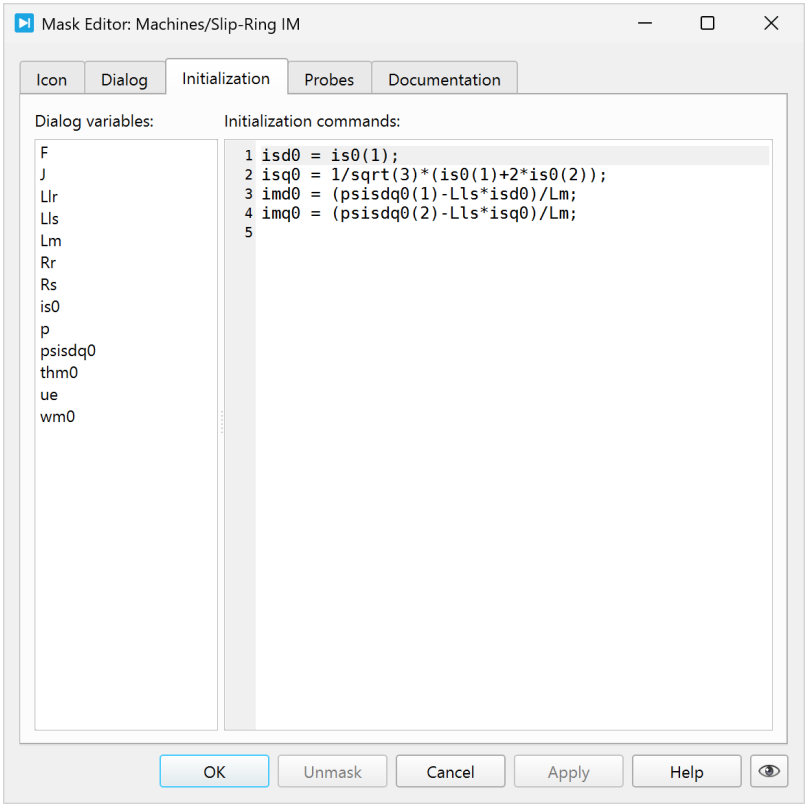


Fig. 3.19: Mask Editor Initialization Tab

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 commands:

```
plecs('error', '<error message>')
return
```

Notice that the `plecs('error', '<error message>')` command by itself does not cause an early exit of the execution of the mask initialization commands, so it should generally be followed by a **`return`** statement to avoid subsequent errors.

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

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

To show information in the Diagnostics window, use the command:

```
plecs('info', '<info message>')
```

> **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.

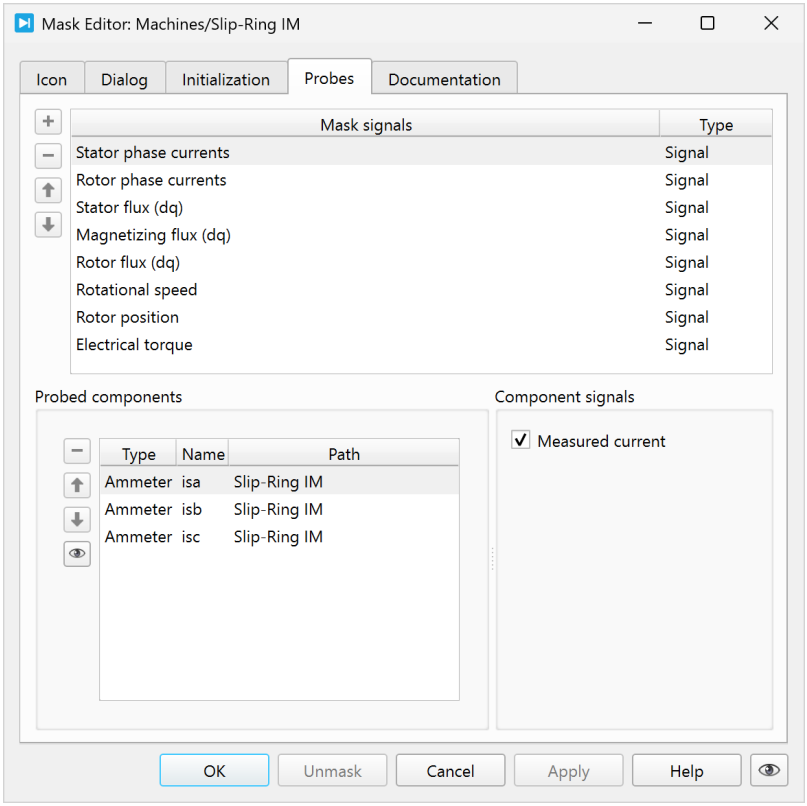


Fig. 3.20: Mask Editor Probes Tab

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.

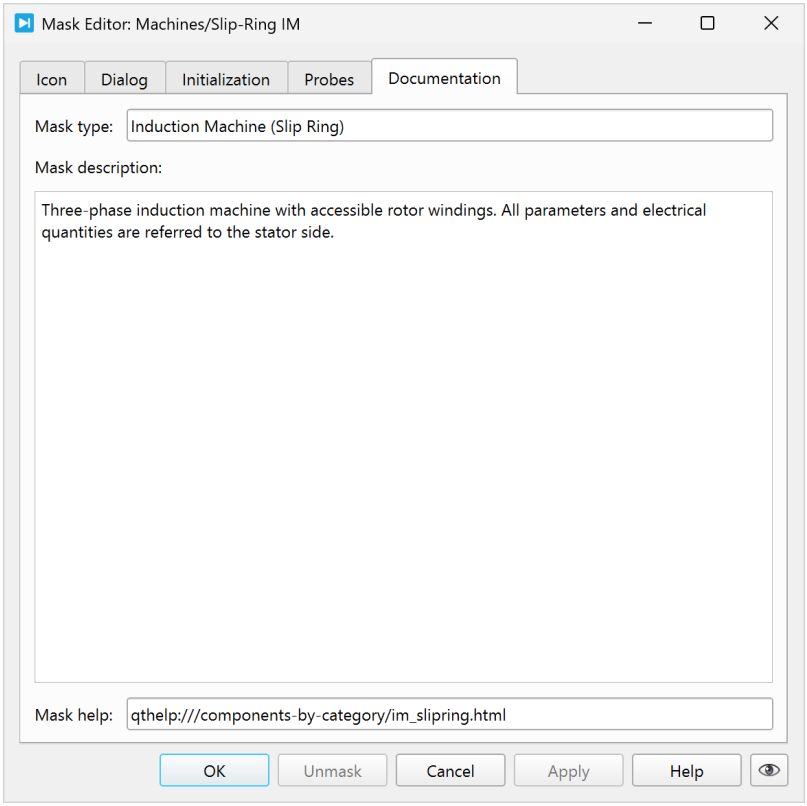


Fig. 3.21: Mask Editor Documentation Tab

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.

Mask Help

↪ *Added in PLECS 4.6.*

The mask help is a URL that provides documentation in addition to that provided by the mask description. This documentation is shown in a separate window when the user clicks the **Help** button in the dialog box of the masked subsystem. Currently, the following URL types are supported:

Remote URL
A URL of the form `https://www.plexim.com` is opened using the default browser of your operating system.

Local HTML File
A local HTML file is specified with an absolute path (e.g. `file:///C:/absolute/path/helpdoc.html`) or with a relative path (e.g. `file:relative/helpdoc.html`). A relative path is resolved relative to the parent folder of the model file containing the subsystem. If the subsystem is a library link (p. 72) or model reference (p. 824), the relative path is resolved relative to the parent folder of the *source* library or model file. Local HTML files are also opened using the default browser of your operating system.

Unprotecting Masked Subsystems

If you define a mask icon for a Subsystem block, PLECS automatically protects the block and the ports of the underlying schematic. You can no longer resize the Subsystem block or change the positions of the terminals on the subsystem frame. Additionally, it is not possible to add or remove ports in the subschematics. The purpose of this protection is to prevent the user from making unintentional changes that might render the icon useless.

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

Getting Started with Lua

↪ *Added in PLECS 4.2.*

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[1].

[1] https://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 `''` or `""` 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 newline are supported.

Long string 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

↪ *Added in PLECS 4.1.*

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 and 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.

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” (p. 135)).

> **Note**
>
> In PLECS Standalone, assertions are partially disabled during analyses (p. 227) and simulation scripts (p. 313). 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

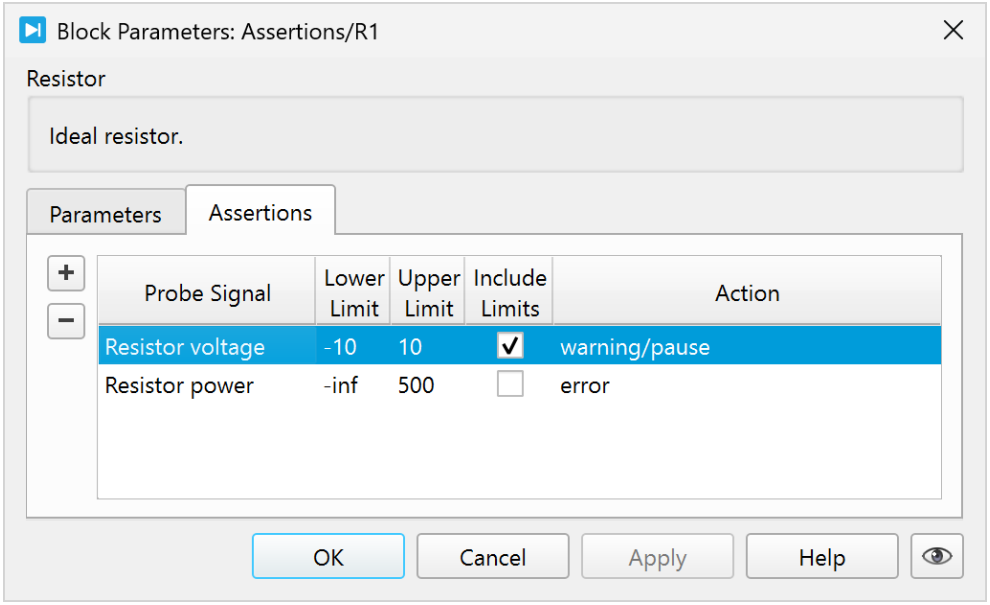
There is one basic Assertion block (p. 563) that interprets its input signal as a boolean value. While the value is non-zero, the assertion holds; whenever the input becomes zero, the assertion fails.

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 (p. 560)
- Assert Dynamic Range (p. 561)
- Assert Dynamic Upper Limit (p. 562)
- Assert Lower Limit (p. 564)
- Assert Range (p. 565)
- Assert Upper Limit (p. 566)

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.



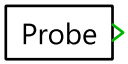
Use the “+” and “-” buttons to add or remove assertions. In the different columns, the parameters for the assertions can be provided: the probe signal

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” (p. 131) 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.

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.

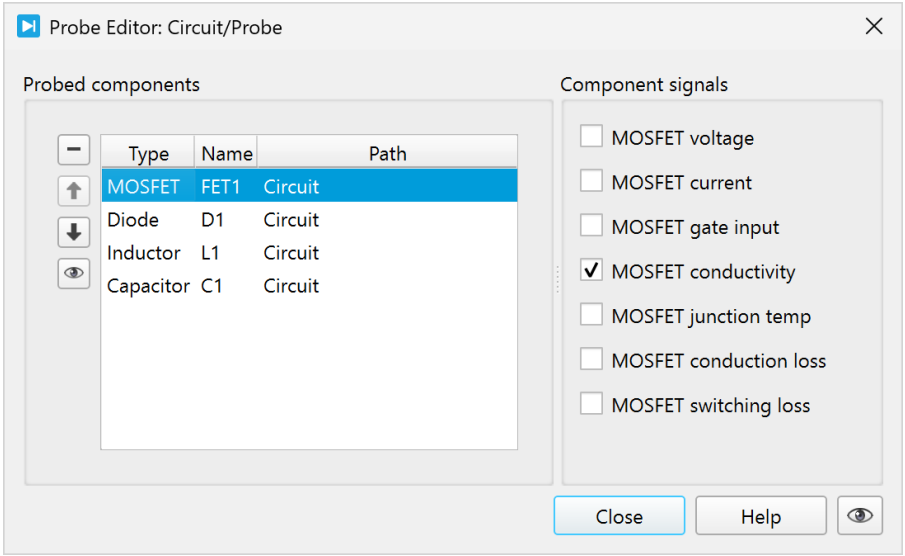


Fig. 3.22: Probe Editor

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** (on macOS: **Cmd**) 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, 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.

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 Scope block (p. 942) can be placed on the Simulink worksheet or in the PLECS circuit. The appearance of the PLECS scope is depicted in Fig. 3.23 on p. 125. The scope contains a plot area and optional **Zoom Area**, **Saved Views**, **Traces** and **Data** 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 Area**, **Saved Views**, **Traces** and **Data** 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.

Legend

↪ *Added in PLECS 4.4.*

You can show a legend on top of each plot by choosing **Show legend** from the **View** menu or from the context menu. The default text of a signal label is derived from the name of the block from which the signal originates and possibly a probe signal name. To modify the text, double-click on a label. If you erase the complete label text, the default text is restored.

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** (on macOS: **Cmd**) key 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**, the zoom area is defined by dragging the zoom cursor over a certain portion of the plot.

Zoom to Fit 

Clicking the **Zoom to Fit** button 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. **Constrained 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 (p. 942).

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 (p. 942).

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 section “Scope” (p. 64)).

To change individual curve properties (color, line style, line width, marker style, and marker size), 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.

↪ *Changed in PLECS 5.0:* Added customizable markers.

Locally changed properties are stored persistently in the model file. 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** 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 table header in the **Data** window allows for additional data analysis columns to be displayed. The following analyses can be performed on the data between the two cursors:

- **Delta** – difference $(y_2 - y_1)$ or $(t_2 - t_1)$ between the cursor values
- **Slope** – $\left(\frac{y_2-y_1}{t_2-t_1}\right)$ for signal rows and $\left(\frac{1}{t_2-t_1}\right)$ for the **Time** row
- **Min** – minimum value

- **Max** – maximum value
- **Abs Max** – maximum absolute value
- **Mean** – mean value
- **RMS** – root mean squared
- **THD** – total harmonic distortion

For meaningful RMS and THD values, the cursor range must be equal to the period of the fundamental frequency.

↪ *Changed in PLECS 4.9:* Added **Slope** analysis.

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” (p. 126).

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. The way traces are distinguished can be modified (see "Changing Curve Properties" (p. 121)).

Traces can also be added and removed by simulation scripts. For details, see section "Handling of Traces in Scopes" (p. 338).

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 (p. 942). The plot background color can be changed in the PLECS preferences (see section "Scope" (p. 64)).

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 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".

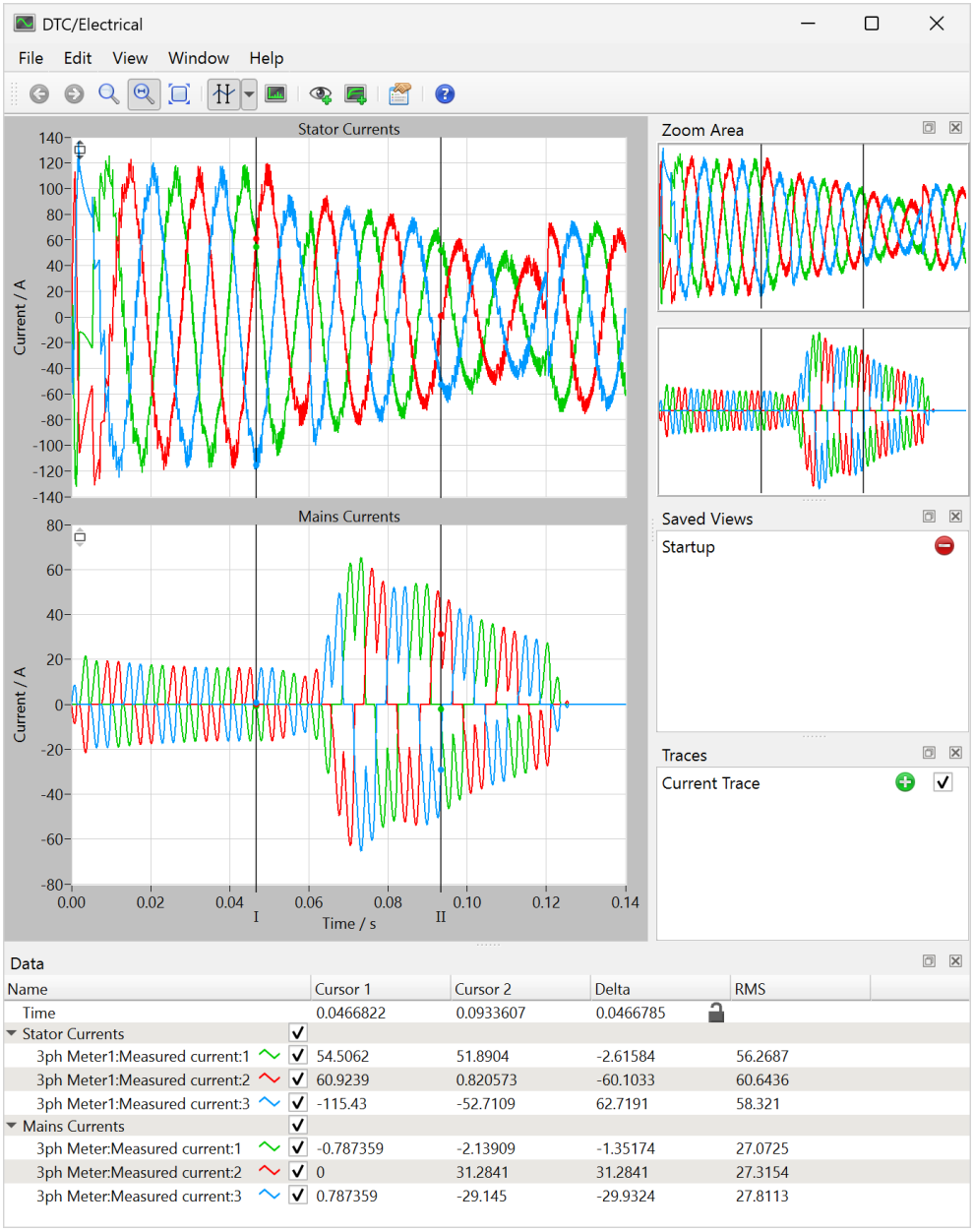


Fig. 3.23: PLECS scope

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. $T = n/f_0$.

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” (p. 119) for details.

Calculation of the Fourier coefficients

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

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

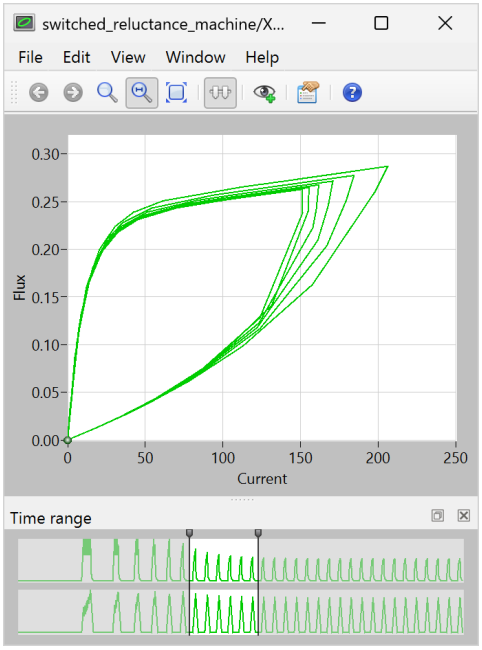
where

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

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

Using the XY Plot

The XY plot is used to display the relationship between two signals, x and y. In every simulation step the x and y input signals are taken as coordinates for a new point in the XY plot. 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 (p. 1130).

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” (p. 119) for details.

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.

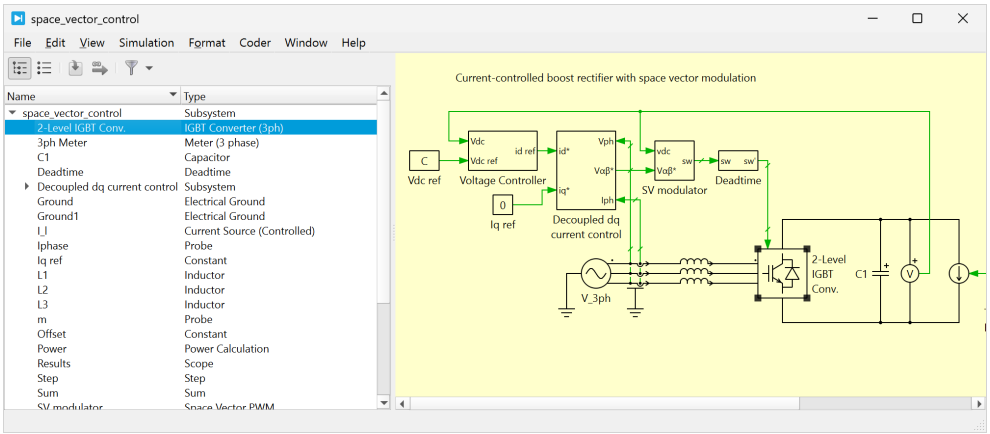


Fig. 3.24: The Circuit Browser

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 three 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"

(p. 113)). 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” (p. 60)).

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 **Edit > Subsystem** 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 a model file, it encrypts the respective sections to protect the circuit description from unauthorized access.

Simulation Parameters

PLECS Standalone Parameters

This section describes the simulation parameters available for PLECS Standalone. For the PLECS Blockset simulation parameters please refer to the "PLECS Blockset Parameters" (p. 143).

To open the parameter dialog, select **Simulation parameters** from the **Simulation** menu of the schematic editor or press **Ctrl+E** (on macOS: **Cmd+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 (s). If a simulation is started from a stored system state (see "System State" (p. 142)), this parameter is ignored and the simulation time specified in the system state is used instead.

Time Span

The simulation ends when the simulation time has advanced by the specified time span.

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.

Variable-step

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.
- When **auto** is selected, PLECS starts a simulation with **DOPRI** and automatically switches to **RADAU** if the system is found to become stiff during a simulation. This is the default choice for a variable-step solver.
  ↪ *Added in PLECS 4.5.*

> **Note**
>
> When using PLECS Spice, additional solvers are available as documented in "SPICE Simulation Parameters" (p. 406).

Fixed-step

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 "Physical Model Discretization" (p. 39). 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:

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

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).

Jacobian computation

This parameter specifies how the solver calculates the Jacobian matrix required for the integration of stiff systems. The default setting *analytical* uses exact derivatives provided by the model for improved accuracy and performance. Select *finite differences* to revert to the legacy method, where the Jacobian is approximated numerically by perturbing each state variable.

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.

Non-ideal switch resistance (code generation)
This parameter specifies the resistance for non-ideal switches when generating code from a simulation model. For details refer to section “Ideal and Non-Ideal Switch Models in Electric Circuits” (p. 344).

↪ *Added in PLECS 4.9.*

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" (p. 46).

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.

Remove unused state-space outputs
When this option is checked, PLECS removes the output equations for physical meters that are not used in the model in order to avoid unnecessary calculations. You may need to uncheck this option if you want to calculate state-space matrices in a simulation script, see "Extraction of State-Space Matrices" (p. 246). By default, this option is checked.

↪ *Added in PLECS 4.8:* In older models,by default,this option is unchecked.

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 (p. 675), Rotational Model Settings (p. 917) and Translational Model Settings (p. 1062)).

Data types

Use floating-point data type for fixed-point signals
When this option is enabled, PLECS will replace all fixed-point data types with the target floating-point data type (see section "Data Types" (p. 48)).

↪ *Added in PLECS 4.8.*

Assertions

Assertion action

Use this option to override the action that is executed when an assertion fails (see Assertion block (p. 563)). 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 section "Assertions" (p. 113)).

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 (p. 892) or a Function block (p. 715). 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`.

↪ *Changed in PLECS 3.7:* In older models,the default is `warning`.

Datatype overflow

This option determines the diagnostic action to take if PLECS encounters a data type overflow. PLECS can issue an error or a warning message or can continue silently. In the latter two cases, the result is handled according to the individual *data type overflow handling* setting of the block. If the individual setting is `Assert with error`, PLECS always issues an error message.

↪ *Added in PLECS 4.8.*

Datatype inheritance conflict

This option allows to apply less strict data type inheritance rules (see section “Data Types” (p. 48)). In new models, the default is `error`.

↪ *Added in PLECS 4.8:* In older models,the default is `warning`.

Continuous sample time conflict

This option determines the diagnostic action to take if PLECS detects a continuous sample time conflict (see section “Continuous Sample Time Conflicts” (p. 45)). In new models, the default is `error`.

↪ *Added in PLECS 4.8:* In older models,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 section “Loss Calculation” (p. 175)). 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 section “Event Detection Loop” (p. 38)). 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 (p. 973). 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 simulation time,
- 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),
- the values of all continuous and discrete state variables in the control block diagram (e.g. integrators, transfer functions, delays),
- custom state information of C-Script blocks

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, removing, renaming, changing path (when moving in or out of a subsystem) or changing the number of internal states of blocks that have continuous or discrete state variables associated with them will invalidate the 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” (p. 67)).

> **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.

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 “PLECS Standalone Parameters” (p. 135).

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” (p. 34).

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 “Physical Model Discretization” (p. 39). All other continuous state variables are updated using the Forward Euler method. This method can be used with both *variable-step* and *fixed-step* solvers.

Discrete State-Space Options

Sample time

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

Refine factor

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

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

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.

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 (p. 949) 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 (p. 892) or a Function block (p. 715). 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`.

↪ *Changed in PLECS 3.7:* In older models,the default is `warning`.

Datatype overflow
This option determines the diagnostic action to take if PLECS encounters a data type overflow. PLECS can issue an error or a warning message or can continue silently. In the latter two cases, the result is handled according to the individual *data type overflow handling* setting of the block. If the individual setting is `Assert with error`, PLECS always issues an error message.

↪ *Added in PLECS 4.8.*

Datatype inheritance conflict
This option allows to apply less strict data type inheritance rules (see section "Data Types" (p. 48)). In new models, the default is `error`.

↪ *Added in PLECS 4.8:* In older models,the default is `warning`.

Continuous sample time conflict
This option determines the diagnostic action to take if PLECS detects a continuous sample time conflict (see section "Continuous Sample Time Conflicts" (p. 45)). In new models, the default is `error`.

↪ *Added in PLECS 4.8:* In older models,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 (p. 973). 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 section "Loss Calculation" (p. 175)). 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 (p. 563)). 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" (p. 113)).

Use floating-point data type for fixed-point signals
When this option is enabled, PLECS will replace all fixed-point data types with the target floating-point data type (see section "Data Types" (p. 48)).

↪ *Added in PLECS 4.8.*

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" (p. 46).

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.

Remove unused state-space outputs
When this option is checked, PLECS removes the output equations for physical meters that are not used in the model in order to avoid unnecessary calculations. You may need to uncheck this option if you want to calculate state-space matrices in a simulation script, see "Extraction of State-Space Matrices" (p. 258). By default, this option is checked.

↪ *Added in PLECS 4.8:* In older models,by default,this option is unchecked.

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 (p. 675), Rotational Model Settings (p. 917) and Translational Model Settings (p. 1062)).

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 "Target" (p. 351).

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” (p. 358).

Error Messages

This section provides detailed information about common error messages in PLECS and their solutions.

Could not solve the algebraic loop

This error occurs when the algebraic loop solver cannot find a solution that is consistent with all the blocks involved in an algebraic loop. 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 block.

Solution 1: Avoid algebraic loop

If possible, the best solution is to avoid the algebraic loop altogether by reformulating the model.

Using the following equation as an example $y = xy/5 - x$, the algebraic loop is avoided if the equation is rewritten as $y = x/(x/5 - 1)$.

Example Model

- See the example model **"Could not solve the algebraic loop - Example 2"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

When to use algebraic loops as a modeling tool

Algebraic loops are a helpful tool for modeling fast interactions between physical quantities with small time constants. A model using such a loop will make the describing differential equations numerically less stiff and so easier to solve.

Take a variable resistor as an example: Since every real resistor will show some, albeit small, inductive behavior, the model of the resistor could be built using a Variable Resistor with Constant Inductor component. In this model, the current through the resistor is not directly affected by a change in the resistance value and thus, there is no algebraic loop. As the additional small inductor will create transients with a very small time constant, this can lead to issues with numerical integration. In this case, a pure Variable Resistor component will add an algebraic loop to the model and solve these issues.

Solution 2: Break the algebraic loop

If the model cannot be reformulated then the loop can be broken by introducing a low-pass filter. The bandwidth of the low-pass filter needs do be within certain boundaries for this to work: It needs to be high enough so its dynamics do not interfere with the system response and low enough to avoid numerical stiffness of the system. Please note that *algebraic loops should not be broken with a Memory block* in simulations with a variable time-step solver.

Example Model

- See the example model **"Could not solve the algebraic loop - Example 2"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

Solution 3: Restrict the operation range

Some algebraic loops can only be solved in a certain operation range of the circuit.

Why the error occurs: An example is a constant power load. If the load voltage or current is zero, the load power cannot be held constant with finite currents or voltages. In this case, a lower limit for the load condition has to be specified.

Example Model

- See the example model **"Could not solve the algebraic loop - Example 1"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

Solution 4: Provide an initial value for the solver

The solver sometimes needs the right initial value at simulation start to find the solution of an algebraic loop. This can be done with an initial condition block.

Example Model

- See the example model **"Could not solve the algebraic loop - Example 3"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Encountered 1001 consecutive zero-crossings

The error message indicates that a particular signal is toggling back and forth during consecutive solver steps. The message highlights the component that causes the zero-crossing issue.

Solution 1: Limiting the switching frequency

This solution relies on limiting the switching frequency of the Compare to Constant block. There are two possibilities to do that:

- Set the switching frequency explicitly by inserting a zero order hold with appropriate sample time in front of the Compare to Constant block.

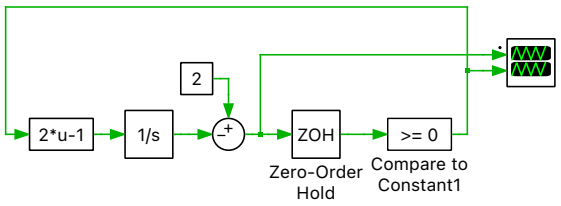


Fig. 3.25: Solution 1a: Zero-Order Hold

- Add a hysteresis band by replacing the Compare to Constant block with a Relay block with suitable upper and lower thresholds (e.g. 1×10^{-6} and -1×10^{-6}).

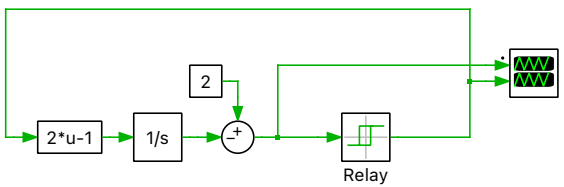


Fig. 3.26: Solution 1b: Hysteresis Band

Why the error message occurs: Whenever the input signal of the Compare to Constant block crosses zero, the change of the output signal immediately causes the input signal to change its direction. This leads to a zero-crossing again in the next simulation step. This repeats until the solver reaches the maximum of configured consecutive zero crossings (1000 by default).

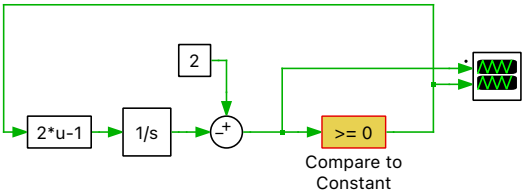


Fig. 3.27: Model causing consecutive zero-crossings

This behavior will bring the simulation to a virtual standstill. To solve the problem, the switching frequency of the Compare to Constant block must be limited.

Solution 2: Adjusting the electrical machine parameter

The problem can be solved by changing the machine configuration parameter to **Voltage behind reactance** which is based on a variable inductor and phase current measurements. These always provide a continuous machine model input during a simulation.

Why the error message occurs: Taking, for example, a model featuring an electrical machine in "Rotor reference frame" configuration which is implemented with voltage controlled current sources.

Example Model

- See the example model **"Encountered 1001 consecutive zero-crossings - Example 2"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

Here, the problem occurs when the zero-crossing of a phase current happens during an interval where both high-side and low-side switch have zero-input gate signals. A first diode is conducting, when the current is negative and a second one, when the current is positive. Thus, the small change around zero in current causes a voltage jump from zero to the DC voltage, which in turn results in the sign of the current injection to flip. The process then repeats. The behavior will bring the simulation to a virtual standstill and PLECS issues the error message.

State discontinuity after switching

The error occurs, when there is an instantaneous change in a model state after a switching event. Examples include interrupting an inductive current or changing a capacitive voltage instantaneously. The error message will explicitly state which type of phenomena causes the error and at what time during the simulation.

Solution 1: Provide an alternative current path or interrupt inductive currents

For DC systems: Please ensure that an alternative current path is provided by free-wheeling diodes or snubber components. Be cautious when using snubber components: Excessively large or small values scan lead to large voltage and current transients, high losses, or numerically stiff systems.

For AC systems: Replace the Switch component with a Breaker. The Breaker component opens when both the input signal and the breaker current are zero. Therefore, this circuit breaker can be used to interrupt inductive AC currents.

Why the error occurs: The error indicates a likely systematic problem of the circuit topology and/or the modulation scheme used. In a real system, chopping inductive currents will result in voltage spikes across the inductors, which likely will damage the circuit. So, when instantaneous change in inductive current happens and the current cannot find a new path in the circuit (e.g. through a free-wheeling diode or resistance), PLECS will report this error.

Occasionally the error is due to insufficient solver accuracy which is set with the **Relative tolerance** parameter in the solver panel.

For example, consider the following AC circuit with an RL load:

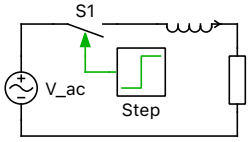


Fig. 3.28: Inductive system with state discontinuity

Example Model

- See the example model **"State discontinuity after switching - Example 2"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

With "S1" initially closed, current will flow in the circuit."S1" is an ideal device. When opened, it will instantaneously chop any currents flowing through the switch. The voltage across the inductor can be described with $v = L\frac{di}{dt}$. If the current changes instantaneously, then there will be an infinite voltage spike across the component. PLECS raises an error under this condition.

In this simple inductive circuit example, the issue is topological since there is no path for the current to flow. This is also possible with more advanced converter circuits.

Solution 2: Ensure a correct Modulator implementation

You debug your modulator implementation. For debugging a model, it may be useful to add series or parallel snubber elements to temporarily silence the error, allowing you to study the modulation sequence in more depth. Remove the snubber elements once the issue is resolved.

Why the error occurs: Fundamental errors in modulator implementation can lead to unexpected interruption of inductive currents or shorts across capacitors.

For example, the error may be shown due to a simple error where the gate drive signals are incorrectly assigned.

Example Model

- See the example model **"State discontinuity after switching - Example 5"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Adding a resistance in series with the DC link capacitances allows you to debug the model and observe that the conductivity of the device does not follow the gate input.

Solution 3: Topological considerations

Make sure that no switching element short-circuits an energy-storing element.

Why the error occurs: In certain cases, the circuit's operation will depend on rapidly charging and discharging energy stored in a component, which could lead to this error message.

One example is the "Resonant Full-Bridge SLR Converter". It analyses zero-voltage turn-off of the SLR circuit by modeling an external capacitance in parallel with the switching devices.

> **Example Model**
>
> - See the example model **"State discontinuity after switching - Example 1"**.
> - Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

> **Note**
>
> The switching devices must have a non-zero resistance, otherwise there is an error when the switch closes across the parallel capacitor. In this case, the switch resistance represents the actual resistance of the IGBT device.

Solution 4: Adjust solver accuracy

One common solution is to decrease the **Relative tolerance** in the **Simulation parameters + Solver** tab by one or two orders of magnitude. If the state values before-and-after switching are close to zero, then changing the **Absolute tolerance** value from the default of **Auto** to a small value may resolve the error message as well.

Why the error occurs: Normal changes in model states may be incorrectly detected as a state-discontinuity when there is insufficient solver accuracy. One indicator that this condition is occurring is the relatively small difference between the state value before and after switching, with neither state being exactly equal to zero.

Using the referenced example of a three-phase voltage-source converter, the default case reports a State Discontinuity in the inductor current. However,

the current path is continuous due to the anti-parallel diodes in the IGBT converter. Setting the **Relative tolerance** to **1e-4** or the **Absolute tolerance** to **1e-3** resolves the error.

Example Model

- See the example model **"State discontinuity after switching - Example 4"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Solution 5: Adjust Diode turn-on threshold

Lastly, adjusting the **Diode turn-on threshold** parameter can be beneficial in some instances.

Derivatives are infinite or NaN

This error occurs due to numerically ill conditioned systems. It can also occur, when an input into the state derivative calculation or the calculated derivative term state derivative $\dot{x}$ is either $\pm\infty$ (**inf**, **-inf**) or "not a number" (**NaN**).

As described in "Physical Model Equations" (p. 34), PLECS models physical systems using a piece-wise linear state-space representation. The state derivative $\dot{x}$ is calculated as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

The state derivative can be infinite or "not a number" in all physical domains and the control domain.

Solution 1: Make the system physically and numerically stable

Check that the controller and the plant are stable.

Why the error occurs: In unstable physical or control systems or state-space systems with a near-singular state matrix, state values can arise that approach infinity.

Consider the following instable system:

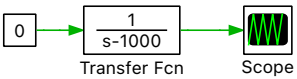


Fig. 3.29: Instable system example

Example Model

- See the example model **"Derivative is infinite or NaN - Example 3"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

In state space the format of the system is $\dot{x} = 1000x$ with an initial condition of $x(0) = 1$. As the simulation proceeds the derivative and state value of x increase exponentially until the state derivative reaches the maximum representable double precision floating point value. At this point, the simulation terminates.

Solution 2: Debug thermal runaway

The solution is to find and correct the underlying flaw in the system causing thermal runaway.

First check the error message to see which model states approach infinity. Then, you can base the debugging process on these states.

Why the error occurs: Typically, the error is due to an underlying flaw in how the system is modeled that results in unstable feedback. The underlying flaw may concern: Excessive current through the device, poor device selection for the given load, excessively high thermal impedances, an unrealistic electrical simulation with large component stresses, and incorrect units in the thermal description file.

Thermal models exhibiting thermal runaway may result in this error message. This is the case, for example, in this modified version of the "Buck Converter with Thermal Model" demo model that has the load resistance reduced by a factor of 10.

Example Model

- See the example model **"Thermal state derivatives are infinite or NaN"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

The resulting current through the device is no longer appropriate for the device's rating. As the temperature increases, the thermal model's extrapolation algorithm predicts higher losses, resulting in even higher temperatures. The device's temperature exceeds the point where the device would be physically damaged and proceeds towards infinity, eventually resulting in the simulation terminating. The simulation error message points towards the thermal capacitance of the diode being the first state to reach infinity. So, in this example model, the root-cause is the excessive current through the device and poor device selection for the given load.

Solution 3: Ensure that division by zero is impossible

Restrict the inputs to any divisor such that it cannot be equal to zero.

Why the error occurs: A simulation block's output may be **inf**, **-inf**, or **NaN** under certain conditions. One common source is division by zero or near-zero,

which will return **inf**, **-inf**, or **NaN** depending on the dividend. If the division result influences a system state then PLECS will report this error message.

Solution 4: Adjust the Division by zero option

Set the **Division by zero** diagnostic option in the **Simulation parameters + Diagnostics** tab to **Error**.

When this option is set to **Warning**, it is recommended to carefully review the simulation diagnostics window. Division operations are possible when using the Divide block or Function block.

Why the error occurs: In some cases dividing by a near-zero value will result in the error message but not a divide by zero warning. See the example below where **eps(0)** is a very small non-zero value. The result of this division is **inf** which in turn influences the capacitor voltage state.

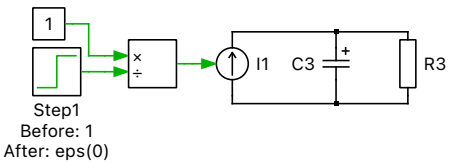


Fig. 3.30: Division by a near-zero value

Example Model

- See the example model **"Derivative is infinite or NaN - Example 1"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Solution 5: Ensure that the C-Script return usable values

Debug the C-Script such that no calculation returns a **inf**, **-inf**, or **NaN** result.

Example Model

- See the example model **"Derivative is infinite or NaN - Example 2"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Why the error occurs: The error message will occur when a **inf -inf** and **NaN** value is assigned to an Output signal that is in-turn used elsewhere in the simulation, such that results eventually influence a model state. Assigning an **inf** or **NaN** value to a continuous or discrete state derivative within the C-Script will also result in this error.

C-Scripts can return **inf**, **-inf**, or **NaN** values. Common examples include division by zero, inverse trigonometric functions, square roots of negative numbers, and numerical overflow. The C-Script compiler will issue no warning when an operation will return **inf** or **NaN** values, as these are valid return values.

Negative losses

This error message occurs when PLECS calculates negative losses of a device based on the thermal description and state of the circuit at the switching instant. Even if the data in the thermal description is consistent and only shows positive values, an extrapolation of the given data may lead to a negative loss value. This is typically the problem when the temperature of the device is very large or very small and the extrapolation of loss data based on the temperature value becomes negative.

Example Model

- See the example model **"Negative loss values for component"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Solution 1: Set zero loss values at the origin

Ensure that the thermal descriptions should have zero loss values at the origin, i. e. where the voltage or the current is zero.

Why the error occurs: If there are positive loss values at the origin, the extrapolation for negative voltage or current will lead to negative losses.

Solution 2: Use assertions for very large and very small temperatures

Use assertions inside the thermal descriptions to avoid negative losses when extrapolating for very large and very small temperatures.

Why the error occurs: The simulation will continue when the device is operating out of the safe operating area where normal loss data is meaningless.

Solution 3: Ignore error message or convert error message into a warning

If the thermal loss data is specified correctly or if the thermal characteristics is not of interest in a particular simulation scenario, the error message can be ignored or converted into a warning. This option is available under the **PLECS Parameters...** (PLECS Blockset) or **Simulation Parameters...** (PLECS Standalone) menu.

State-source dependency of components

This error occurs due to inconsistent definitions of states.

PLECS models physical systems using state-space equations of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

where x is a vector of state variables (i.e. inductor currents and capacitor voltages for electrical systems) and u is a vector of input variables (i.e. voltage and current sources for electrical systems).

A state/source dependency is created when there is a direct relation between a state and source quantity. For example, a capacitor that is connected directly in parallel with a voltage source is described by the equation:

$$0 = v_\mathrm{c} + v_\mathrm{src}$$

At the same time the capacitor voltage should be integrated from the equation:

$$\frac{\mathrm{d}v_\mathrm{c}}{\mathrm{d}t} = \frac{i_\mathrm{c}}{C}$$

This creates an inconsistent definition of v_c and leads to an error. The following figure shows the possible state/source dependencies for each physical domain:

Example Model

- See the example model **"State/source dependence in components - Example 2"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Once the state/source dependency has been created, for instance in the electrical domain, the solution to this problem can be:

Solution 1: Eliminate the state component

Eliminating the state component will eliminate the inconsistency.

For example, you can eliminate a capacitance in parallel if a voltage source has been introduced in the circuit to fix or control the voltage. The same can be applied in current/inductance cases.

Similar strategies can be applied for all other physical domains.

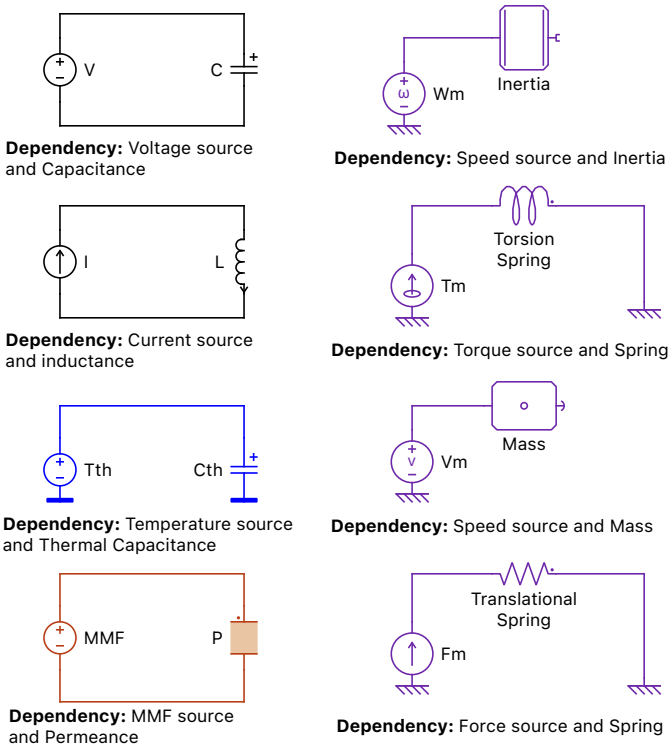


Fig. 3.31: State/source dependencies in physical simulation domains in PLECS

Solution 2: Break the dependency

Decouple the source and the state component when you cannot eliminate the state component.

For instance, a small resistance can be introduced in series with the voltage source. This allows to eliminate the state/source dependency and at the same time model a voltage source with a limited maximum power delivery. A similar approach can be followed by introducing a relatively large parallel resistance with the current source in order to break the inductance/current source dependency.

Similar strategies can be applied for all other physical domains.

Example Model

- See the example model **"State/source dependence in components - Example 3"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

Note

The resistance must be chosen carefully so that it does not influence the system dynamics too much or create numerical problems like stiffness.

Solution 3: Setting Inertia and Thermal capacitance to 0

Simply set the "Inertia" and "Thermal capacitance" parameters to **0**.

Why the error occurs: There are few, but very common, component arrangements where state/source dependencies can be created. For example, such a dependency is created when the shaft's speed of an electrical machine needs to be fixed by an external speed source, or the temperature of a heat sink needs to be fixed by using a temperature source.

Example Model

- See the example model **"State/source dependence in components - Example 1"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

The problem here is that the machine models have an internal inertia connected to the shaft's mechanical ports, and the heat sink component contains an internal thermal capacitance. Therefore, state/source dependencies are created (like the ones shown above).

Out of memory

This error occurs when there is not enough memory to run the entire PLECS simulation. Unfortunately, there is no way to get the simulation data out of any scopes prior to the simulation stopping.

Solution 1: Reduce number of scopes

The best way to avoid this error is to reduce the number of scopes. PLECS can allow for large models and long simulation times, but multiple scopes used in a model is the main memory sink.

Solution 2: Use the 64-bit version

You can use a 64-bit version of Matlab and PLECS. The 32-bit version of PLECS is limited to 3 GB of virtual memory by the operating system, while a 64-bit machine can allocate several terabytes of virtual address space.

Solution 3: Decrease the cache size limit

You can *de*crease the cache size limit in the PLECS preferences.

Solution 4: Use the correct solver settings

You can use a stiff solver when solving stiff systems. Also, set the **Max step size** parameter as large as possible.

Why the error occurs: The wrong solver settings force the use of unnecessarily small steps and thus increases the memory needed.

Solution 5: Limit the maximum number of samples stored per scope

You can limit the maximum number of samples stored per scope in the scope's preferences to the last N samples. Remember, though, that you cannot decimate the samples in the scope that are stored (i.e. store only 1 of N samples).

Solution 6: Set a large and fixed sample time in the scopes

You can set a large, fixed sample time in the Scope Parameter menu.

Solution 7: Exchange scope

You can use a To-File block instead of a scope.

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.

Web-Based Simulation Designer

↪ *Added in PLECS 4.9.*

Web-Based Simulation (WBS) allows a visitor of a web page to run PLECS simulations interactively in a web browser. A PLECS model together with a Web-Based Simulation design define how a web page visitor can interact with the WBS. The **Web-Based Simulation Designer** can be used to create WBS designs interactively. Please see the accompanying documentation within the WBS Designer.

To launch the WBS Designer from PLECS, select **Web-Based Simulation Designer** from the **Window** menu. The WBS Designer will be automatically downloaded before first use[2].

To uninstall the WBS Designer, refer to your operating system's instructions in the table below. To reinstall the WBS Designer, start by uninstalling it. Then, launch the WBS Designer from PLECS to install it again.

Table 3.4: Uninstalling WBS Designer

Platform	Instructions
Windows	Run `uninstall.exe` in folder `C:\Users\<USER>\AppData\Local\Plexim\PLECS\wbsdesigner`
macOS	Delete `PLECS WBS Designer.app` in folder `~/Library/Application Support/Plexim/PLECS/wbsdesigner`
Linux	Delete `PLECS WBS Designer.AppImage` in folder `~/.local/share/Plexim/PLECS/wbsdesigner`

[2] AppImage support is required for Linux operating systems.

4

Thermal Modeling

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

Thermal Concepts

Heat Sink Concept

The core component of the thermal library is an idealized Heat Sink (p. 738) depicted as a semitransparent box as in Fig. 4.1. 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.

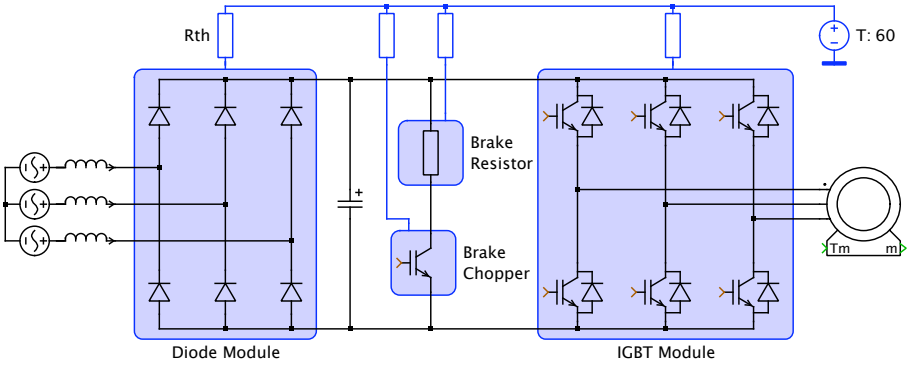


Fig. 4.1: Heat Sinks as Semitransparent Boxes

Heat conduction from one heat sink to another or to an ambient temperature is modeled with lumped thermal resistances and capacitances that are connected 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 (p. 623) or the Controlled Temperature block (p. 631).

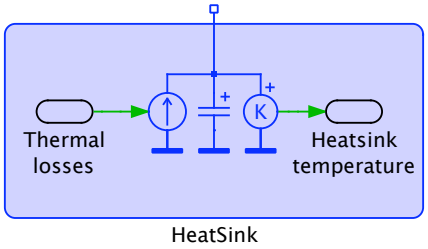


Fig. 4.2: Schematic Representation of a Heat Sink

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 “Thermal Description Parameter” (p. 179) and “Thermal Library” (p. 183) 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 $v = V_\mathrm{f} + R_\mathrm{on} \cdot i$.

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: $v = v_\mathrm{on}(i, T)$. You may also specify additional custom function arguments. This function is defined in the **Conduction Loss** tab of the thermal description as a 2D lookup table or a functional expression (see “Thermal Editor” (p. 185)).

A setting of $0\,\mathrm{V}$ for a single temperature and current value means no conduction losses. If you do not specify a thermal description in the device parameters,

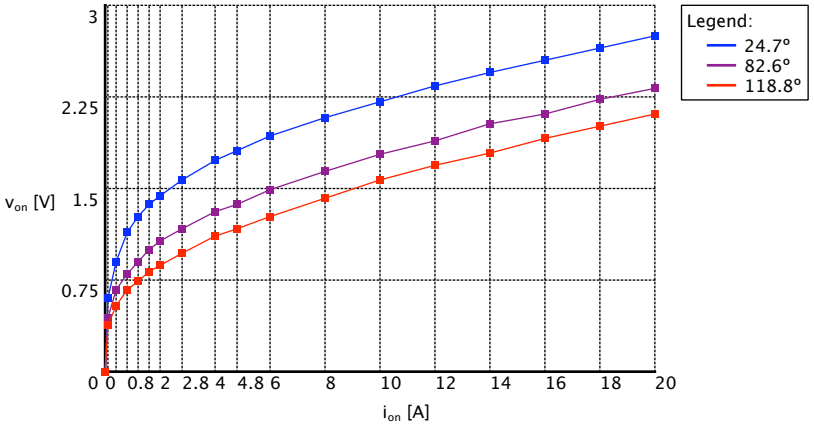


Fig. 4.3: Example on-state voltage characteristic of a semiconductor device

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 Fig. 4.4. 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.

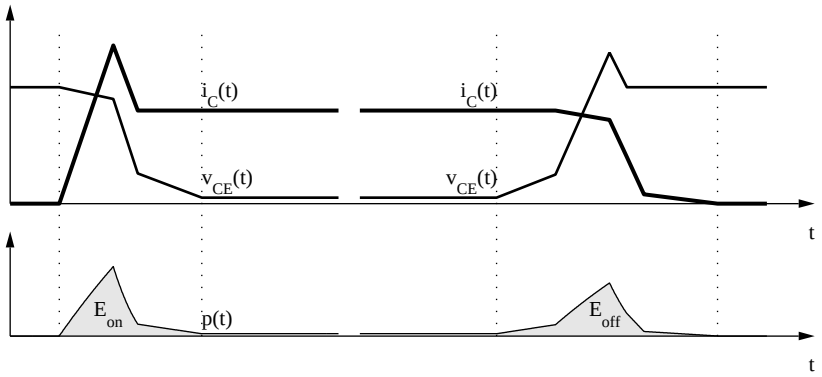


Fig. 4.4: Turn-on and turn-off behavior of a IGBT device during one switching cycle

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: $E = E_{\mathrm{on}}(v_{\mathrm{block}}, i_{\mathrm{on}}, T)$, $E = E_{\mathrm{off}}(v_{\mathrm{block}}, i_{\mathrm{on}}, T)$. 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 lookup tables or functional expressions (see “Thermal Editor” (p. 185)).

A setting of $0\,\mathrm{J}$ 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 lookup 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

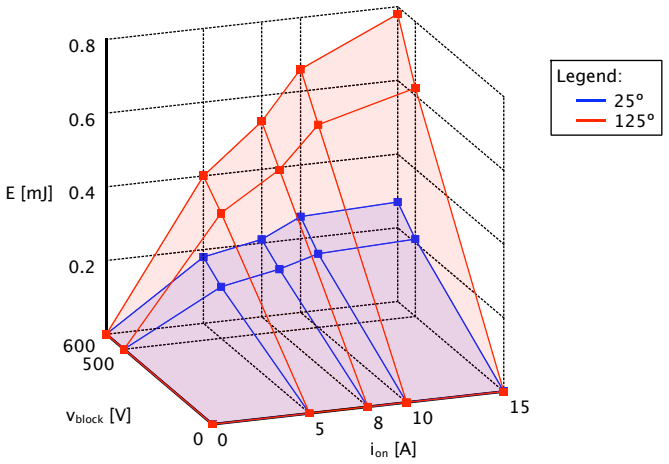


Fig. 4.5: Example switching energy characteristic of a semiconductor device

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 Standalone Parameters” (p. 135) and “PLECS Blockset Parameters” (p. 143)).

Supported Devices

The following semiconductor components implement this loss model:

- Diode (p. 641)
- Thyristor (p. 1024)
- GTO (p. 718)
- GTO (Reverse Conducting) (p. 720)
- IGBT (p. 746)
- IGBT with Diode (p. 752)
- IGCT (Reverse Blocking) (p. 758)
- IGCT (Reverse Conducting) (p. 760)
- MOSFET (p. 828)
- MOSFET with Diode (p. 832)

- TRIAC (p. 1080)

In addition, the Set/Reset Switch (p. 944) 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:

- Resistor (p. 905)
- Variable Resistor with Variable Inductor (p. 1114)
- Variable Resistor with Constant Inductor (p. 1111)
- Variable Resistor with Variable Capacitor (p. 1112)
- Variable Resistor with Constant Capacitor (p. 1110)

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 (p. 557). This latter block provides a thermal connection to the heat sink enclosing the parent subsystem block.

As an example the figure Fig. 4.6 shows the subschematic of the Diode with Reverse Recovery (p. 646). 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 (p. 630) 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

↪ *Added in PLECS 4.0.*

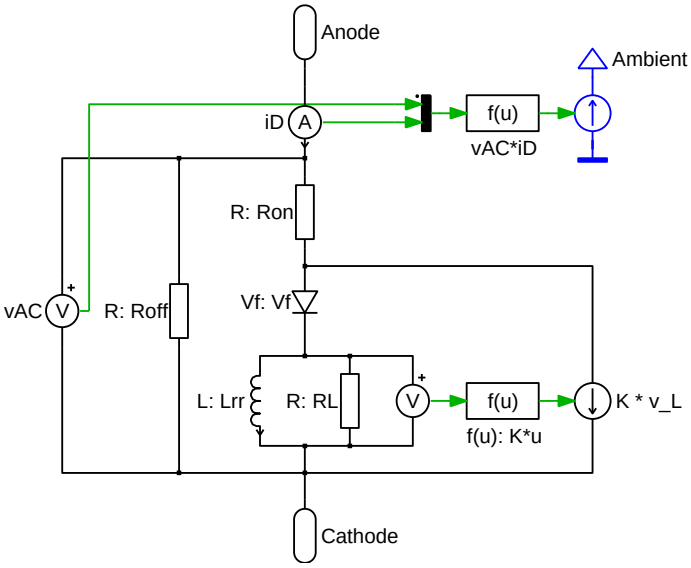


Fig. 4.6: Heat Sink to Ambient

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 (p. 1013) 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 (Fig. 4.7). 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.

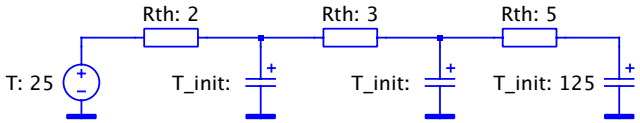


Fig. 4.7: Example thermal system

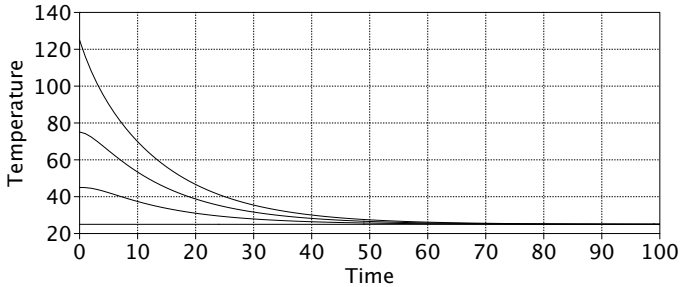


Fig. 4.8: Thermal behavior of the example thermal system shown in Fig. 4.7

Thermal Description Parameter

Most semiconductor components in PLECS have a parameter **Thermal description**. Also, masked subsystems can have mask parameter of type **Thermal** (see "Mask Dialog" (p. 98)). Thermal parameters can be used in two ways:

- to select a data sheet from the thermal library or
- to assign the value of a reference variable that is defined e.g. as a thermal mask parameter or in the base workspace.

Selecting Thermal Data Sheets

To select a data sheet from the thermal library, choose the menu entry **From library...**. This will open a submenu "Selecting a data sheet from a thermal library" that shows all data sheets that match the device type; e.g. in the dialog box of a diode only those data sheets appear that have their **Type** field set to `Diode`.

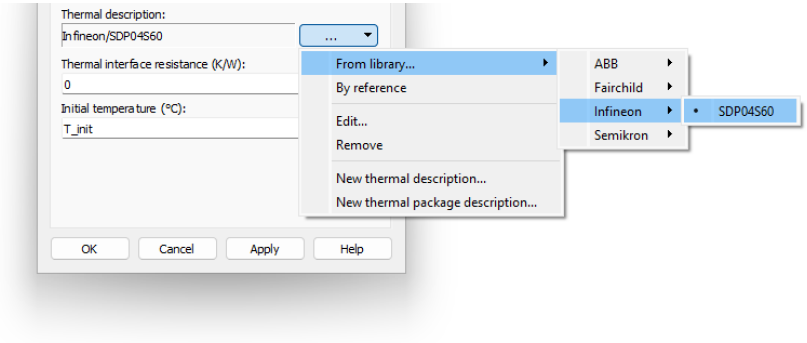


Fig. 4.9: Selecting a data sheet from a thermal library

If no data sheet is available, the menu entry is disabled. See section "Thermal Library" (p. 183) 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 be entered in the text field.

The reference variable can be the variable name of a **Thermal** mask parameter in the mask definition of a parent Subsystem (see "Mask Dialog" (p. 98)).

The reference variable can also be a string specifying a thermal description file. The string must begin with `file:` followed by the file path of the data sheet. It is possible to use an absolute file path to a thermal description file including the `.xml` extension, 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'
```

Thermal Modeling with Parallel Devices

In power electronic systems, semiconductor devices are often connected in parallel to share power dissipation and reduce the thermal stress on individual components. PLECS supports several approaches for modeling parallel semiconductor devices:

- **Multiple discrete components**: Multiple semiconductor components can be placed in parallel, each with its own electrical and thermal connection. This approach provides full electrical and thermal detail, but becomes increasingly complex as the number of parallel devices grows. Increased schematic block density, additional gate signal routing, and the need to maintain consistent parameterization across all paralleled devices increase the likelihood of configuration errors. This approach is therefore not recommended.
- **Custom masked subsystem**: You can create a masked subsystem (see "Masking Subsystems" (p. 84)) in which paralleling equations are applied in the initialization commands to scale device currents, losses, and thermal impedance used by the thermal model. This approach offers flexibility and can be useful for custom modeling workflows, but requires manual setup and maintenance.
- **Built-in parallel device support**: As of PLECS 5.0, the following semiconductor components include a **Number of parallel devices** parameter that automatically scales thermal losses while maintaining a single electrical device model. This approach is recommended for most applications as it provides accurate thermal scaling with minimal modeling effort.
  - Diode (p. 641)
  - MOSFET (p. 828)
  - MOSFET with Diode (p. 832)

Example Model

- See the example model **"Thermal Modeling with Parallel Devices"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

Note

The paralleling approaches above assume ideal current sharing between the parallel devices.

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" (p. 185)). 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" (p. 60)). 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` or `.plecs` extension) plus a suffix `_plecs`, e.g. `plSMPS_CCM_plecs` for model `plSMPS_CCM.mdl` (`plSMPS_CCM.plecs`).

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

To create a new thermal data sheet, choose **Thermal description...** or **Thermal package description...** from the **File > New...** 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.

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 create a new data sheet, choose **Thermal description...** or **Thermal package description...** from the **File > New...** menu. In order to access the data sheet in a PLECS model, you must save it in a directory on the thermal search path. See section “Thermal Library” (p. 183) for details of the structure of the thermal library.

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...**.

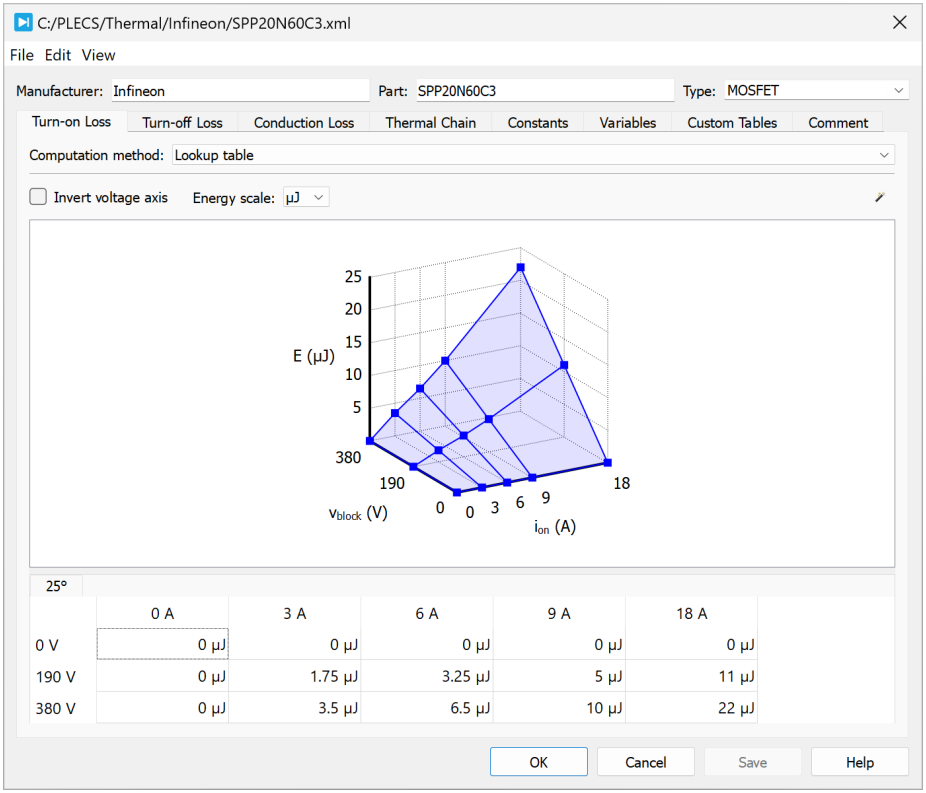


Fig. 4.10: Thermal Editor window

The top row of the Thermal Editor window shows three input elements:

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

Type
This string is used by the **Thermal Description** parameter of semiconductor devices and thermal mask parameters to filter matching thermal descriptions (see “Selecting Thermal Data Sheets” (p. 179)).

Thermal Description for a Single Device

When you edit the thermal description of a single device, the editor shows the following tabs:

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” and “Editing Conduction Losses”.

Thermal Chain

On this tab you define the thermal impedance between the junction and the case of the device. See “Editing the Thermal Equivalent Circuit” (p. 188).

Constants

On this tab you define custom constants that can be used in the loss formulae for switching and conduction losses.

Variables

On this tab you define custom variables that can be used in the loss formulae for switching and conduction losses. Custom variables can be edited in the parameter dialog of the component using the thermal description (see “Specifying custom variable values” (p. 192)). On this tab you can also specify hard limits both for your custom variables 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” (p. 194).

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” (p. 191)). You may also reference custom lookup tables using the function `lookup` (see “Adding Custom Lookup Tables” (p. 193)).

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” (p. 194). On

using the import wizard for constructing lookup table data from vendor plots, see “Importing Data from Graphical Datasheets” (p. 199).

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” (p. 191)). You may also reference custom lookup tables using the function `lookup` (see “Adding Custom Lookup Tables” (p. 193)).

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.

Gate Dependent Conduction Losses

↪ *Added in PLECS 4.6.*

A MOSFET can conduct current in both directions, and so the conduction losses of a MOSFET with integrated anti-parallel diode depend on the gate signal when the current flows in reverse direction. To account for this effect, you can provide two separate conduction loss definitions, one of which is used when the gate signal is non-zero, and the other, when the gate signal is zero.

To create separate tabs that define the conduction losses with respect to the gate signal, select the **MOSFET with Diode** device type, right-click on the tab bar and select **Use gate dependent conduction losses** from the context menu.

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. Due to the simple structure of these thermal equivalent circuits, these networks may simply be referred to as *thermal (RC) chains*. They can be edited on the **Thermal Chain** tab of the thermal editor. The thermal equivalent circuit is specified in either Cauer or Foster form.

The structure of a Cauer network is shown in Fig. 4.12. In the thermal editor, the number of chain elements n and the values for R_i in $(\mathrm{K/W})$ and C_i in $(\mathrm{J/K})$ for each chain element need to be entered.

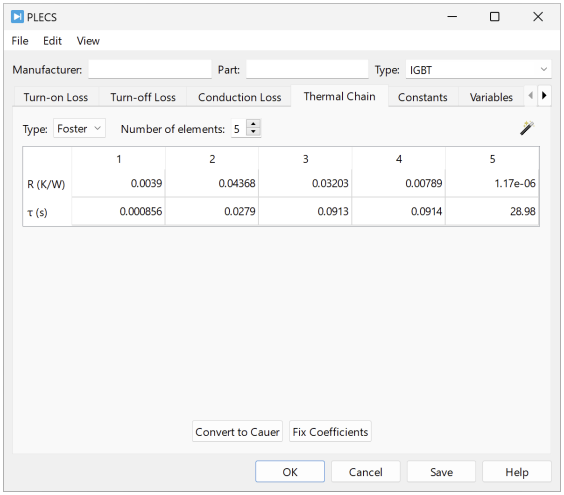


Fig. 4.11: Thermal Chain Tab of the Thermal Editor

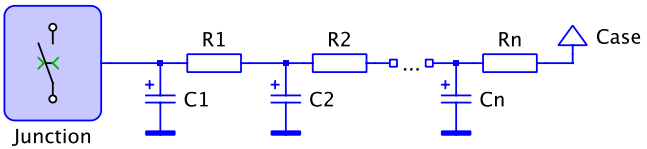


Fig. 4.12: Cauer network

Fig. 4.13 illustrates the structure of a Foster network. In the thermal editor, the number of chain elements n and the values for R_i in $(\mathrm{K/W})$ and τ_i in (s) for each chain element need to be entered. Foster networks can be converted to Cauer networks by pressing the **Convert to Cauer** button.

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 meaningful 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.

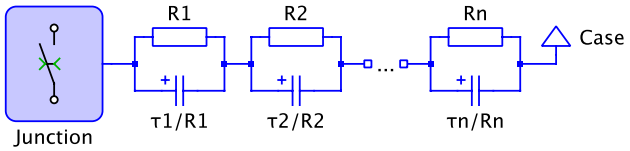


Fig. 4.13: Foster network

Detecting and Fixing Parameterization Problems

↪ *Added in PLECS 4.9.*

Sometimes the conversion from the Foster to the Cauer type can lead to unreasonably high thermal capacitance values. This is because some Foster networks contain elements with a large time constant (τ) and small thermal resistance or different elements with very similar time constants. This is typically the result of a non-optimal fit of the thermal impedance response curve obtained by measurement or finite element simulation. Because there is a unique transformation between Foster and Cauer network coefficients for circuits with constant ambient temperature, these non-optimal coefficients will also be reflected in the Cauer chain. PLECS is able to fix Foster coefficients (and therefore Cauer coefficients) in the following situations:

- A Foster chain features multiple elements with a similar τ value. In this case the number of Foster elements can be reduced. For identifying similar τ values, an internal tolerance is used.
- A Foster chain features elements that are barely visible in the impedance curve but have a large impact on the thermal capacitance coefficients when converted to the Cauer type. These elements can be eliminated by an overall fix which uses a model order reduction algorithm.

If PLECS detects one of these two problems, you can press the **Fix Coefficients** button. Otherwise it is grayed out. Clicking this button opens a dialog that displays a preview of the fixed Foster coefficients and shows the transient thermal impedance curves before and after fix (see Fig. 4.14).

In this particular example of the Foster coefficients of an IGBT device, a pair reduction (the two elements with close $\tau = 0.0913\,\mathrm{s}$ and $\tau = 0.0914\,\mathrm{s}$, as shown in Fig. 4.11 of the **Thermal Chain** tab, have been combined into a single element) and an overall fix (addressing the last Foster element with $\tau = 28.98\,\mathrm{s}$) have reduced the original five-element thermal chain to a three-element one. The importance of this fix can be understood by comparing the total capacitance of the two Cauer-type networks derived from the Foster coefficients before

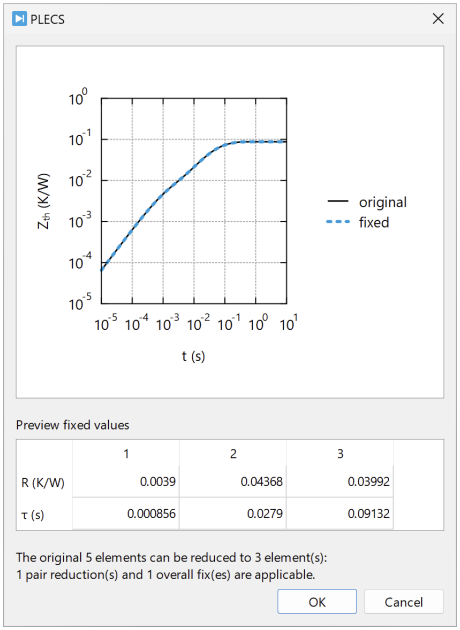


Fig. 4.14: Preview of Fixed Foster Coefficients after Conversion Issues

and after the fix. The original coefficients correspond to a total heat capacity of $C_{\text{tot}}^{\text{original}} = 25.1\,\text{MJ/K}$, while the fixed coefficients result in $C_{\text{tot}}^{\text{fixed}} = 4.96\,\text{J/K}$. These figures become even more tangible when calculating the equivalent mass of copper required to realize such a heat capacity. Assuming a specific heat capacity of copper of $385\,\text{J/(kg K)}$, the original case would correspond to a copper mass of about 65 tons (!), while the fixed case would correspond to only $12.9\,\text{g}$ of copper. In this way, the fix feature can help to avoid massively delayed heat flows through the final Cauer-type network, while at the same time preserving the shape of the transient thermal impedance curve used to derive the original coefficients.

Adding Custom Variables

↪ *Added in PLECS 4.6.*

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 (see Fig. 4.15).

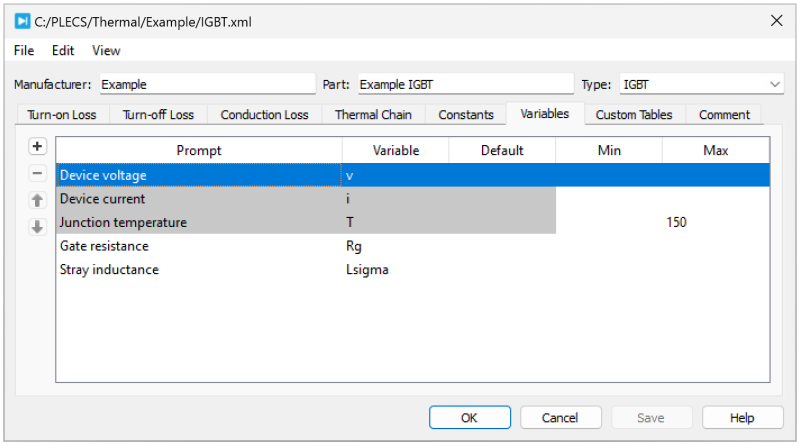


Fig. 4.15: Thermal Variables Tab

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.

A custom variable is defined by a **Prompt**, which should provide a brief description of the purpose of the 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. You may specify a **Default** value that is used if the end user of the thermal description does not provide a value.

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 Standalone Parameters” (p. 135) and “PLECS Blockset Parameters” (p. 143)).

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 in Fig. 4.16.

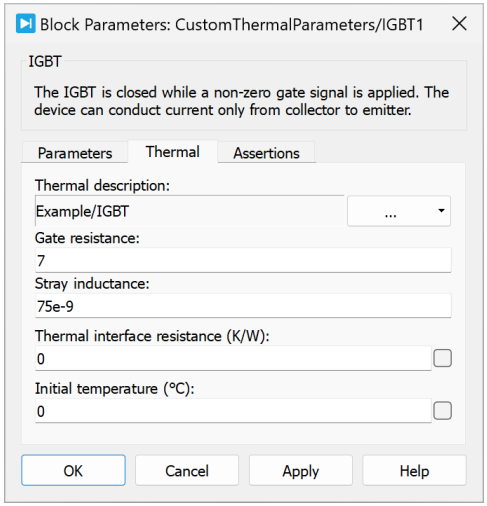


Fig. 4.16: Example Dialog for Custom Thermal Parameters

Instead of static values that remain constant during a simulation you may also specify the label of a Signal Goto block (p. 948) 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.

Example Model

- See the example model **"Custom Variables in Thermal Descriptions"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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 `Rg` 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** (on macOS: **Cmd+C**) and **Ctrl+V** (on macOS: **Cmd+V**). 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 $0.23\,\mathrm{J}$ to $0.23\,\mathrm{mJ}$, 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 $0.23\,\mathrm{J}$ to $230\,\mathrm{mJ}$, use the **Energy scale** drop box at the top right.

Thermal Package Description

↪ *Added in PLECS 4.6.*

A thermal package description is used to describe the thermal behavior of a power module that consists of multiple semiconductor chips. It contains the loss descriptions of the individual semiconductors as well as a structural model of the thermal coupling between the semiconductors and the package case.

The concept is illustrated using the example of a T-type inverter module shown in Fig. 4.17.

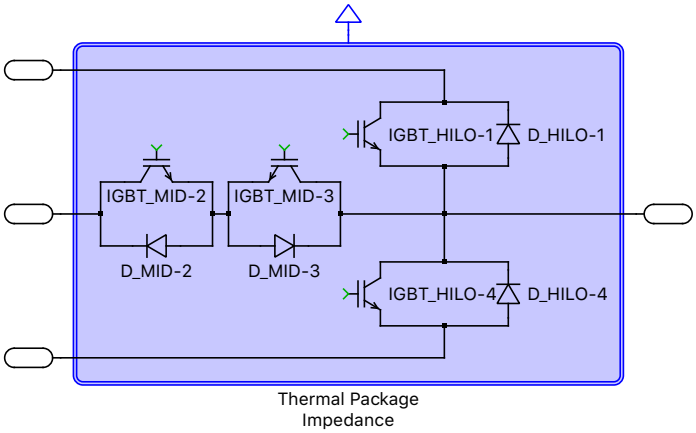


Fig. 4.17: Example of a T-type inverter power module consisting of multiple semiconductor chips

The module contains two different types of IGBTs and two different types of diodes because the high-side and low-side semiconductors (1 and 4) need to block higher voltages than the mid-point semiconductors (2 and 3). The package description therefore needs to contain four semiconductor device descrip-

tions and a thermal model for the coupling between eight semiconductors and the module case.

Editing a Thermal Package Description

When you edit a thermal package description, the editor shows the following tabs:

Device Types

On this tab you define the different semiconductor types that the package contains. The left hand side shows the list of device types, the right hand side shows tabs to define the switching and conduction losses and the thermal impedances of the individual device types. Each device type has a *name filter* that is used to associate the thermal description for this device type with the individual components in the module.

For the example module, the thermal package description would contain four device types with name filters `IGBT_HILO-*`, `IGBT_MID-*`, `D_HILO-*` and `D_MID-*` as shown in Fig. 4.18.

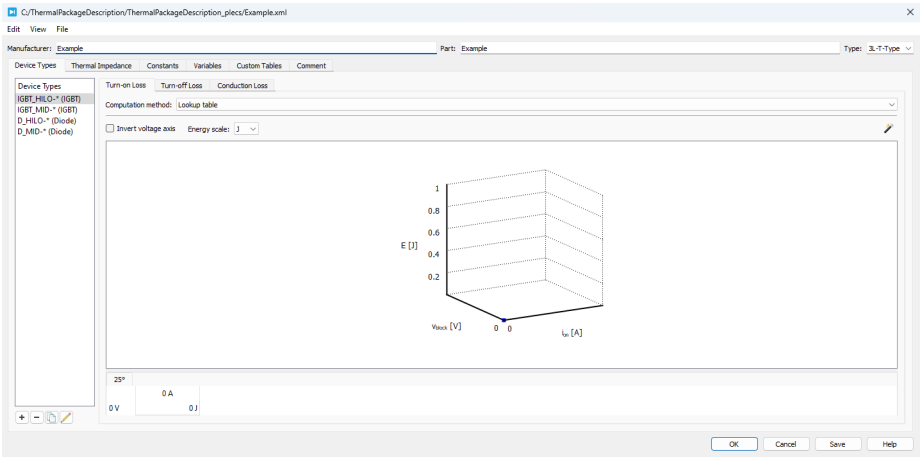


Fig. 4.18: Thermal Package Description window containing four device types

Thermal Impedance

↪ *Changed in PLECS 4.8:* Thermal coupling can now be characterized with an impedance matrix.

On this tab you define the thermal impedance between the individual semiconductor devices and the case of the package. This definition considers devices to be heat sources, the case to be a temperature source and the impedance to be an LTI system with or without internal states.

The heating of an IGBT or diode device inside the T-type inverter module not only increases the temperature directly at the heat source device but also causes the temperature of all other devices to rise. A straightforward way to describe such thermal coupling effects is to model the temperature rise of one device by summing the individual contributions to that temperature rise from all the heat sources contained in the package. For the eight devices of the T-type inverter module such a thermal *linear superposition model* can be formulated in matrix notation as

$$\begin{bmatrix} \Delta T_{\text{IGBT_HILO}-1} \\ \Delta T_{\text{D_HILO}-1} \\ \vdots \\ \Delta T_{\text{D_HILO}-4} \end{bmatrix} = \begin{bmatrix} Z_{1,1} & Z_{1,2} & \cdots & Z_{1,8} \\ Z_{2,1} & Z_{2,2} & \cdots & Z_{2,8} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{8,1} & Z_{8,2} & \cdots & Z_{8,8} \end{bmatrix} \cdot \begin{bmatrix} Q_{\text{IGBT_HILO}-1} \\ Q_{\text{D_HILO}-1} \\ \vdots \\ Q_{\text{D_HILO}-4} \end{bmatrix}$$

A single component of the impedance matrix describes a thermal impedance that relates a heating power to a temperature difference. For example, heating up the second device with the heating power $Q_{\text{D_HILO}-1}$ contributes also to increasing the temperature difference $\Delta T_{\text{IGBT_HILO}-1} = T_{\text{IGBT_HILO}-1} - T_{\text{case}}$ of the first device, which is captured in the transient thermal impedance $Z_{1,2}(t)$. In PLECS, $Z_{1,2}(t)$ is described by a Foster or Cauer thermal chain that can be defined by clicking on a matrix element and setting type, number of elements and chain parameters as exemplified in the following screenshot.

From the provided impedance matrix, PLECS will then transform all thermal chains to the Cauer type and generate the matrices of a state-space model

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

where $\mathbf{u}$ is a vector comprising the device heat sources and case temperature source(s), $\mathbf{y}$ is a vector of the same length comprising the device temperatures and the heat flow *out of* the case and $\mathbf{x}$ is the vector of internal states.

The corresponding numerical representation of the state-space model can be displayed by clicking on the **Edit State-Space Matrices...** button at the bottom right. In the opened window (shown in Fig. 4.20 on p. 199), it is then possible not only to view the generated state-space matrices, but also to make

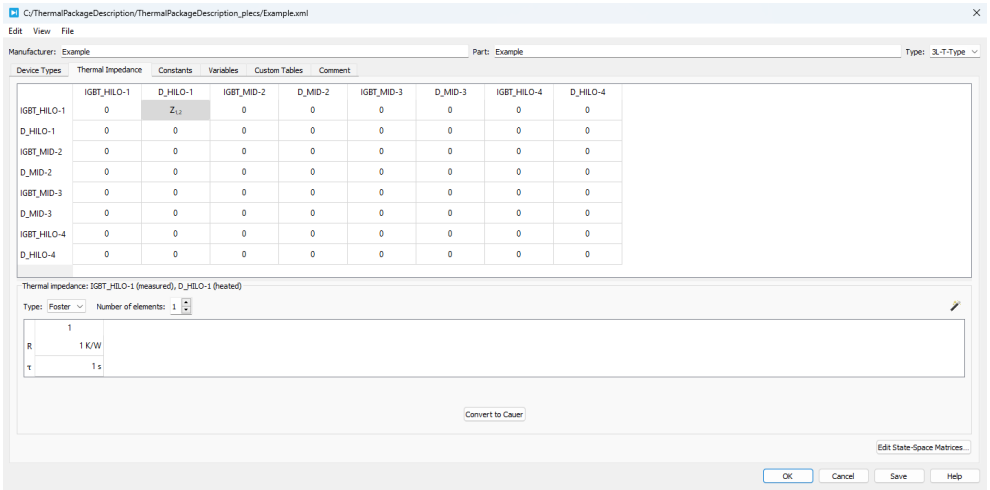


Fig. 4.19: Thermal impedance matrix

changes to them that go beyond the linear superposition model. On the left-hand side you specify the number of internal states, the number of devices in the package, the width of the case connection and the device names which must match the names of the actual devices in the module subsystem. On the right-hand side you enter the numerical values of the state-space matrices. Note that direct changes made to the state-space matrices are not converted back to the thermal impedance matrix when you return to the linear superposition model by clicking the **Edit Impedance Matrix...** button at the bottom right.

Constants, Variables, Custom Tables, Comment

Constants, variables and custom tables are shared by all device types in a package description. Please see section “Thermal Description for a Single Device” (p. 186) for a description of these tabs.

Example Model

- See the example model **“Thermal Coupling”**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

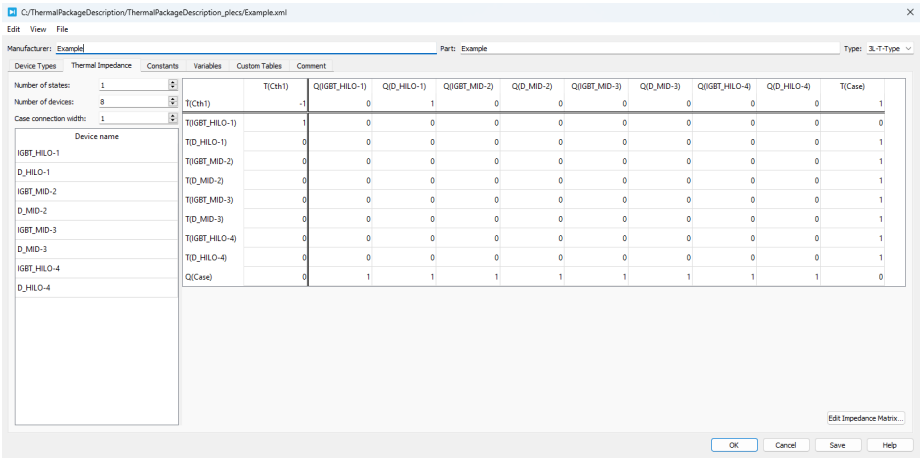


Fig. 4.20: Thermal impedance state-space matrix

Using a Thermal Package Description

A thermal package description is typically used in conjunction with a masked subsystem that defines the *electrical* behavior of the module. The subsystem mask defines a **Thermal** parameter with a **Device type filter** that matches the type of the thermal package description (see also “Mask Dialog” (p. 98)).

The mask variable that is associated with this thermal parameter (here: `thermal`) is then referenced *as is* in the **Thermal description** parameters of the individual semiconductor devices in the subsystem. The component names (e.g. `IGBT_HILO-1` or `IGBT_HILO-4`) are matched against the name filters (e.g. `IGBT_HILO-*`) to extract the appropriate loss definitions and thermal impedance for the semiconductor device.

The thermal impedance between the devices and the module case is modeled by the Thermal Package Impedance component (p. 1018) that is placed on top of the semiconductor devices similar to a heat sink. This component also has a parameter **Thermal description** that expects the thermal mask variable `thermal` to extract the definition of the thermal network from it.

This is illustrated in Fig. 4.22 on p. 201.

Importing Data from Graphical Datasheets

↪ *Added in PLECS 4.3.*

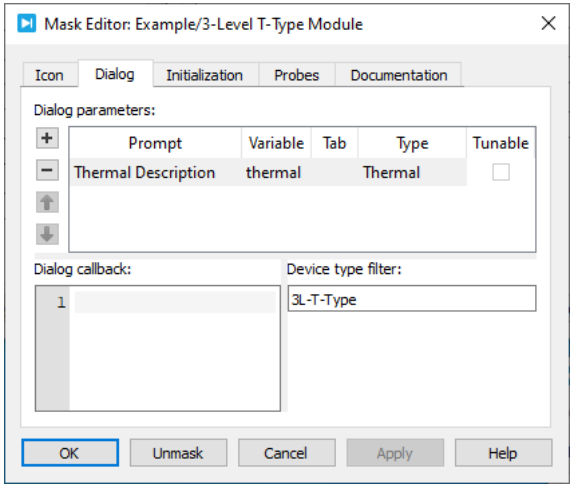


Fig. 4.21: Dialog tab of the thermal package description mask

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.

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 an image file to the empty wizard area or click on the appropriate underlined text 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 or press **Ctrl+V** (on macOS: **Cmd+V**) 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 (A) and the y-axis, in joules (J); for conduction losses, the x-axis is expected to be in amperes (A) and the y-axis, in volts (V). If the dimensions in the image are swapped, you can flip the image

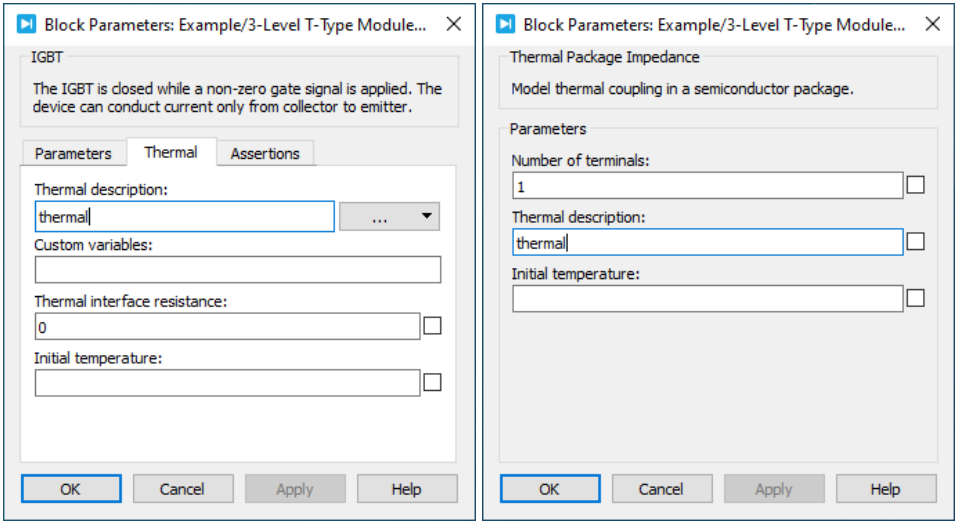


Fig. 4.22: Modeling of thermal impedance between device and module case by means of Thermal Package Impedance

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 the 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

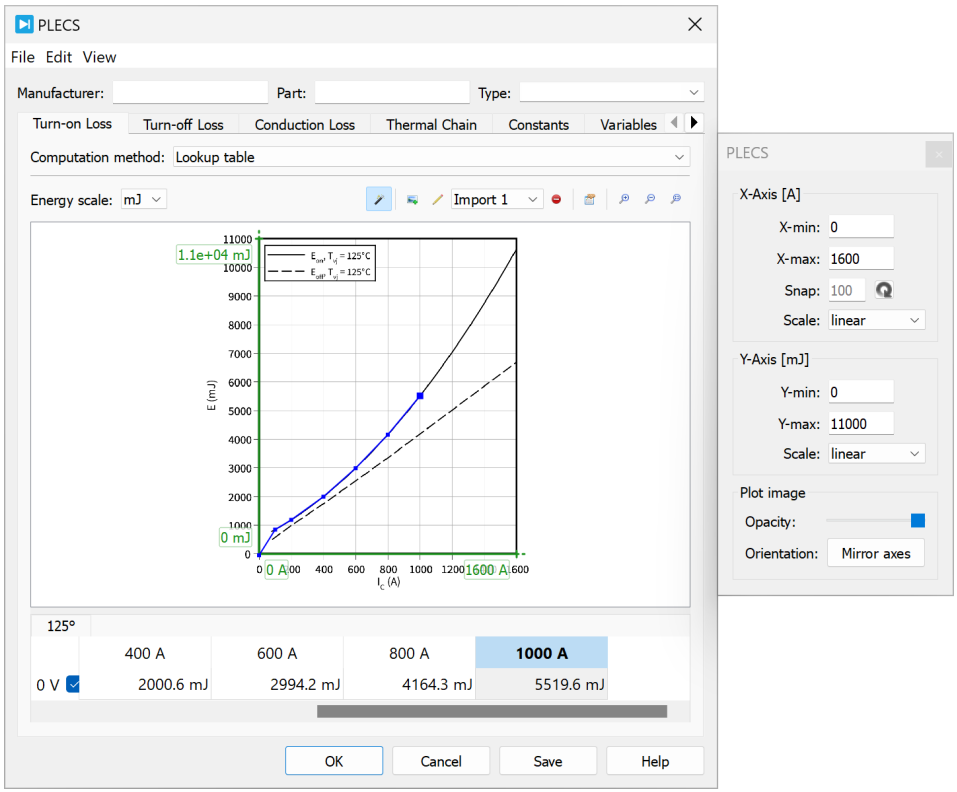


Fig. 4.23: Importing the turn-on loss characteristic of a device using the Import Wizard feature

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 lookup 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** (on macOS: **Cmd**) key 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** (on macOS: **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 lookup 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.

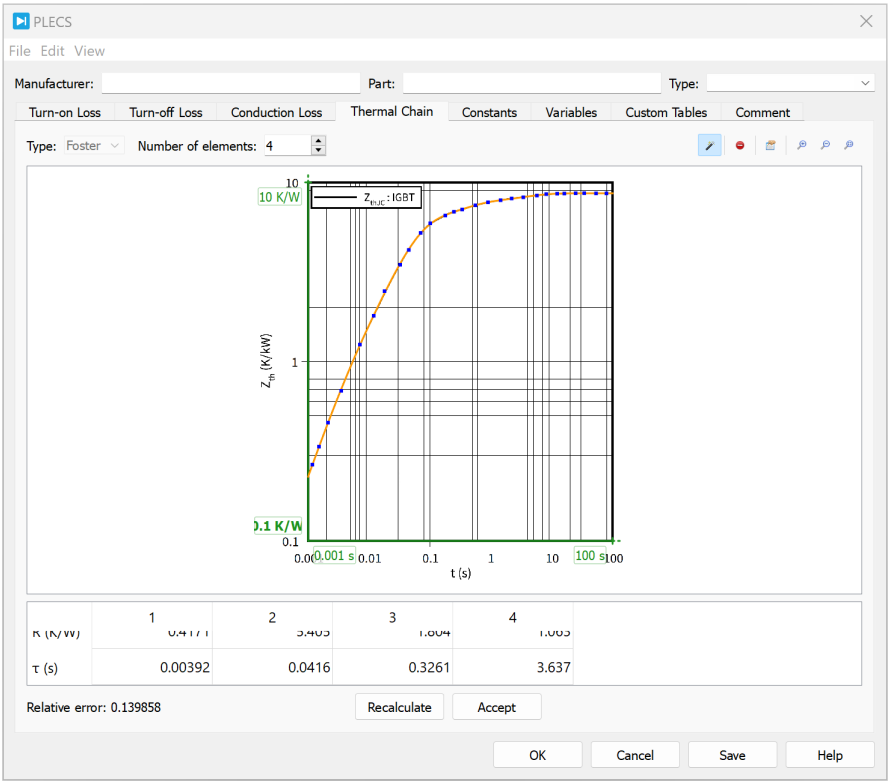


Fig. 4.24: Importing and fitting the thermal impedance characteristic of a device using the Import Wizard feature

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 Fig. 4.25.

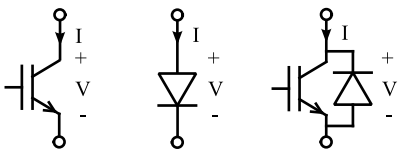


Fig. 4.25: Voltage and current polarity of single semiconductor switch, diode and semiconductor switch with diode

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 Fig. 4.26. 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.

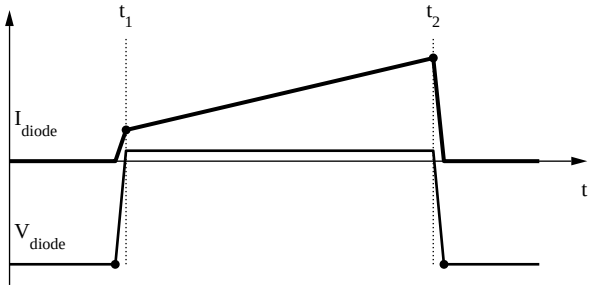


Fig. 4.26: Diode voltage and current during switching

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.

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 following 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	-	+	+	+	+	+	+	-

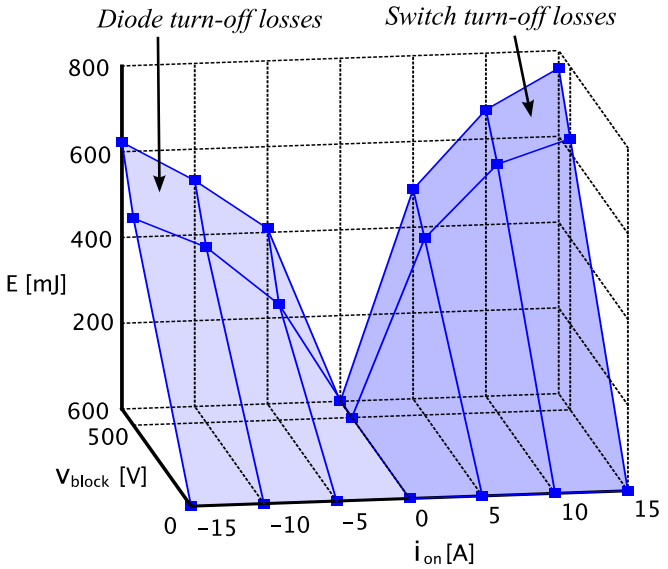


Fig. 4.27: Turn-off loss lookup table for semiconductor switch with diode

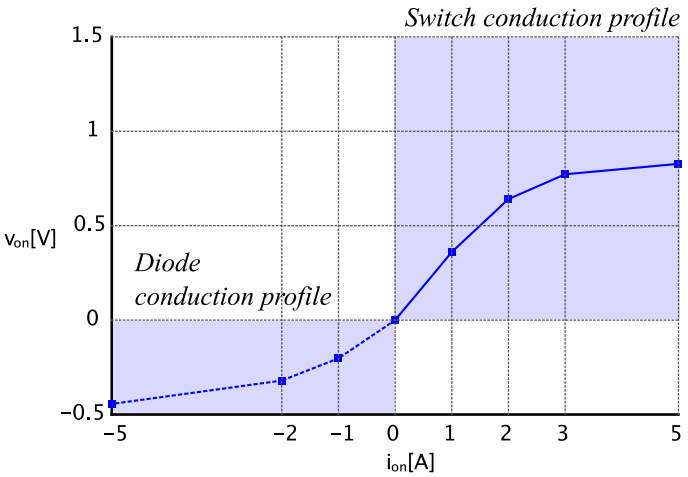


Fig. 4.28: Conduction loss profile for semiconductor switch with diode

5

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 (p. 1126), cores (p. 788) and air gaps (p. 555) are provided in the Magnetics Library. The available core models include saturation (p. 928) and hysteresis (p. 740). Frequency dependent losses can be modeled with magnetic resistances (p. 798). 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. Fig. 5.1 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 magnetizing inductance and R_{fe} the iron losses. The copper resistances of the windings are modeled with R_1 and R_2.

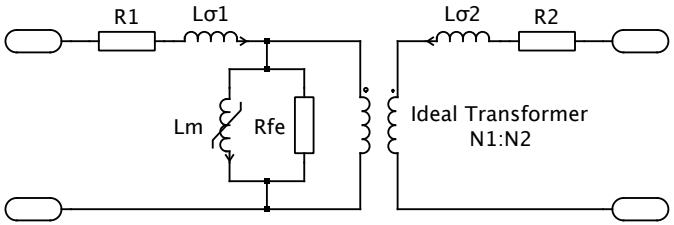


Fig. 5.1: Transformer implementation with coupled inductors

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) $\mathcal{F}$ is regarded as analogous to voltage and the magnetic flux Φ as analogous to current. As a consequence, magnetic reluctance of the flux path $\mathcal{R}$ corresponds to electrical resistance:

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

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

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

$$v = N\frac{\mathrm{d}\Phi}{\mathrm{d}t}$$

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

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:

$$\Phi = \frac{1}{N}\int v\,\mathrm{d}t$$

The schematic in Fig. 5.2 outlines a possible implementation of the magnetic interface in PLECS.

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.

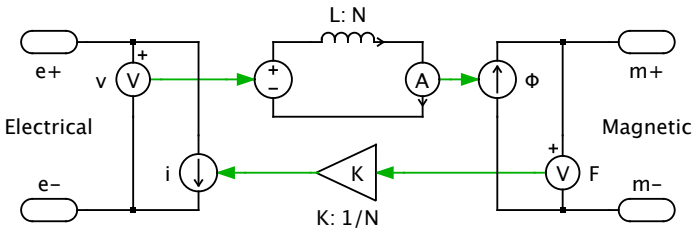


Fig. 5.2: Implementation of magnetic interface

- 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 $\mathcal{F}$ 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:

$$\dot{\Phi} = \mathcal{P}\,\frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t}$$

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:

$$\mathcal{P} = \frac{1}{\mathcal{R}} = \frac{\mu_0 \mu_\mathrm{r} A}{l}$$

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

Magnetic resistors (analogous to electrical resistors) can be used in the magnetic circuit to model losses. They can be connected in series or in parallel to a

permeance component, depending on the nature of the specific loss. The energy relationship is maintained as the power

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

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.

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

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

Because a winding converts through-quantities ($\dot{\Phi}$ resp. i) in one domain into across-quantities (v resp. $\mathcal{F}$) in the other domain, it can be implemented with a gyrator, in which N is the gyrator resistance R. Fig. 5.3 shows the symbol for a gyrator and a possible implementation in PLECS.

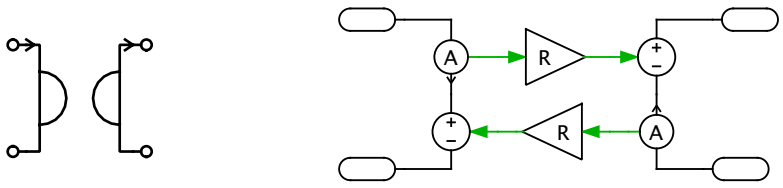


Fig. 5.3: Gyrator symbol and implementation

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 (p. 1126), constant (p. 795) and variable (p. 1103) permeances as well as magnetic resistors (p. 798). By connecting them according to the physical structure the user can create equivalent circuits for arbitrary magnetic components. The two-winding transformer from Fig. 5.1 on p. 210 will look like the schematic in Fig. 5.4 when modeled in the magnetic domain.

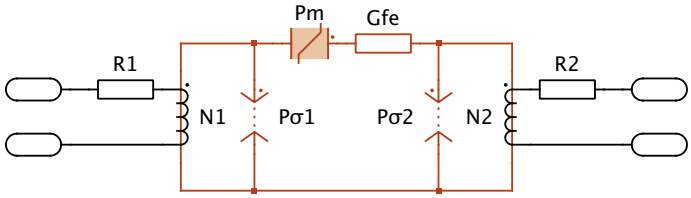


Fig. 5.4: Transformer implementation in the magnetic domain

$\mathcal{P}_{\sigma 1}$ and $\mathcal{P}_{\sigma 2}$ represent the permeances of the leakage flux path, $\mathcal{P}_{\mathrm{m}}$ 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.

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:

$$\dot{\Phi} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\mathcal{P}\cdot\mathcal{F}\right) = \mathcal{P}\cdot\frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t} + \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{P}\cdot\mathcal{F}$$

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

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

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

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

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

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

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

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

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

$$\begin{aligned}\dot{\Phi} &= \frac{\mathrm{d}\Phi}{\mathrm{d}t} \\ &= \frac{\mathrm{d}\Phi}{\mathrm{d}\mathcal{F}} \cdot \frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t} \\ &= \mathcal{P}_{\mathrm{diff}} \cdot \frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t} \quad ,\end{aligned}$$

the control signal is:

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

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 Langevian equation for bulk magnetization without interdomain coupling is used, which is referred to as the $\coth$ function:

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

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

$$\frac{\mathrm{d}B}{\mathrm{d}H} = B_{\mathrm{sat}} \left(\frac{\tanh^2 (H/a) - 1}{a \tanh^2 (H/a)} - \frac{a}{H^2} \right) + \mu_{\mathrm{sat}}$$

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

References

S. El-Hamamsy and E. Chang, “Magnetics modeling for computer-aided design of power electronics circuits,” in *Power Electronics Specialists Conference*, vol. 2, pp. 635-645, 1989.

R. W. Buntenbach, “Improved circuit models for inductors wound on dissipative magnetic cores,” in *Proc. 2nd Asilomar Conf. Circuits Syst.*, Pacific Grove, CA, Oct. 1968, pp. 229-236 (IEEE Publ. No. 68C64-ASIL).

R. W. Buntenbach, “Analogs between magnetic and electrical circuits,” in *Electron. Products*, vol. 12, pp. 108-113, 1969.

D. Hamill, “Lumped equivalent circuits of magnetic components: the gyrator-capacitor approach,” in *IEEE Transactions on Power Electronics*, vol. 8, pp. 97-103, 1993.

D. Hamill, “Gyrator-capacitor modeling: A better way of understanding magnetic components,” in *APEC Conference Proceedings* pp. 326-332, 1994.

6

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.

Table 6.1: 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

Example Model

- The example model **"Analogies Between the Electrical and Mechanical Domain"** shows the analogy of mechanical masses and electrical capacitors and how they behave in simulation.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Flanges and Connections

The two mechanical subdomains use separate connectors: a translational flange) and a rotational flange C. 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 in Fig. 6.1 are equivalent:

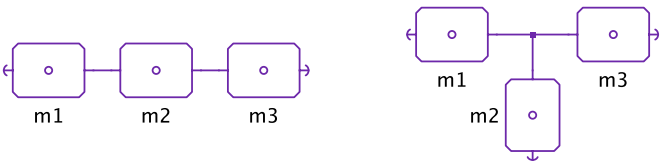


Fig. 6.1: Equivalent connections of three translational bodies

Force/Torque Flows and Sign Conventions

As the table of electrical and mechanical analogies (Tab. 6.1 on p. 217) 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 in Fig. 6.2 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.

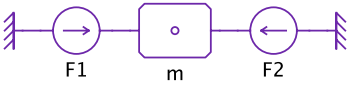


Fig. 6.2: Mass and two forces

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 in Fig. 6.3 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 ($\mathrm{d}x_0 = 0$) but its equilibrium displacement x_0 is not specified.

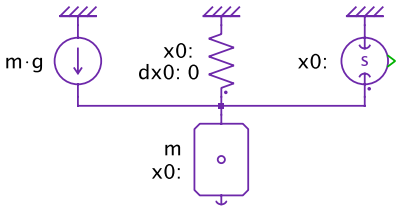


Fig. 6.3: Spring and mass

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, ω_1^- and ω_2^- are the two angular speeds prior to the collision and ω^+ 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:

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

This is demonstrated using the simple example shown in Fig. 6.4 consisting of two bodies with the same inertia $J = 1\frac{\mathrm{kg\cdot m^2}}{\mathrm{rad^2}}$(see the example model below). One initially rotates with $\omega_1^- = 1\frac{\mathrm{rad}}{\mathrm{s}}$ while the other is stationary $\omega_2^- = 0$. There is no friction or external torque acting on the bodies. When the clutch engages at $t = 1\,\mathrm{s}$, the two bodies immediately rotate with the same speed $\omega^+ = .5\frac{\mathrm{rad}}{\mathrm{s}}$ and the total kinetic energy of the system reduces instantaneously from $.5\,\mathrm{J}$ to $.25\,\mathrm{J}$.

Example Model

- See the example model **"Inelastic Collisions"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

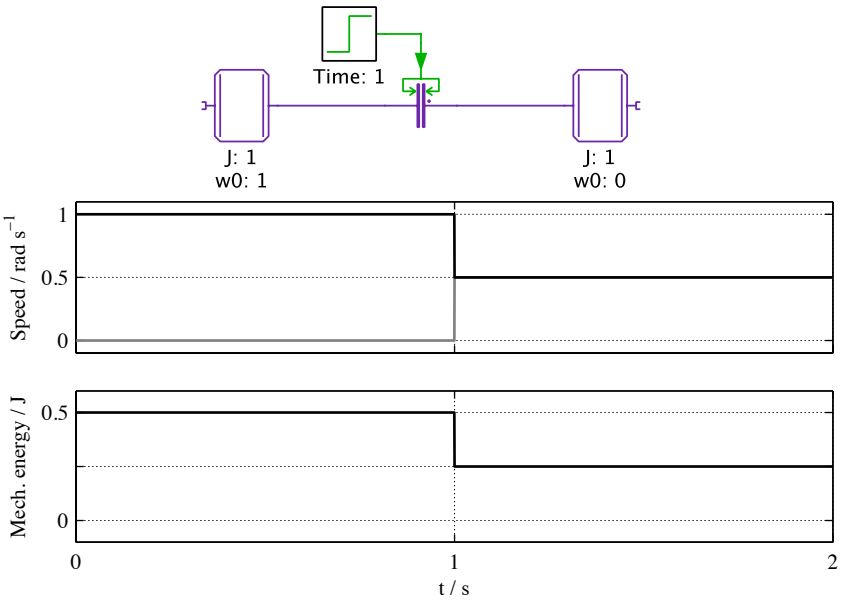


Fig. 6.4: Inelastic collision with ideal clutch

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 assumed to have a certain elasticity and damping coefficient. The corresponding schematic and plots are shown in Fig. 6.5; 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.

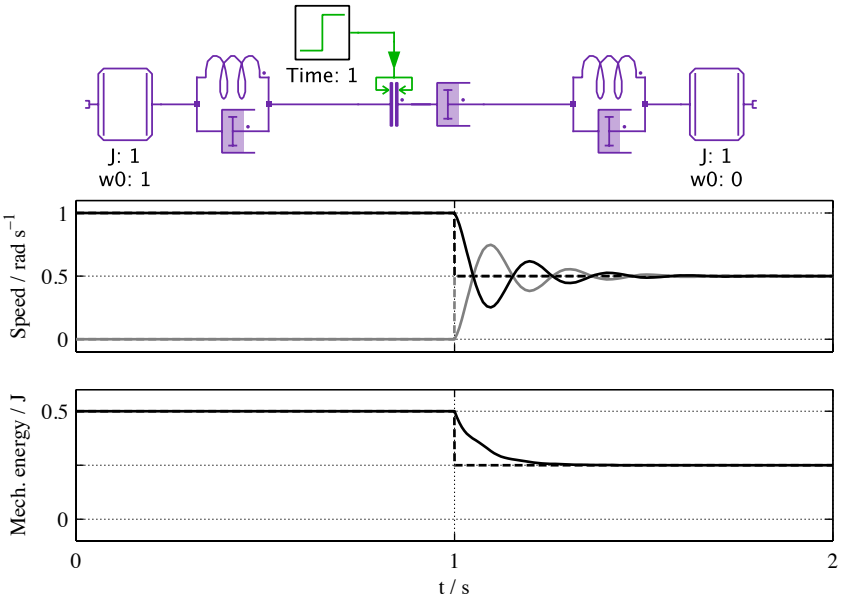


Fig. 6.5: Inelastic collision with non-ideal clutch and elastic shafts

7

Analysis Tools

PLECS offers several analysis tools for efficiently examining power electronic systems beyond plain time-domain simulation. These tools help obtain steady-state operating points, frequency responses, and other pertinent system characteristics.

Steady-State Analysis

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

Algorithm

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

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

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

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

The above problem can be solved iteratively using

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

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 $\mathbf{J}$:

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

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 $rtol$:

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

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 convergence. Normally, however, the overall performance will be much higher than calculating the full Jacobian matrix.

The calculation method is controlled by the parameter **Jacobian calculation** (see “Steady-State Analysis” (p. 238)).

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

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

i.e. the time derivative of the vector of state variables x. Since no simulation has to be performed to compute f, the full Jacobian 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 Sweep

The AC Sweep 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.

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. Barrado, “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.

Frequency Response

↪ *Added in PLECS 5.0.*

The Frequency Response Analysis (PLECS Standalone only) computes the frequency response of a system at discrete analysis frequencies by sequentially applying sinusoidal perturbations. The analysis is executed by following these steps:

1. A transient simulation is used to drive the system into steady-state. The length of the simulation is determined by the **Initialization time span** parameter T_{init}. This is used to define the operating point at which the frequency response will be calculated.

2. Transient simulations are executed in parallel for every discrete frequency f_k. All simulations start from the steady-state point reached in the previous step. The length of the perturbed simulations is defined as $T_s + T_c$, where T_s is the **Settling time span** parameter, and $T_c = N * 1/f_k$ is the number of **Extraction cycles** parameter times the period of the sinusoidal perturbation. An additional transient simulation with no perturbation is also executed.

3. The Fourier analysis is computed for each discrete perturbation frequency $\omega_k = 2\pi f_k$, of T_c period in the perturbation and response signals of both the perturbed and unperturbed simulations. Being $U(\omega_k)$ and $U_0(\omega_k)$ the Fourier transforms of the perturbation signal in the perturbed and unperturbed simulations, and $Y(\omega_k)$ and $Y_0(\omega_k)$ the Fourier transforms of the response signals in the perturbed and unperturbed simulations, respectively.

4. The frequency response of the system is obtained as $G(\omega_k) = (Y(\omega_k) - Y_0(\omega_k))/(U(\omega_k) - U_0(\omega_k))$.

AC Analysis (Linearized Circuit)

↪ *Added in PLECS 5.0.*

The AC Analysis linearizes the state-space and modified-nodal-analysis circuit equations at a given operating point. The response function is then determined using linear system theory. 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**
>
> For pure PLECS simulations, this analysis is equivalent to the AC Sweep with operating point defined as “non-periodic” (DC).

Usage in PLECS Standalone

In PLECS Standalone, all analyses are managed in the Analysis Tools Dialog shown in Fig. 7.1. To open the dialog, select **Simulation > Analysis tools...** from the menu of the schematic editor.

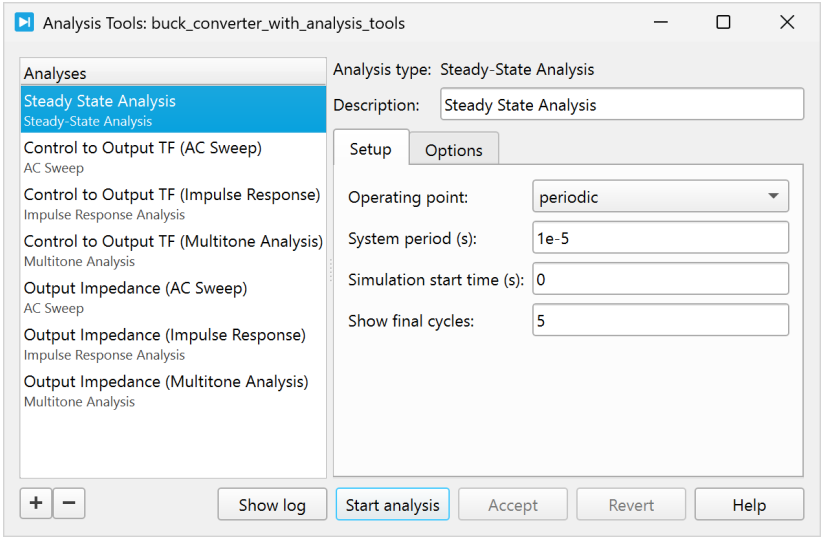


Fig. 7.1: Analysis tools dialog

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 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, in seconds (s). 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_{start} to be used in the transient simulation runs, in seconds (s). Simulations run from t_{start} to $t_{\mathrm{start}} + T$, 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`.

Max. number of threads
Specifies the maximum number of parallel threads that may be used during the analysis. When set to `auto`, the limit specified in the PLECS preferences is used.

↪ *Added in PLECS 4.8.*

AC Sweep

In order to perform an AC Sweep, you need to insert a Small Signal Perturbation block (p. 962) and a Small Signal Response block (p. 963) in order to define the points at which the perturbation is injected and the response is measured. The Small Signal Gain block (p. 961) can be used to obtain the loop gain of a closed 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, in seconds (s). 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, in hertz (Hz).

Amplitude
A vector containing the amplitudes of the perturbation signal at the lowest and highest frequency. The amplitudes at intermediate frequencies are

interpolated linearly. If a scalar is entered, the amplitude will be constant for all frequencies.

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

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

Simulation start time
The start time t_{start} to be used in the transient simulation runs, in seconds (s). Simulations run from t_{start} to $t_{\mathrm{start}} + T$, 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, in hertz (Hz).

Max. number of threads
Specifies the maximum number of parallel threads that may be used during the analysis. When set to `auto`, the limit specified in the PLECS preferences is used.

↪ *Added in PLECS 4.8.*

For a description of the steady-state options please refer to “Steady-State Analysis” (p. 238).

Impulse Response Analysis

In order to perform an impulse response analysis, you need to insert a Small Signal Perturbation block (p. 962) and a Small Signal Response block (p. 963) 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" (p. 239). In an impulse response analysis, the computational effort for an individual frequency is very cheap. Therefore, the parameter **Additional frequencies** is omitted; instead, the **Number of points** can be set to a large value in order obtain smooth curves.

Multitone Analysis

In order to perform a multitone analysis, you need to insert a Small Signal Perturbation block (p. 962) and a Small Signal Response block (p. 963) 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, in hertz (Hz). 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" (p. 233)). 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 sightly 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.

Simulation start time
The start time t_{start} to be used in the transient simulation runs, in seconds (s). The default is `0`.

Max. number of threads
Specifies the maximum number of parallel threads that may be used during the analysis. When set to `auto`, the limit specified in the PLECS preferences is used.

↪ *Added in PLECS 4.8.*

Frequency Response

In order to perform a frequency response analysis, you need to insert a Small Signal Perturbation block (p. 962) and a Small Signal Response block (p. 963) 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 on the **Show results** button.

> **Note**
>
> The button **Show log** opens the log of the analysis. It shows the progress of the analysis and other information which may help to improve the analysis result.

Initialization time span
The duration of an initial simulation performed before the sinusoidal perturbations are applied, in seconds (s). It is assumed that the system reaches its unperturbed steady state during this period.

The initial simulation runs only once. The reached system state is then used to initialize all the following perturbed simulations.

> **Note**
>
> If your model contains components which do not support storing and restoring their internal state (for example Subsystem (p. 977) in CodeGen Mode (see "Simulating a Subsystem in CodeGen Mode" (p. 354)), certain C-Script (p. 594), FMU (p. 686) and DLL (p. 664) components), set the **Initialization time span** to 0 and increase the **Settling time span**, so the perturbed system reaches steady state.

Settling time span

Time needed by the system to reach steady state after the sinusoidal perturbation is applied, in seconds (s).

Frequency ranges

A scalar or vector defining the perturbation frequency or frequencies, in hertz (Hz):

- For a scalar value, the analysis will be executed only for a single frequency. In this case, **Amplitude** and **Extraction cycles** also must be scalar values, **Number of points** must be 1, and **Additional frequencies** must be empty.
- A two-element vector with the lowest and highest perturbation frequencies.
- A vector of three or more strictly monotonically increasing values which divide the frequency range into multiple sections. Each section can have different **Amplitude** or **Extraction cycles** values.

> **Note**
>
> If the frequency response shows unexpected results at a certain frequency, create an additional *Frequency Response Analysis* to investigate the behaviour of the model at that specific frequency. Then apply the improved parameters to your original frequency response analysis.

Amplitude

A scalar or vector defining the amplitudes of the perturbation signal at frequencies specified in **Frequency ranges**:

- A scalar value will be used for all frequencies.
- A vector must have the same number of elements as the **Frequency ranges** vector. The amplitudes at intermediate frequencies are interpolated linearly.

Extraction cycles

A scalar or vector defining the number of cycles of each sinusoidal perturbation used to compute the frequency response after the settling time span has elapsed.

- A scalar value applies the same number of extraction cycles to all frequencies.
- A vector which defines different number of extraction cycles per frequency section. The length of the vector must be $n - 1$, where n is the number of elements in the **Frequency ranges** vector.

Perturbation

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

Response

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

Simulation start time

The start time t_{start} to be used in the transient simulation runs, in seconds (s). Simulations run from t_{start} to $t_{\text{start}} + T_{init} + T_s + T_c$, where T_{init} is the **Initialization time span**, T_s is the **Settling time span** and T_c is the extraction time span depending on the frequency and the **Extraction cycles**. The default is the **Start time** specified in the "PLECS Standalone Parameters" (p. 135).

> **Note**
>
> If the simulation starts from a **Stored system state** (see "System State" (p. 142)), the **Simulation start time** parameter is ignored. This is reported in the analysis log.

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, in hertz (Hz). These frequencies must be inside the **Frequency ranges**.

Max. number of threads
Specifies the maximum number of parallel threads that may be used during the analysis. When set to `auto`, the limit specified in the PLECS preferences is used.

↪ *Added in PLECS 4.8.*

AC Analysis (Linearized Circuit)

In order to perform an AC Analysis (Linearized Circuit), you need to insert a Small Signal Perturbation block (p. 962) and a Small Signal Response block (p. 963) in order to define the points at which the perturbation is injected and the response is measured. The Small Signal Gain block (p. 961) can be used to obtain the loop gain of a closed 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.

> **Note**
>
> The AC Analysis (Linearized Circuit) linearizes the circuit equations at the given operating point. It is hence not suited for determination of the response function of switched systems.

Operating point
This parameter is always set to “transient”. The analysis runs a transient simulation and then linearizes the circuit at the operating point.

Frequency range
A vector containing the lowest and highest perturbation frequency, in hertz (Hz).

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. This is used only for the determination of the standard PLECS response, not for PLECS Spice.

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 time t_{start} at which to perform the circuit linearization.

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, in hertz (Hz).

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 (p. 314) 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 $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ and $\mathbf{I}$ for the vector of switch positions specified by the previous command. The matrix $\mathbf{I}$ is the identity matrix if all electrical states are independent. Otherwise it specifies the relationship between the dependent variables.

These matrices can be further used for state_space averaging (p. 263).

Example Model

- See the example model **"State Space Matrices"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Above commands can also be invoked via the RPC interface (p. 323) using analogous syntax.

Application Example

The following demo model implements the buck converter shown in Fig. 7.2. It operates at a switching frequency of $100\,\mathrm{kHz}$ with a fixed duty-cycle of $15/28$. To run a transient simulation from zero initial conditions, select **Simulation > Start** from the menu.

Demo Model

- See the demo model **"Buck Converter with Analysis Tools"**.
- Find it in PLECS under **Window > Demo Models >** Power Supplies.

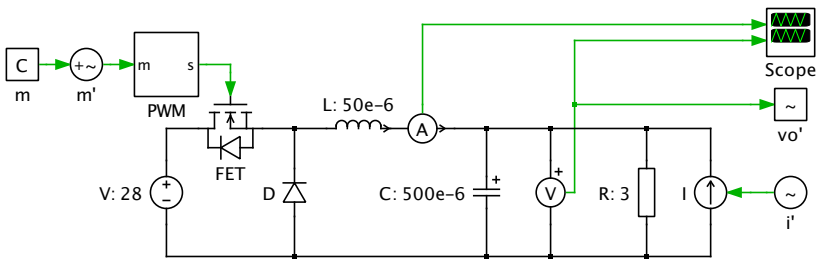


Fig. 7.2: Simulation model of the open-loop buck converter

To view the analyses configured in this model, select **Simulation > Analysis tools...** from the schematic editor 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 $T = 1/100\,\mathrm{kHz} = 10^{-5}\,\mathrm{s}$.

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 (p. 962) `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 (p. 1123) is therefore connected to the Small Signal Response block (p. 963) `vo'`.

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

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

Output impedance

For the calculation of the output impedance, a small perturbation current is injected into the converter output using a current source that is controlled by the Small Signal Perturbation block (p. 962) `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 following demo model implements the controlled buck converter shown in Fig. 7.3. A PID controller regulates the output voltage to 15 V.

Demo Model

- See the demo model **"Buck Converter with Loop Gain Analysis"**.
- Find it in PLECS under **Window > Demo Models >** Power Supplies.

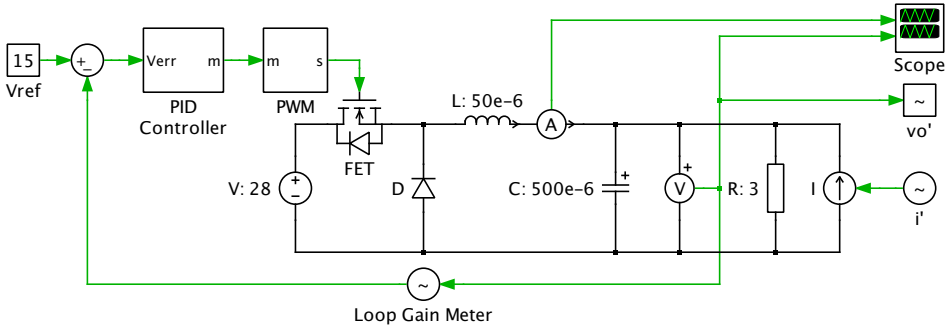


Fig. 7.3: Simulation model of the controlled buck converter

For the calculation of the voltage loop gain, the Small Signal Gain block (p. 961) `Loop Gain Meter` has been inserted into the feedback path. If you look under the block's mask, you can see how the block, both, injects a small perturbation and measures the system response.

To calculate the loop gain, select **Simulation > Analysis tools...** from the menu, then choose **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 (p. 1147) 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.

Table 7.1: Steady-State Analysis Parameters

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

continues on next page

Table 7.1 – continued from previous page

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

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

- Start an analysis using the parameters specified in the dialog box:

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

- Perform 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.

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

AC Sweep / Loop Gain Analysis

To perform an AC Sweep, copy the AC Sweep block (p. 1138) 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 (p. 1143) 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 block (p. 1138) or Loop Gain Analysis (AC Sweep) block (p. 1143). 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 block (p. 1138) and Loop Gain Analysis (AC Sweep) block (p. 1143). 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.

Table 7.2: AC Sweep Parameters

Parameter	Description
`TimeSpan`	Period length of the unperturbed system. Set by the **System period length** field.
`TStart`	Simulation start time. Set by the **Simulation start time** field.

continues on next page

Table 7.2 – continued from previous page

Parameter	Description
`FreqRange`	Range of the perturbation frequencies. Set by the **Frequency sweep range** field.
`FreqScale`	Specifies whether the sweep frequencies should be distributed on a `linear` or `logarithmic` scale. Set by the **Frequency sweep scale** field.
`NPoints`	Number of data points generated. Set by the **Number of points** field.
`InitialAmplitude`	Perturbation amplitude at the first perturbation frequency. Set by the **Amplitude at first freq** field.
`Method`	Method used for obtaining the periodic steady-state operating point of the perturbed system: `Brute force simulation - start from model initial state`, `Brute force simulation - start from unperturbed steady state`, `Steady-state analysis - start from model initial state`, `Steady-state analysis - start from unperturbed steady state`. Set by the **Method** drop-down list.
`Tolerance`	Relative error tolerance used in the convergence criterion. Set by the **Termination tolerance** field.
`MaxIter`	Maximum number of iterations allowed. Set by the **Max number of iterations** field.
`Display`	Specifies the level of detail of the diagnostic messages displayed in the command window (`iteration`, `final`, `off`). Set by the **Display** drop-down list.
`HideScopes`	Hide all Simulink scope windows during an analysis in order to save time.
`HiddenStates`	Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (`error`, `warning`, `none`). Set by the **Hidden model states** drop-down list.

continues on next page

Table 7.2 – continued from previous page

Parameter	Description
`OutputName`	Name of a MATLAB variable used to store the transfer function at the end of an analysis. Set by the **Output variable** field.
`BodePlot`	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the **Plot Bode diagram** drop-down list.
`JPert`	Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.
`NInitCycles`	If a steady-state analysis is used to obtain the starting point of the ac analysis (see parameter `Method` above), this parameter specifies the number of cycle-by-cycle simulations that should be performed before the steady-state analysis. This parameter can be used to provide the algorithm with a better starting point. The default is `0`.

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

- Start an analysis using the parameters specified in the dialog box.

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

- Perform two analyses with different start times and assigns the resulting transfer functions to two different variables `T0` and `T1`. 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.

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

Impulse Response Analysis

To perform an impulse response analysis, copy the Impulse Response Analysis block (p. 1141) 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 (p. 1141). 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 (p. 1141). 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.

Table 7.3: Impulse Response Analysis Parameters

Parameter	Description
`TimeSpan`	Period length of the unperturbed system. Set by the **System period length** field.
`TStart`	Simulation start time. Set by the **Simulation start time** field.
`FreqRange`	Range of the perturbation frequencies. Set by the **Frequency sweep range** field.
`FreqScale`	Specifies whether the sweep frequencies should be distributed on a `linear` or `logarithmic` scale. Set by the **Frequency sweep scale** field.
`NPoints`	Number of data points generated. Set by the **Number of points** field.
`Perturbation`	Perturbation amplitude of the discrete impulse. Set by the **Perturbation** field.

continues on next page

Table 7.3 – continued from previous page

Parameter	Description
`Compensation`	Specifies whether and how the effect of the sampling should be compensated (`none`, `discrete pulse`, `external reference`). Set by the **Compensation for discrete pulse** drop-down list.
`Tolerance`	Relative error tolerance used in the convergence criterion of the initial steady-state analysis. Set by the **Termination tolerance** field.
`MaxIter`	Maximum number of iterations allowed during the initial steady-state analysis. Set by the **Max number of iterations** field.
`Display`	Specifies the level of detail of the diagnostic messages displayed in the command window (`iteration`, `final`, `off`). Set by the **Display** drop-down list.
`HideScopes`	Hide all Simulink scope windows during an analysis in order to save time.
`HiddenStates`	Specifies how to handle Simulink blocks with 'hidden' states, i.e. states that are not stored in the state vector (`error`, `warning`, `none`). Set by the **Hidden model states** drop-down list.
`OutputName`	Name of a MATLAB variable used to store the transfer function at the end of an analysis. Set by the **Output variable** field.
`BodePlot`	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the **Plot Bode diagram** drop-down list.
`JPert`	Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.
`NInitCycles`	Number of cycle-by-cycle simulations that should be performed before the initial steady-state analysis. This parameter can be used to provide the algorithm with a better starting point. The default is `0`.

Multitone / Loop Gain Analysis

To perform a multitone analysis, copy the Multitone Analysis block (p. 1145) 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 (p. 1144) 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 block (p. 1145) and Loop Gain Analysis (Multitone) block (p. 1144). The **Parameter** column shows the parameter names to be used with the `plmultitone` 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.

Table 7.4: Multitone Analysis Parameters

Parameter	Description
`FreqRange`	Range of the perturbation frequencies. Set by the **Frequency sweep range** field.
`Amplitude`	Amplitude of the perturbation signal. Set by the **Amplitude** field.
`TStart`	Simulation start time. Set by the **Simulation start time** field.
`Display`	Specifies the level of detail of the diagnostic messages displayed in the command window (`iteration`, `final`, `off`). Set by the **Display** drop-down list.

continues on next page

Table 7.4 – continued from previous page

Parameter	Description
`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 **Output variable** field.
`BodePlot`	Plot a Bode diagram of the transfer function at the end of an analysis. Set by the **Plot Bode diagram** drop-down list.

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.

1 Query the state-space order:

```
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.

2 Set the switch positions for the subsequent analysis to `switchpos`:

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

3 Retrieve state-space matrices:

```
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. These matrices can be used for state_space averaging (p. 263).

The matrices obtained can be further used for state_space averaging (p. 263).

Example Model

- See the example model **"State Space Matrices"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

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 Fig. 7.4. The converter should supply a regulated 15 V to a resistive load at a nominal load current of 5 A.

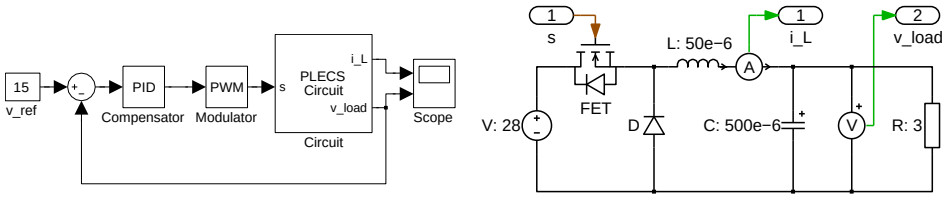


Fig. 7.4: Simulation model of the regulated buck converter

The examples used in this section follow the design example in *Erickson*[3], Chapter 9. They have been implemented in the following demo models:

Demo Model

- See the demo model **"Buck Converter with Parameter Sweep"**.
- Find it in PLECS under **Window** > **Demo Models** > Power Supplies.

Demo Model

- See the demo model **"Buck Converter with Analysis Tools"**.
- Find it in PLECS under **Window** > **Demo Models** > Power Supplies.

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

Demo Model

- See the demo model **"Buck Converter with Loop Gain Analysis"**.
- Find it in PLECS under **Window > Demo Models >** Power Supplies.

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 $V_{\mathrm{out}}/V_{\mathrm{src}} = 15\,\mathrm{V}/28\,\mathrm{V}$. You can verify this by using the Steady-State Analysis block (p. 1147) 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 Fig. 7.5.

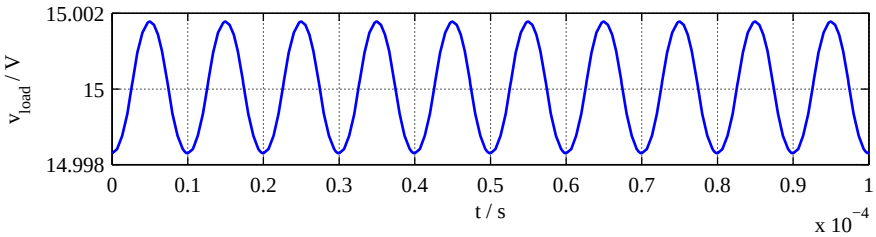


Fig. 7.5: Steady-state output voltage

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 (p. 1138) as shown in Fig. 7.6. The block's output is the perturbation signal; it is added to the steady-state duty cycle. The block's input is connected to the load's voltage signal.

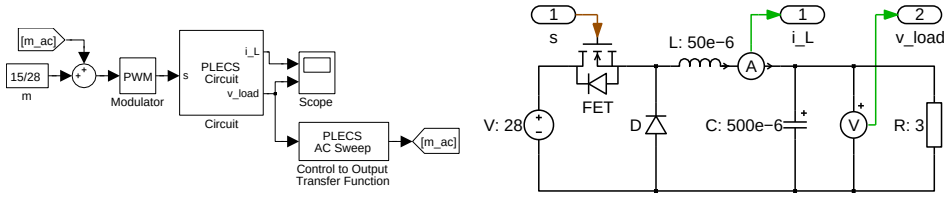


Fig. 7.6: Simulation model to determine the open-loop control-to-output transfer function

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 $100\,\mathrm{Hz}$ and $50\,\mathrm{kHz}$ with a few extra points between 800 Hz and $1200\,\mathrm{Hz}$. This is achieved by setting the parameter to `[100 800:50:1200 50000]`. As expected, the resulting bode plot (Fig. 7.7) of the transfer function shows a double pole at $f_0 = 1/(2\pi\sqrt{LC}) \approx 1\,\mathrm{kHz}$ and a dc gain of $G_0 = 28\,\mathrm{V} \approx 29\,\mathrm{dB}$.

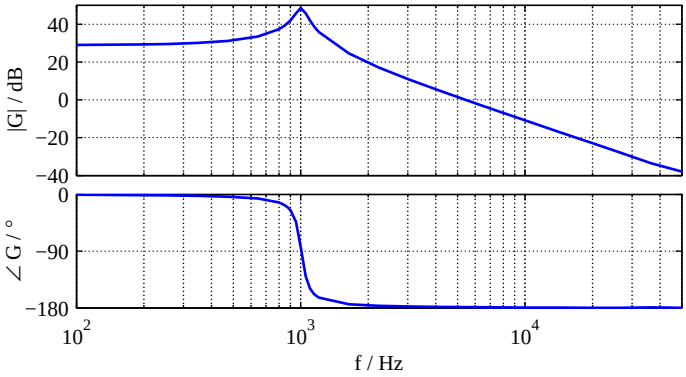


Fig. 7.7: Bode plot of the open-loop control-to-output transfer function

Open-loop output impedance

Although not required for the compensator design, we will now calculate the output impedance for demonstration purposes. To do so we need to inject a small ac current into the converter output and measure the resulting perturbation of the output voltage. We therefore connect a Current Source (Controlled) (p. 632) in parallel with the load Resistor (p. 905) `R` as shown in Fig. 7.8. 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 A; we therefore set the initial perturbation amplitude to `1e-2`.

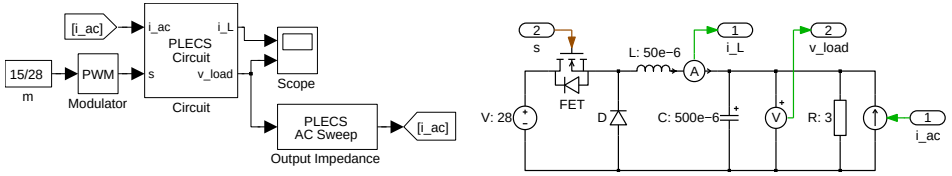


Fig. 7.8: Simulation model to determine the open-loop output impedance

The resulting output impedance is shown in Fig. 7.9.

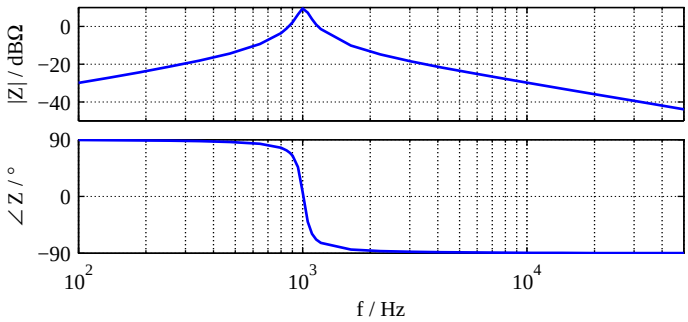


Fig. 7.9: Bode plot of the open-loop output impedance

Impulse Response Analysis

Alternatively you can determine the open-loop transfer functions using the Impulse Response Analysis block (p. 1141) as shown in Fig. 7.10. In this analysis method, the calculation of an individual output point is relatively inexpensive. We therefore set the number of points to `300` and extend the sweep range to `[10 50000]`. In order to compensate for the discrete rectangular pulse used to perturb the system, we choose the setting `external reference` for the control-to-output transfer function and `discrete pulse` for the output impedance. This option is specific to the PLECS Blockset. It separately handles the calculation of control-to-output transfer functions that depend on the duty cycle of a switched converter, where the rectangular input signal can interfere with the modulator's sampling.

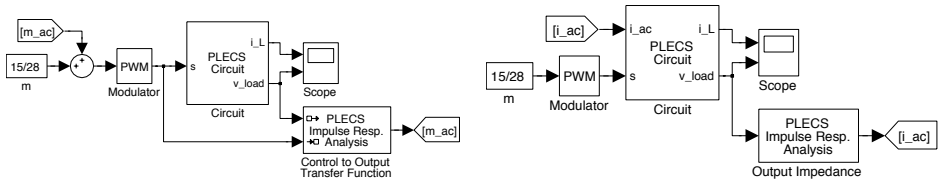


Fig. 7.10: Using the Impulse Response Analysis block (p. 1141)

Loop Gain Analysis

Compensator settings

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

The compensator is implemented as shown in Fig. 7.11 (left). The compensator output is limited to the interval $[0.1, 0.9]$. In order to prevent windup problems during the steady-state analysis, the integrator is limited to the same range. Its transfer function is shown in Fig. 7.11 (right).

Loop gain

The loop gain of the closed control loop is measured by inserting the Loop Gain Analysis (AC Sweep) block (p. 1143) into the loop path. A good place is the feedback path as shown in Fig. 7.12. The average steady-state load voltage is $15\,\mathrm{V}$; the initial perturbation amplitude is therefore chosen as 10^{-2}. The convergence of the initial steady-state analysis can be accelerated by pre-charging the capacitor to its average steady-state voltage.

As can be seen from the resulting bode plot (Fig. 7.13 on p. 265), the design goals for crossover frequency and phase margin have been reached.

State-Space Averaging

↪ *Added in PLECS 4.8.*

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

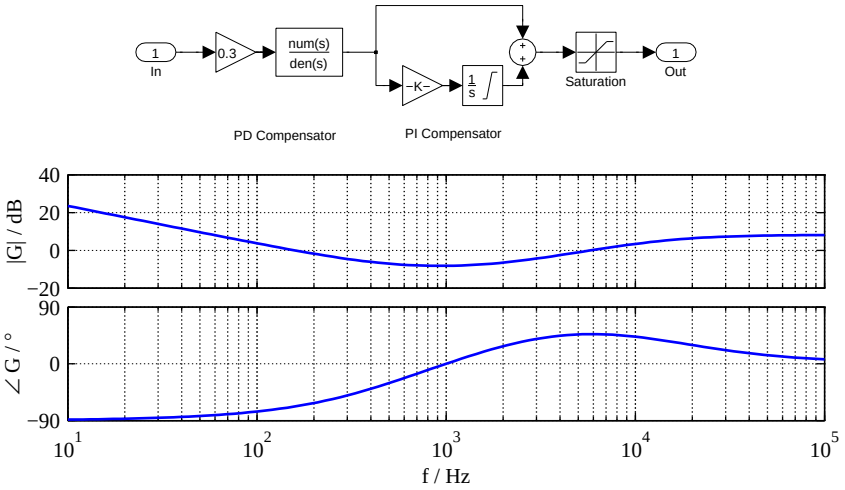


Fig. 7.11: PID compensator design (top) and its transfer function (bottom)

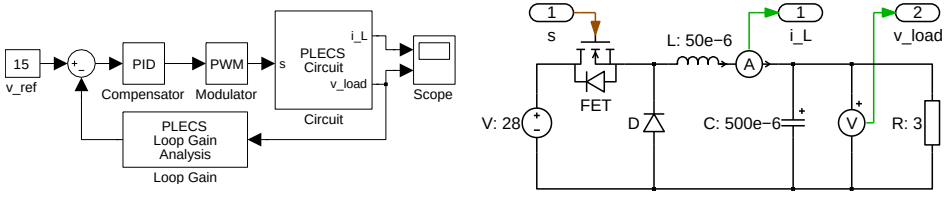


Fig. 7.12: Simulation model to determine the loop gain

easily fill a book of its own. This manual therefore assumes that you are familiar with the concept and just highlights how to use PLECS in the process.

Example Model

- See the example model **"State Space Matrices"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

The small-signal ac model of a dc converter operating in continuous conduction

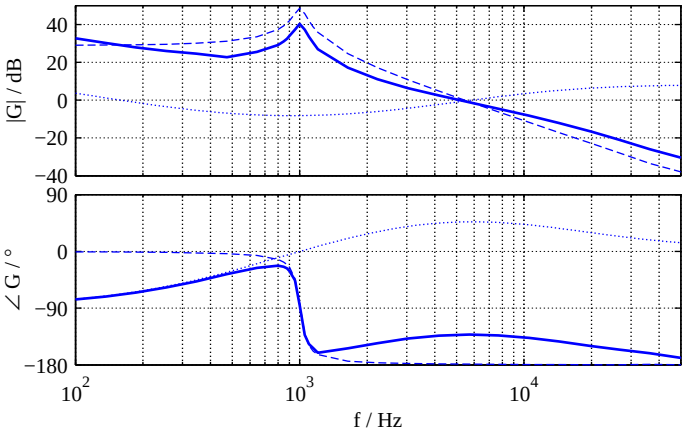


Fig. 7.13: Bode plot of the loop gain (solid) shown together with the open-loop control-to-output function (dashed) and the PID compensator transfer function (dotted).

mode (CCM) is described by the equation system

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

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

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

where the subscript $_1$ denotes the interval when the switch is conducting and the diode blocking, and the subscript $_2$ denotes the interval when the switch is blocking and the diode conducting.

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

```
load_system('plBuckSweep');
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 `Vm`). Such a transfer function is given by:

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

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
```

(continues on next page)

(continued from previous page)

```
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 as shown in Fig. 7.14.

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

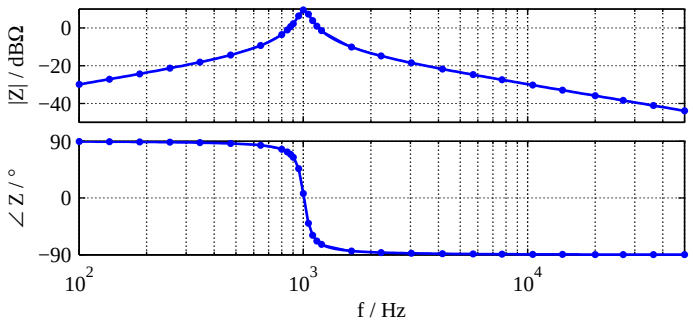


Fig. 7.14: Open-loop output impedance (solid line) calculated with state-space averaging compared to the data points returned by the AC Sweep (dots).

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

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

with

$$\mathbf{A}_{\mathrm{co}} = \bar{\mathbf{A}}$$
$$\mathbf{B}_{\mathrm{co}} = \big\{ -(\mathbf{A}_1 - \mathbf{A}_2)\bar{\mathbf{A}}^{-1}\bar{\mathbf{B}} + (\mathbf{B}_1 - \mathbf{B}_2)\big\}\bar{\mathbf{u}}$$
$$\mathbf{C}_{\mathrm{co}} = \bar{\mathbf{C}}$$
$$\mathbf{D}_{\mathrm{co}} = \big\{ -(\mathbf{C}_1 - \mathbf{C}_2)\bar{\mathbf{A}}^{-1}\bar{\mathbf{B}} + (\mathbf{D}_1 - \mathbf{D}_2)\big\}\bar{\mathbf{u}}$$

Note that $\mathbf{B}_{\mathrm{co}}$ and $\mathbf{D}_{\mathrm{co}}$ 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, resulting in the bode plot shown in Fig. 7.15.

```
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})
```

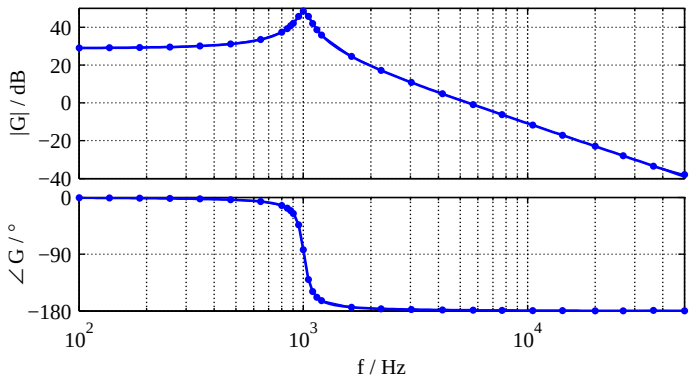


Fig. 7.15: Bode plot of the open-loop control-to-output transfer function calculated with state-space averaging (solid) and the data points returned by the AC Sweep (dots).

Reference

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

8

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” (p. 279).

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” (p. 31).

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:

$$\begin{aligned} y &= f_{\text{output}}(t, u, x_\text{c}, x_\text{d}) \\ x_\text{d}^\text{next} &= f_{\text{update}}(t, u, x_\text{c}, x_\text{d}) \\ \dot{x}_\text{c} &= f_{\text{derivative}}(t, u, x_\text{c}, x_\text{d}) \end{aligned}$$

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” (p. 37)).

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” (p. 36)).

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”.

> **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”.

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.).

Store Custom State Function

This function is called at the end of a simulation before the terminate function. Use this function to store any custom values which are needed to restore the state of the C-Script. Continuous and discrete states are automatically stored. Use the macros `WriteCustomStateDouble`, `WriteCustomStateInt` and `WriteCustomStateData` to serialize your data.

Restore Custom State Function

This function is called after the start function and before the first call of the output function. The continuous and discrete states have already been restored at this point. Use this function to initialize your block with a previously stored custom state. Use the macros `ReadCustomStateDouble`, `ReadCustomStateInt` and `ReadCustomStateData` to deserialize your data and `SetErrorMessage` to report any issues to the user during restoring.

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.

The following two subsections explain how to achieve this.

Example Model

- See the example model **"C-Script: Zero-Crossing Signals"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

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" (p. 38).

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" (p. 275).

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” (p. 38). 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 $[T_\mathrm{p}, T_\mathrm{o}]$) 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 $0 \leq T_\mathrm{o} < T_\mathrm{p}$; it may be omitted if it is zero.

The time steps, at which the output and update functions are executed, are calculated as $n \cdot T_\mathrm{p} + T_\mathrm{o}$ 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. Therfore, the actual simulation time at a discrete-variable sample hit may be slightly larger than the value that was specified as the next sample hit.
> The solvers of PLECS Standalone, however, can evaluate the sample hit requests directly and are therefore guaranteed to meet the requests exactly.

Multiple Sample Times

If you want to model a hybrid system, you can specify multiple sample times in different rows of an $n \times 2$ matrix. For example, if your C-Script has continuous states but you must also ensure that it is executed every 0.5 seconds with an offset of 0.1 seconds, you would enter `[0, 0; 0.5, 0.1]`.

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

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

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” (p. 43)).

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

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));
```

Example Model

- See the example model **"Using Model Parameters"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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 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:

$$y = f_{\text{output}}(t, u, x_{\text{c}}, x_{\text{d}}) = 2 \cdot u$$

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.

> **Example Model**
>
> - See the example model **"Times Two Function"**.
> - Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

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:

$$y = f_{\text{output}}(t, u, x_\text{c}, x_\text{d}) = x_d$$
$$x_\text{d}^{\text{next}} = f_{\text{update}}(t, u, x_\text{c}, x_\text{d}) = u$$

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);
```

Example Model

- See the example model **"Sampled Delay"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

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:

$$\begin{aligned} y &= f_{\text{output}}(t, u, x_\text{c}, x_\text{d}) &&= x_\text{c} \\ \dot{x}_\text{c} &= f_{\text{derivative}}(t, u, x_\text{c}, x_\text{d}) &&= u \end{aligned}$$

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);
```

Example Model

- See the example model **"Continuous Integrator"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

Event Handling – Wrapping Integrator

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

Block Setup

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

Output Function Code

```
#define PI 3.141592653589793
if (IsMajorStep)
{
  if (ContState(0) > 2*PI)
    ContState(0) -= 2*PI;
  else if (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, 2π is added to or subtracted from the state so that it lies within the boundaries again. In every major and minor time step, the zero-crossing signals are calculated so that they become zero when the state is 0 resp. 2π. Finally, the integrator state is assigned to the output.

Note, that the state *must 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);
```

Example Model

- See the example model **"Example Wrapping Integrator"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Piecewise Smooth Functions – Saturation

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

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

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” (p. 272)).

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;
}
```

(continues on next page)

(continued from previous page)

```
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.

Example Model

- See the example model **"Example Saturation"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

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.

Store Custom State Code

```
WriteCustomStateDouble(NextSampleHit);
```

The previous input value is stored automatically because it is a discrete state. The `NextSampleHit` has to be stored in the custom state.

Restore Custom State Code

```
NextSampleHit = ReadCustomStateDouble();
```

Restore the `NextSampleHit` value. When the simulation starts from a stored system state that was stored before this block was added to the schematic, the read operation will fail and PLECS reports a runtime error.

Example Model

- See the example model **"Turn-on delay using a C-Script"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

Load External Files

↪ *Added in PLECS 4.8.*

The C-Script can be used to load external functions. In this example, the two functions `sum` and `product` are defined in external files and executed via the C-Script.

Header File

```
float sum(float a, float b);
float product(float a, float b);
```

Content of the file `cscript_example_external_files.h`.

C Source Code File

```
#include "cscript_example_external_files.h"

float sum(float a, float b)
{
    return a + b;
}

float product(float a, float b)
{
    return a * b;
}
```

Content of the file `cscript_example_external_files.c`.

Code Declarations

```
#include "cscript_example_external_files.h"
#include "cscript_example_external_files.c"

#define In1 InputSignal(0,0)
#define In2 InputSignal(1,0)

#define Out1 OutputSignal(0,0)
#define Out2 OutputSignal(1,0)
```

The external header and C source code files are loaded. Furthermore, two input and two output variables are definded.

Output Function Code

```
Out1 = sum(In1,In2);
Out2 = product(In1,In2);
```

In the update function, the two output variables are updated with the sum and product of the two input variables.

Example Model

- See the example model **"Including External .c and .h Files"**.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

C-Script Macros

The following table summarizes the macros that can be used in the C-Script function code sections.

Table 8.1: C-Script Data Access Macros

Macro	Type	Access	Description
`NumInputTerminals`	int	R	Returns the number of input terminals.
`NumOutputTerminals`	int	R	Returns the number of output terminals.
`NumInputSignals (int i)`	int	R	Returns the number of elements (i.e. the width) of the signal connected to the `i`th input terminal.
`NumOutputSignals (int i)`	int	R	Returns the number of elements (i.e. the width) of the signal connected to the `i`th output terminal.
`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).
`NumSampleTimes`	int	R	Returns the number of sample times.

continues on next page

Table 8.1 – continued from previous page

Macro	Type	Access	Description
`SampleTimePeriod (int i)`	int	R	Returns the period of the `i`th sample time.
`SampleTimeOffset (int i)`	int	R	Returns the offset of the `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`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, int j)`	double	R	Returns the value of the `j`th element of the `i`th input signal terminal. See the C-Script block (p. 594) for information on how to increase the default number of input signal terminals.

continues on next page

Table 8.1 – continued from previous page

Macro	Type	Access	Description
`OutputSignal (int i, int j)`	double	R/W	Provides access to the value of the `j`th element of the `i`th output signal terminal. See the C-Script block (p. 594) for information on how to increase the default number of output signal terminals. Output signals may *only* be changed during the output function call.
`ContState (int i)`	double	R/W	Provides access to the value of the `i`th continuous state. Continuous state variables may *not* be changed during minor time steps.
`ContDeriv (int i)`	double	R/W	Provides access to the derivative of the `i`th continuous state.
`DiscState (int i)`	double	R/W	Provides access to the value of the `i`th discrete state. Discrete state variables may *not* be changed during minor time steps.
`ZCSignal (int i)`	double	R/W	Provides access to the `i`th zero-crossing signal.
`ParamNumDims (int i)`	int	R	Returns the number of dimensions of the `i`th user parameter.
`ParamDim (int i, int j)`	int	R	Returns the `j`th dimension of the `i`th user parameter.

continues on next page

Table 8.1 – continued from previous page

Macro	Type	Access	Description
`ParamRealData (int i, int j)`	double	R	Returns the value of the `j`th element of the `i`th user parameter. The index `j` is a linear index into the parameter elements. Indices into multi-dimensional arrays must be calculated using the information provided by the `ParamNumDims` and `ParamDim` macros. 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`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.

`WriteCustomStateDouble (double val)`
`WriteCustomStateInt (int val)`
`WriteCustomStateData (void *data, int len)`
	void
	void
	void
		W	Write a custom state of type double, int or custom data with `len` number of bytes. Use multiple calls for multiple values.

continues on next page

Table 8.1 – continued from previous page

Macro	Type	Access	Description

`ReadCustomStateDouble()`
`ReadCustomStateInt()`
`ReadCustomStateData (void *data, int len)`
	double
	int
	void
		R	Read a custom state of type double, int or custom data with `len` number of bytes. Use multiple calls for multiple values.

`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 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
```

Note

Prefer reading and writing of custom state variables with double and int values over void* custom data. The latter cannot handle the byte order (big endian, little endian) of your platform. To store a vector of doubles use the following code:

```
// platform independent code, recommended
WriteCustomStateInt(vectorSize);
for (int i = 0; i < vectorSize; ++i)
{
    WriteCustomStateDouble(vector[i]);
}

// platform dependent code, not recommended
WriteCustomStateData(&vectorSize, sizeof(vectorSize));
WriteCustomStateData(vector, vectorSize*sizeof(double));
```

Deprecated Macros

↪ *Deprecated since PLECS 3.7:* The macros `NumInputs`,`NumOutputs`,`Input(int i)` and `Output(int i)` are deprecated but are still supported for C-Scripts that have only a single input and output terminal.

9

State Machines

↪ *Added in PLECS 4.1.*

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.

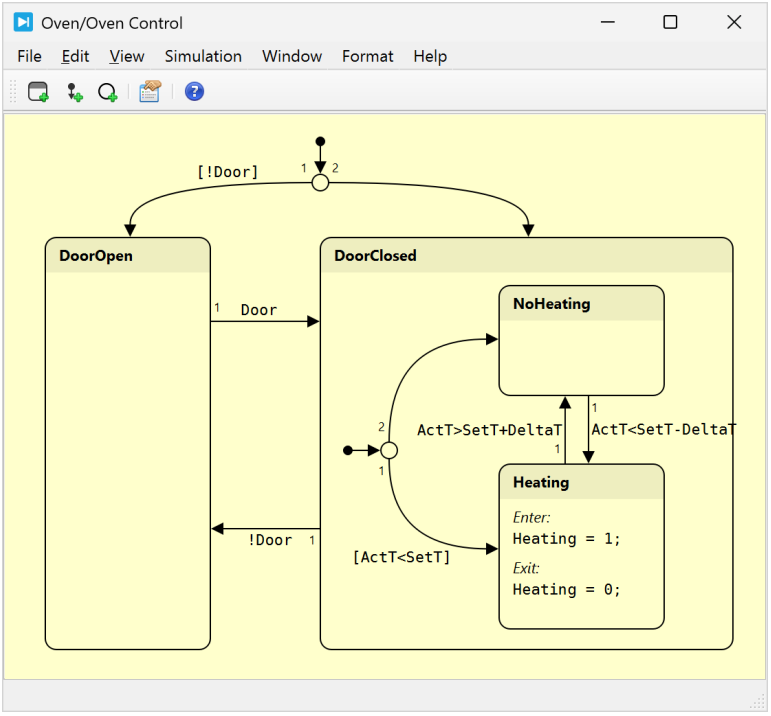


Fig. 9.1: State machine editor window

> **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.

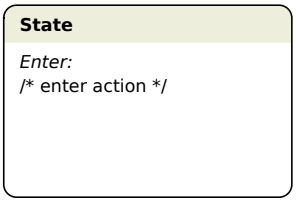


Fig. 9.2: An empty state

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** (on macOS: **Cmd**) key, 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 contain 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.

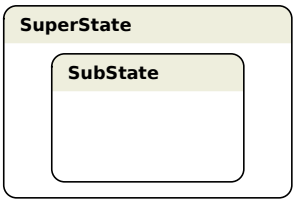


Fig. 9.3: A super-state containing a sub-state

> **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 label and drag it to the desired location. By double-clicking on the label you can edit it directly on the chart.

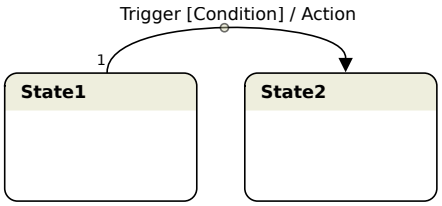


Fig. 9.4: A transition with labels and control handle

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.

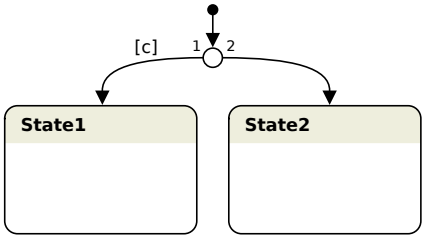


Fig. 9.5: A junction forking a default transition

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.

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 sig-

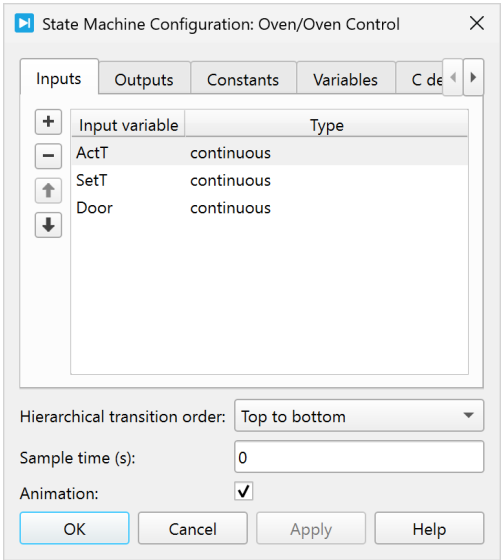


Fig. 9.6: State machine configuration editor

nal 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 predefined macros can be used in actions:

Table 9.1: Predefined Action Macros

Macro	Type	Access	Description
`CurrentTime`	double	R	Returns the current simulation time.

continues on next page

Table 9.1 – continued from previous page

Macro	Type	Access	Description
`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” (p. 297) and “Execution of Hierarchical State Machines” (p. 307)). 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 Times” (p. 43).

Table 9.2: Sample Time Parameter Values

Type	Value	
Continuous	`[0, 0]` or `0`	
Discrete-Periodic	`[`T_p`, `T_o`]` or T_p	T_p: Sample period, $0 < T_p$ 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” (p. 303)). 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”).

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 in Fig. 9.7, 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.

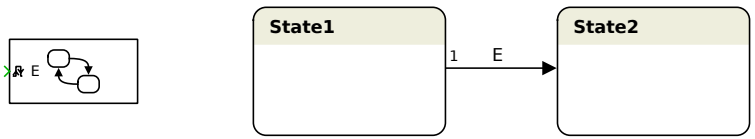


Fig. 9.7: State machine with explicit trigger

> **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 Fig. 9.8.

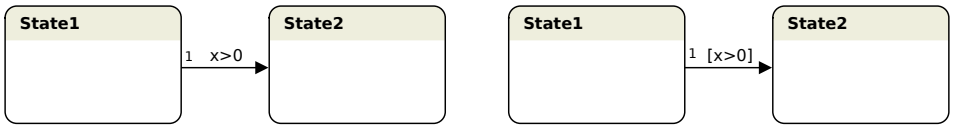


Fig. 9.8: Implicit trigger (left) versus condition (right)

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 Fig. 9.9. Consider that **S11** is the currently active state, so both **S1** and **S** are also implicitly active.

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

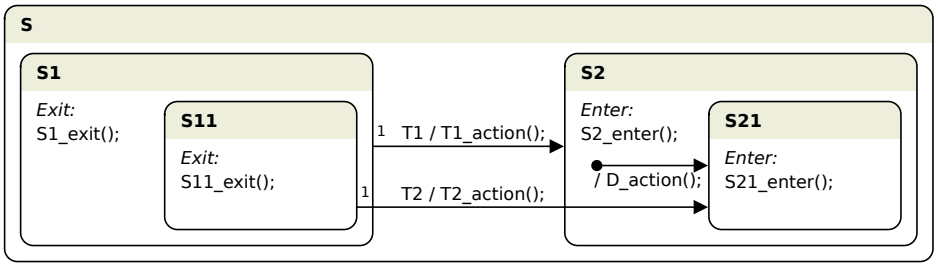


Fig. 9.9: Execution sequence in a hierarchical state machine

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.

Example Model

- See the example model **"Oven Control"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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 state machine uses a constant variable `DeltaT` 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, the (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 transition, 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.

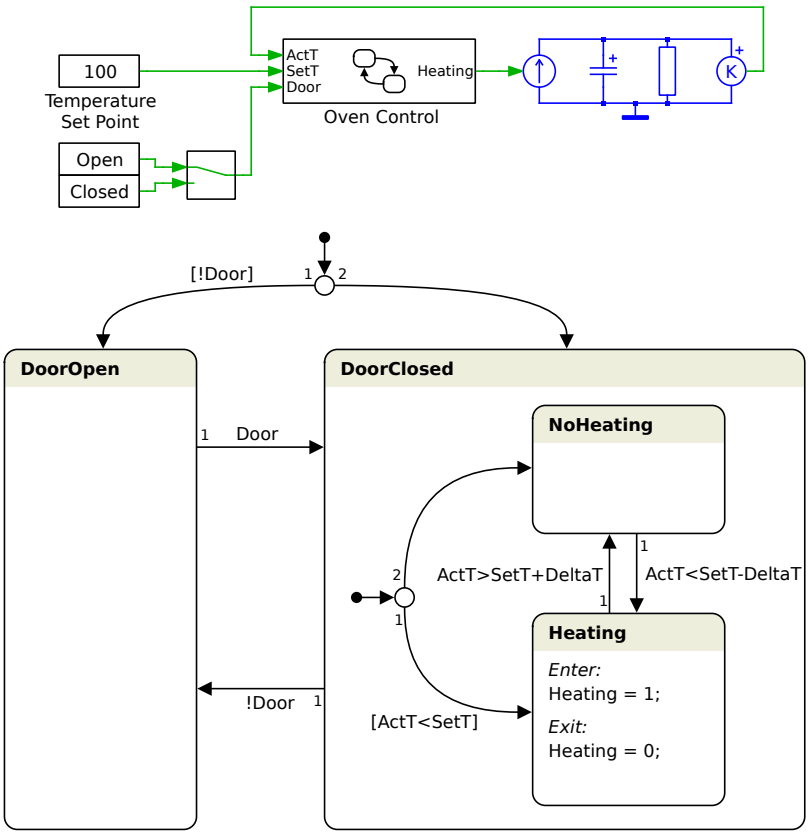


Fig. 9.10: Example of a simple oven control

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+DeltaT` 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.

Demo Models

Demo Model

- The demo model **"Current Controlled Z-Source Inverter"** features a modulator implemented with the state machine block.
- Find it in PLECS under **Window** > **Demo Models** > Power Supplies.

Demo Model

- The demo model **"Buck Converter with Cascaded Controls"** features a state machine based current controller.
- Find it in PLECS under **Window** > **Demo Models** > Power Supplies.

Demo Model

- The demo model **"Windpower System with Permanent Magnet Synchronous Generator"** features a maximum power point tracker implemented with the state machine block.
- Find it in PLECS under **Window** > **Demo Models** > Power Generation.

10

Simulation Scripts

Running simulations from a script allows you to perform automated tasks such as examining the effect of varying parameters or post-processing the simulation results to extract relevant information.

PLECS Standalone offers two different scripting methods:

- Simulation scripts written in Octave can be executed directly in PLECS Standalone. This is described in section “Simulation Scripts in PLECS Standalone”.
- PLECS offers an RPC interface that allows any other program that can send XML-RPC or JSON-RPC requests to control PLECS. Many scripting languages support XML-RPC or JSON-RPC out of the box, for example Python or Ruby. Other scripting language extensions for XML-RPC or JSON-RPC support are available for free on the internet. This is described in section “RPC Interface in PLECS Standalone” (p. 323).

In PLECS Blockset, scripts are written and executed 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. They are described in section “Command Line Interface in PLECS Blockset” (p. 337).

Simulation Scripts in PLECS Standalone

Simulation scripts are managed in the Simulation Scripts dialog shown in Fig. 10.1. To open the dialog, select **Simulation scripts...** from the **Simulation** menu of the schematic editor.

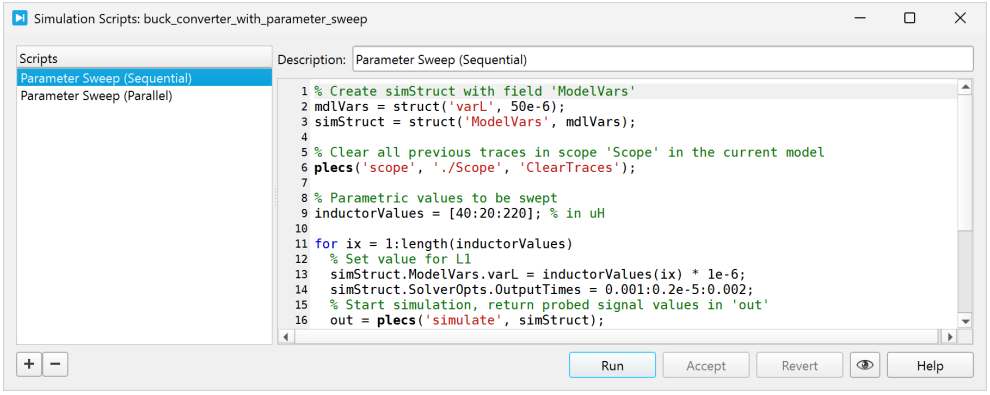


Fig. 10.1: Simulation Scripts dialog in PLECS Standalone

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[4].

↪ *Changed in PLECS 4.4:* Updated to Octave 4.4.

[4] https://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" (p. 319).

Handling of Information, Warnings and Errors in Initialization Commands

↪ *Added in PLECS 4.1.*

During the execution of mask initialization commands,

```
plecs('info', infoMessage)
plecs('warning', warningMessage)
plecs('error', errorMessage)
```

add entries to the Diagnostics window. Error messages issued with the Octave command `error` are caught by PLECS and are also shown in the Diagnostics window.

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" (p. 319).

Retrieving Scope Cursor Data

↪ *Added in PLECS 4.2.*

```
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 the "Cursors" section (p. 122).

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

↪ *Added in PLECS 4.2.*

```
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

↪ *Added in PLECS 4.2.*

```
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:

Table 10.1: Scope Settings for Bitmap Export

Name	Description
`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.

continues on next page

Table 10.1 – continued from previous page

Name	Description
`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`, `bottom-middle` 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.

Running a Simulation

```
plecs('simulate')
plecs('simulate', optStruct)
```

runs a simulation. The optional argument `optStruct` can be used to override model parameters; for detailed information see section “Scripted Simulation and Analysis Options” (p. 329).

If any outports exist on the top level of the simulated model, the command returns a struct consisting of two fields, `Time` and `Values`. `Time` is a vector of simulation time stamps. `Values` is an $m \times n$ array containing the output values, where m is the number of time stamps and n is the number of output signals. The order of the signals is determined by the port numbers.

By default, the simulation result contains all simulation steps that the solver makes. This can be controlled with the solver options `OutputTimes` and `OutputTimesOption`, where `OutputTimes` is a double vector and `OutputTimesOption` is a string as described in Tab. 10.4 on p. 334.

Running an Analysis

```
plecs('analyze', analysisName)
plecs('analyze', analysisName, optStruct)
```

runs 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" (p. 329).

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. By default, the simulation result contains all simulation steps that the solver makes. This can be controlled with the solver options `OutputTimes` and `OutputTimesOption`, where `OutputTimes` is a double vector and `OutputTimesOption` is a string as described in Tab. 10.4 on p. 334.

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.

Running Multiple Simulations or Analyses in Parallel

↪ *Added in PLECS 4.6.*

Instead of a single `optStruct` argument you can also pass a $1 \times N$ cell array of option structures to the `simulate` and `analyze` commands:

```
plecs('simulate', optStructs)
plecs('analyze', analysisName, optStructs)
```

PLECS will then automatically distribute the individual simulations or analyses to the CPU cores available on your computer. After completion, the commands return a $1 \times N$ cell array containing the individual result structs or – in case of a runtime error – a string with the error message. To avoid memory problems when running a large number of parallel simulations, you can define an appropriate `OutputTimes` vector in the `SolverOpts` struct in order to reduce the number of data points.

Additionally, you can define a callback function that is called whenever an individual simulation or analysis has completed. The arguments of this callback function are the index of the corresponding option set in the input cell array and the result of the simulation or analysis. The return value of the callback function is stored in the output cell array instead of the simulation or analysis result.

The following code example shows how such a callback function can be used to replace the simulation result with a single aggregated value (the maximum value of a certain output signal).

```
% Aggregate the simulation results in a callback function.
function result = callback(index, result)
   if isstruct(result)
      % If the simulation succeeded,
      % replace the simulation result with
      % the maximum value of the first output signal.
      result = max(result.Values(1,:));
   end
end

out = plecs('simulate', optStructs, ...
            @(index, result) callback(index, result));
```

The notation using an anonymous function handle is due to a technical limitation of Octave 4.4 regarding nested functions.

Example Script

The following script runs a parameter sweep by setting the variable `varL` to the values in `inductorValues`. It is used in the demo model BuckParamSweep.

```
mdlVars = struct('varL', 50e-6);
opts = struct('ModelVars', mdlVars);
```

(continues on next page)

(continued from previous page)

```
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” (p. 319)). 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.

RPC Interface in PLECS Standalone

The RPC interface allows you to control PLECS Standalone from an external program. PLECS acts as an HTTP server which processes XML-RPC[5] or JSON-RPC[6] requests from clients. PLECS automatically determines the actual protocol used by a client from the request and sends the response using the same protocol.

↪ *Changed in PLECS 4.6:* Added support for JSON-RPC.

The 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.

> **Note**
>
> RPC connections to PLECS are only allowed from clients running on the same computer as PLECS. Therefore, the connection should always be initiated using `localhost` in the server URL.

Usage Examples

The code examples in this section assume that PLECS is configured to use TCP port 1080 for RPC.

Using XML-RPC in Python

The following Python 3 code initializes an XML-RPC client for PLECS:

```
import xmlrpc.client
proxy = xmlrpc.client.ServerProxy("http://localhost:1080")
```

Using JSON-RPC in MATLAB

To facilitate interaction between PLECS Standalone and MATLAB, Plexim provides a MATLAB class that can be downloaded from our GitHub matlab-jsonrpc repository[7]. After copying the class file `jsonrpc.m` to your MATLAB path, you can initialize a JSON-RPC client for PLECS as follows:

[5] http://xmlrpc.com
[6] https://www.jsonrpc.org
[7] https://github.com/plexim/matlab-jsonrpc

```
proxy = jsonrpc('http://localhost:1080')
```

Overview of RPC Commands

Commands for PLECS start with `plecs` followed by a dot. If you are using a Python or MATLAB client as described above, you invoke a command by appending it to the proxy object using a dot, e.g.:

```
proxy.plecs.load('C:/path/to/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 if it has unsaved changes.

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)'`.

Running a Simulation

The command

```
plecs.simulate(mdlName)
plecs.simulate(mdlName, optStruct)
```

runs a simulation of the model named `mdlName`. The optional argument `optStruct` can be used to override model parameters and solver options; for more information about this struct see section “Scripted Simulation and Analysis Options” (p. 329).

If any outports exist on the top level of the simulated model, the command returns a struct consisting of two fields, `Time` and `Values`. `Time` is a vector of simulation time stamps. `Values` is an $m \times n$ array containing the output values, where m is the number of time stamps and n is the number of output signals. The order of the signals is determined by the port numbers.

By default, the simulation result contains all simulation steps that the solver makes. This can be controlled with the solver options `OutputTimes` and `OutputTimesOption`, where `OutputTimes` is a double vector and `OutputTimesOption` is a string as described in Tab. 10.4 on p. 334.

> **Note**
>
> Running a simulation is a blocking command. The RPC server can execute only one blocking command at a time and will answer requests to execute a second blocking command with an error message. If you want to run multiple simulations in parallel via RPC, see "Running Multiple Simulations or Analyses in Parallel (RPC)".

Running an Analysis

The command

```
plecs.analyze(mdlName, analysisName)
plecs.analyze(mdlName, analysisName, optStruct)
```

runs 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" (p. 329).

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. By default, the simulation result contains all simulation steps that the solver makes. This can be controlled with the solver options `OutputTimes` and `OutputTimesOption`, where `OutputTimes` is a double vector and `OutputTimesOption` is a string as described in Tab. 10.4 on p. 334.

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.

> **Note**
>
> Running an analysis is a blocking command. The RPC server can execute only one blocking command at a time and will answer requests to execute a second blocking command with an error message. If you want to run multiple analyses in parallel via RPC, see "Running Multiple Simulations or Analyses in Parallel (RPC)".

Running Multiple Simulations or Analyses in Parallel (RPC)

↪ *Added in PLECS 4.6.*

Instead of a single `optStruct` argument you can also pass a list of option structures to the `simulate` and `analyze` commands:

```
plecs.simulate(mdlName, optStructs)
plecs.analyze(mdlName, analysisName, optStructs)
```

PLECS will then automatically distribute the individual simulations or analyses to the CPU cores available on your computer. After completion, the commands return a list containing the individual result structs or – in case of a runtime error – a string with the error message. To avoid memory problems when running a large number of parallel simulations, you can define an appropriate `OutputTimes` vector in the `SolverOpts` struct in order to reduce the number of data points.

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 xmlrpc.client
proxy = xmlrpc.client.ServerProxy("http://localhost:1080")

proxy.plecs.load("C:/Models/BuckParamSweep.plecs")
proxy.plecs.scope('BuckParamSweep/Scope', 'ClearTraces')

opts = {'ModelVars' :  { 'varL' : 50e-6 } }
```

(continues on next page)

(continued from previous page)

```
result = proxy.plecs.simulate("BuckParamSweep", opts)
proxy.plecs.scope('BuckParamSweep/Scope',
                  'HoldTrace', 'L=50uH')

opts['ModelVars']['varL'] = 100e-6;
result = proxy.plecs.simulate("BuckParamSweep", opts)
proxy.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 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 `OutputFormat`, `ModelVars`, `SolverOpts` and – when starting an analysis – `AnalysisOpts`, which are again structs as described below.

OutputFormat

The optional field `OutputFormat` is a string that lets you choose whether the results of a simulation or analysis should be returned as a RPC struct (`Plain`) or in binary form using the MAT-file format (`MatFile`). The binary format is much more efficient if the result contains many data points, but the client may not be able to interpret it, so the default is `Plain`.

↪ *Added in PLECS 4.4.*

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 Fig. 10.2.

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. Tab. 10.2 and Tab. 10.3 on p. 332 list the possible parameters.

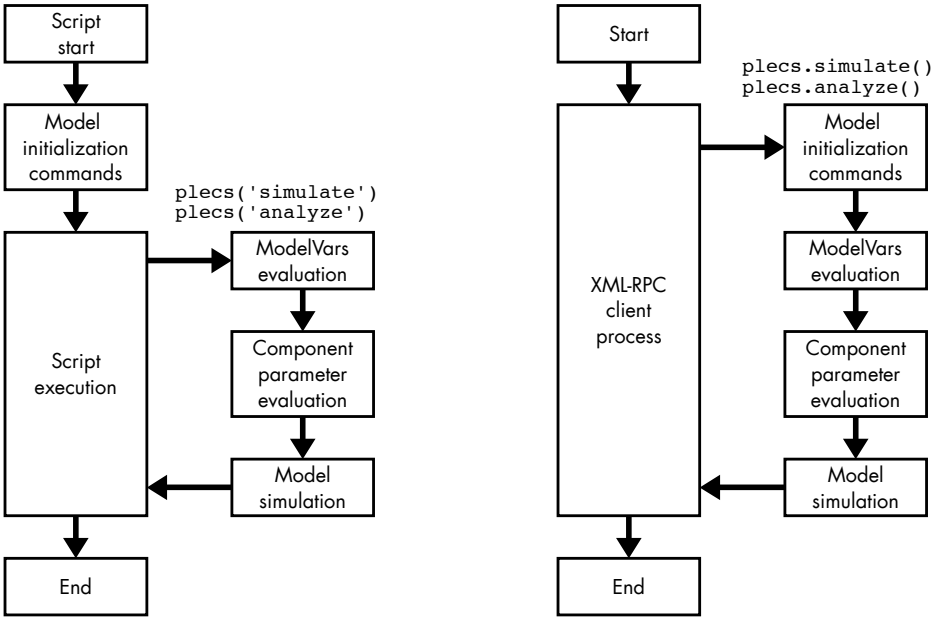


Fig. 10.2: Execution order for Simulation Scripts (left) and RPC (right)

Table 10.2: Solver Options in Scripted Simulations (see "PLECS Standalone Parameters" (p. 135))

Parameter	Description
`Solver`	The solver to use for the simulation. Possible values are `auto`, `dopri`, `radau` and `discrete` (see "PLECS Standalone Parameters" (p. 135)), and with a PLECS Spice license `sdirk` and `esdirk` (see "SPICE Simulation Parameters" (p. 406)).
`StartTime`	The start time specifies the initial value of the simulation time variable t at the beginning of a simulation, in seconds (s).

continues on next page

Table 10.2 – continued from previous page

Parameter	Description
`TimeSpan`	The simulation ends when the simulation time has advanced by the specified time span.
`StopTime`	This option is obsolete. It is provided to keep old scripts working. We strongly advise against using it in new code.
`OutputTimes` `OutputTimesOption`	These options control the simulation times that are included in the result of a scripted simulation. See Tab. 10.4 on p. 334 below. ↪ *Added in PLECS 4.5:* `OutputTimes` ↪ *Added in PLECS 4.7:* `OutputTimesOption`
`InitialSystemState`	Specifies the initial system state used for the simulation. This will override the **System State** setting in the simulation parameters (see "System State" (p. 142)). The system state struct can be retrieved after the completion of a simulation or steady-state analysis using the command: `plecs('get', mdlName, 'SystemState')` ↪ *Added in PLECS 4.7.*
`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. ↪ *Added in PLECS 4.1.*

continues on next page

Table 10.2 – continued from previous page

Parameter	Description
`MaxStep`	See the description for **Max Step Size**. This parameter is only evaluated for variable step solvers.
`InitStep`	See the description for **Initial Step Size**. This parameter is only evaluated for variable step solvers.
`FixedStep`	This option specifies the fixed time increments for the solver and also the sample time used for the state-space discretization of the physical model. It is only evaluated for the fixed step solver.
`AbsTol`	See the description for **Tolerances**.
`RelTol`	See the description for **Tolerances**.
`Refine`	See parameter **Refine factor**.

Table 10.3: SPICE Solver Options in Scripted Simulations (see "SPICE Simulation Parameters" (p. 406))

Parameter	Description
`MixedSignalSolverBypassOperatingPoint`	When set to `on`, the operating point analysis is skipped and the simulation starts from zero. See **Bypass operating point**. Possible values are `on` and `off`.
`MixedSignalSolverInitialConditions`	Controls whether user-provided initial conditions are honored by the operating point analysis. See **Initial conditions**. Possible values are `on` and `off`.

continues on next page

Table 10.3 – continued from previous page

Parameter	Description
`MixedSignalSolverMaxNewtonIterations`	Limits the number of Newton iterations per implicit stage during transient analysis. See **Max. number of iterations** under *Transient Analysis Options*.
`MixedSignalSolverNumThreads`	The maximum number of threads the solver may use during a SPICE simulation. See **Max. number of threads**.
`OperatingPointMaxNewtonIterations`	Limits the number of Newton iterations during the operating point analysis. See **Max. number of iterations** under *Operating Point Analysis Options*.
`OperatingPointRelativeNewtonTolerance`	The relative tolerance for the local integration error during the operating point analysis. See **Tolerances** under *Operating Point Analysis Options*.
`OperatingPointAbsoluteNewtonTolerance`	The absolute tolerance for the local integration error during the operating point analysis. See **Tolerances** under *Operating Point Analysis Options*.
`OperatingPointNodalVoltageNewtonTolerance`	The tolerance for nodal voltages computed during the operating point analysis. See **Tolerances** under *Operating Point Analysis Options*.
`SpiceTemperature`	The default temperature used for SPICE circuits, corresponding to the netlist directive `.temp`. See “Temperature (.TEMP)” (p. 415).
`SpiceShuntGMin`	A small conductance added from every node to ground, preventing ill-conditioned matrices. See **Minimum conductance to ground**.
`SpiceInductorRPar`	The ratio used to add a parallel resistance across all inductors. See **Parallel resistance to inductance ratio**.

continues on next page

Table 10.3 – continued from previous page

Parameter	Description
`SpiceInductorRSer`	A small resistance inserted in series with every inductor. See **Inductor series resistance**.
`SpiceCapacitorGPar`	The ratio used to add a parallel conductance across all capacitors. See **Parallel conductance to capacitance ratio**.

Table 10.4: Control of Output Times in Scripted Simulations

OutputTimesOption	Interpretation of OutputTimes
`specified`	The simulation result contains only the simulation times specified in the `OutputTimes` vector. This is also the default behavior if only the vector `OutputTimes` is provided and `OutputTimesOption` is omitted.
`additional`	The simulation result contains the simulation times specified in the `OutputTimes` vector *in addition to* all simulation steps that the solver makes.
`range`	The simulation result contains all simulation steps that the solver makes within the time span from `OutputTimes(1)` to `OutputTimes(end)`. If the vector `OutputTimes` contains more than two values, the additional simulation times are also included in the result.

↪ *Added in PLECS 4.7.*

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:

Table 10.5: Analysis Options in Scripted Analyses

Parameter	Description
`TimeSpan`	System period length; this is the least common multiple of the periods of independent sources in the system.
`StartTime`	Simulation start time.
`Tolerance`	Relative error tolerance used in the convergence criterion of a steady-state analysis.
`MaxIter`	Maximum number of iterations allowed in a steady-state analysis.
`JacobianPerturbation`	Relative perturbation of the state variables used to calculate the approximate Jacobian matrix.
`JacobianCalculation`	Controls the way the Jacobian matrix is calculated (`full`, `fast`). The default is `fast`.
`InitCycles`	Number of cycle-by-cycle simulations that should be performed before the actual analysis. This parameter can be used to provide the initial steady-state analysis with a better starting point.
`ShowCycles`	Number of steady-state cycles that should be simulated at the end of an analysis. This parameter is evaluated only for a steady-state analysis.
`FrequencyRange`	Range of the perturbation frequencies. This parameter is evaluated only for a small-signal analysis.
`FrequencyScale`	Specifies whether the sweep frequencies should be distributed on a `linear` or `logarithmic` scale. This parameter is evaluated only for a small-signal analysis.

continues on next page

Table 10.5 – continued from previous page

Parameter	Description
`AdditionalFreqs`	A vector specifying frequencies to be swept *in addition* to the automatically distributed frequencies. This parameter is evaluated only for a small-signal analysis.
`NumPoints`	The number of automatically distributed perturbation frequencies. This parameter is evaluated only for a small-signal analysis.
`Perturbation`	The full block path (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.
`MaxNumberOfThreads`	The maximum number of parallel threads that may be used during the analysis.
`ShowResults`	Specifies whether to show a Bode plot after a small-signal analysis. This parameter is evaluated only for a small-signal analysis. ↪ *Added in PLECS 4.3.*

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 struct array with all available parameters is returned.

```
plecs('set', componentPath, parameter, value)
```

sets the value of `parameter` of the PLECS component indicated by the `componentPath` to `value`.

The special parameter `CurrentCircuit` can be used to query the 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 Information, Warnings and Errors in Initialization Commands

↪ *Added in PLECS 4.1.*

During the execution of mask initialization commands the following commands add entries in the Diagnostics window:

```
plecs('info', infoMessage)
plecs('warning', warningMessage)
plecs('error', errorMessage)
```

Notice that the `plecs('error', errorMessage)` command by itself does not cause an early exit of the execution of the mask initialization commands, so it should generally be followed by a `return` statement to avoid subsequent errors.

See also the "Initialization Commands" (p. 103).

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

↪ *Added in PLECS 4.2.*

```
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 the "Cursors" section (p. 122).

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

↪ *Added in PLECS 4.2.*

```
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

↪ *Added in PLECS 4.2.*

```
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 Tab. 10.1 on p. 318.

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.

11

Code Generation

As a separately licensed feature, PLECS can generate C code from a simulation model to facilitate “Real-Time Code Execution” (p. 360) including “Real-Time Simulations” (p. 360) of physical systems. 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" (p. 34).

During normal "Offline Simulations" (p. 361), PLECS calculates new sets of state-space matrices on the fly when a new switching transition is detected. In generated C code, this is not possible because calculating the state-space coefficients is computationally too expensive 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 (p. 1090) 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.

Maximum Number of Switches

The number of ideal 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 (p. 1090), internally consist of more than one switch element.

Handling Naturally Commutated Devices

Switched physical models are difficult to handle in "Real-Time Simulations" (p. 360) 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 "Offline Simulations" (p. 361), PLECS handles such non-sampled switching events using an interpolation scheme (see "Interpolation of Non-

Sampled Switching Events" (p. 40)). 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 an offline 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 "Offline Simulations" (p. 361), 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" (p. 34)).

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 Rotational Model Settings (p. 917) and Translational Model Settings (p. 1062). The default switching algorithm and the algorithm for all electrical systems 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" (p. 360) 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 in Fig. 11.1. Two diodes are connected in series but opposing each other so that no current can flow regardless of the polarity of the source voltage:

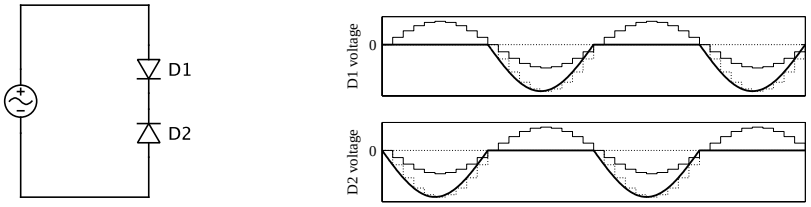


Fig. 11.1: Simulation of a circuit with two opposing diodes

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 block 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.

Ideal and Non-Ideal Switch Models in Electric Circuits

↪ *Added in PLECS 4.9.*

PLECS includes two different switch modeling options when generating code for selected electrical switches.

Ideal switch model

Ideal switches provide an ideal short or open circuit between its two elec-

trical terminals. The ideal switch model adds one or more switching elements to the electrical system, increasing the total number of switching combinations and state-space matrices as described previously.

Non-ideal switch model

A fixed-value resistor in parallel with a controlled current source represents the switching element. The state-space matrices do not change if a non-ideal switch is open or closed because the non-ideal switch resistance is constant. Therefore, non-ideal switches do not increase the total number of switching combinations and are suitable for models with large numbers of switches.

Each non-ideal switch includes an internal logic that models forced and natural commutation behavior. The switch state determines the controlled current source's update rule. When the switch is closed, the current update rule drives the voltage across the switching element to zero. The update rule for an open switch drives the current through the device to zero.

The update rules do not result in an ideal short or open circuit. Rather, a closed non-ideal switch behaves like a small inductor and an open non-ideal switch a small capacitor with additional series damping elements. The non-ideal switch resistance and the discretization step size determine the effective inductance and capacitance of the switch. The **Non-ideal switch resistance (code generation)** parameter in the Simulation Parameters menu (see "PLECS Blockset Parameters" (p. 143) and "PLECS Standalone Parameters" (p. 135)) sets the resistance.

The non-zero impedance of an open-circuit non-ideal switch will result in a leakage current through the open switch. In AC systems the leakage current may be unacceptably large at the fundamental frequency. Certain electrical switches, like the Breaker component (p. 583), offer the capability to increase the effective switch impedance at a defined frequency, thereby blocking a specific AC component of the current through the switch.

Unsupported Components

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

- Algebraic Constraint (p. 556)
- Brushless DC Machine (p. 584)
- Electrical Algebraic Component (p. 671)
- Rotational Algebraic Component (p. 907)
- Switched Reluctance Machine (p. 982)
- Translational Algebraic Component (p. 1052)
- Variable Capacitor (p. 1092)
- Variable Inductor (p. 1099)
- Variable Resistor (p. 1109)
- Variable Resistor with Constant Capacitor (p. 1110)
- Variable Resistor with Constant Inductor (p. 1111)
- Variable Resistor with Variable Capacitor (p. 1112)
- Variable Resistor with Variable Inductor (p. 1114)

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

Code Generation with PLECS Standalone

↪ *Changed in PLECS 4.4:* Added support for multi-tasking systems.

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()
void model_step(int task_id)
```
This function should be called at every simulation step to advance the model by one step. The second form is used if the multi-tasking mode is enabled (see “Scheduling” (p. 351)) and the model has more than one task.

```
void model_output()
void model_output(int task_id)
```
This function calculates the outputs of the current simulation step. It should be called at every simulation step before the `model_update` function. The second form is used if the multi-tasking mode is enabled.

```
void model_update()
void model_update(int task_id)
```
This function updates the internal states of the current simulation step. It should be called at every simulation step after the `model_output` function. The second form is used if the multi-tasking mode is enabled.

```
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.

The `model_output` and `model_update` functions are only generated if the corresponding option is set in the target specific settings (see “Target” (p. 351)). In this case, the `model_step` function is not generated.

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.

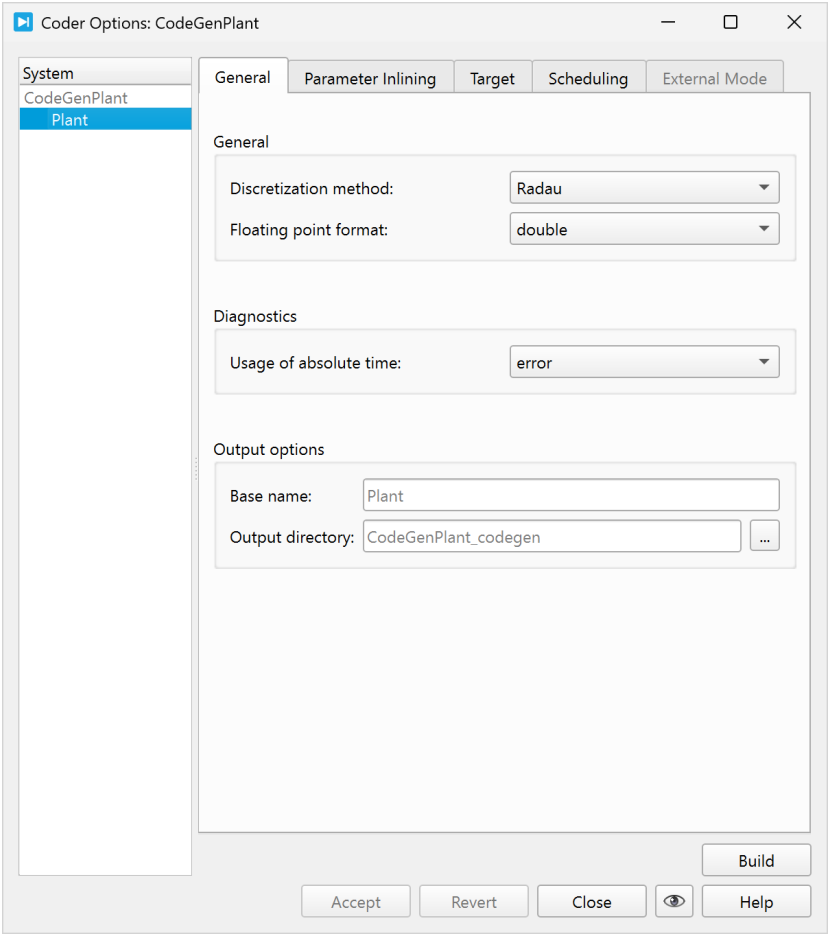


Fig. 11.2: Coder Options dialog window

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.

Code Generation subsystems

To enable a subsystem for code generation, select the subsystem, then choose **Execution settings...** from the **Edit > Subsystem** menu or the block's context menu. In the Subsystem Settings dialog check the option **Enable code generation**. Alternatively, use the dedicated Code Genereation Subsystem component from the PLECS component library.

Note that it is not possible to enable code generation for a subsystem that is contained by or that itself contains other subsystems that enable code generation. 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" (p. 978).

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 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" (p. 329). 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 method

This parameter specifies the algorithm used to discretize the physical model equations (see "Physical Model Discretization" (p. 39)).

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 (p. 976) 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. For information on how to install additional code generation targets, see Coder Configuration (p. 65).

↪ *Changed in PLECS 4.0:* Added support for specific hardware targets.

The **Generic** target has the following settings.

Generate separate output and update functions

If you choose this option, the `model_step` function is replaced by the two functions `model_output` and `model_update`. Choose this option when you need access to the outputs as early as possible during task execution (e.g. to transfer the values to an analog or digital output, update a logger, start a DMA transfer, etc.).

↪ *Added in PLECS 4.5.*

Scheduling

Tasking mode

This parameter allows you to choose between the single-tasking and multi-tasking modes.

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" (p. 39)) and continuous state variables of control blocks.

↪ *Added in PLECS 4.9.*

Task configuration

If you choose the multi-tasking mode, the dialog will show a table that lets you define a set of tasks. A task has a **Task name** and a **Sample time** that must be an integer multiple of the base sample time. The value `0` is replaced with the base sample time itself. If the selected target supports multi-core processing, a task is also associated with a **Core**. If the selected target has multiple processors, a task is also associated with a **CPU**. The radio buttons in the **Default** column specify the *default task* (see below).

↪ *Changed in PLECS 4.8:* Added support for targets with multiple processors.

Each task must have a unique name, and the core/sample time pairs must also be unique. Note that the task set must comprise a *base task* that is associated with the base sample time and core 0 (if applicable). For this reason, these columns in the first row are locked.

To assign a block or a group of blocks to a task, copy a Task Frame (p. 1011) into the schematic, open the Task Frame dialog to choose the desired task, and drag the frame around the blocks. Blocks that are not enclosed by a Task Frame are scheduled in the *default task*.

Note that blocks do not need to have the same sample time as the task that they are assigned to; the block sample time can be continuous or an integer multiple of the task sample time.

Thermal model task

In multi-tasking mode you can choose to execute all thermal model calculations in a dedicated task, typically on a separate core with a slower sample time in order to allow for the longer computation time for the calculation of switch losses. This asynchronous execution mode is supported only for power modules in `Sub-cycle average` configuration.

To configure a dedicated thermal model task, select its name in the combo box. The default setting, `use native task`, means that a thermal model is executed according to the Task Frame configuration synchronously with an associated electrical model.

External Mode

↪ *Added in PLECS 4.0.*

This tab is enabled only 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 (p. 942), XY Plot (p. 1130) and Display (p. 663) blocks in the model on the host computer.
- Change values of tunable parameters (see "Parameter Inlining" (p. 350)) in the model on the host computer and upload the changed values to an application running on a target device.

Task Transitions in Multi-Tasking Mode

↪ *Added in PLECS 4.4.*

If a block in one task receives one or more input signals from a block in another task, PLECS automatically inserts code to ensure that the signal data is transferred in a safe manner.

Task Transitions on the Same Core

If the source task is *faster* than the destination task and the sample time of the destination task is an integer multiple of the sample time of the source task (i.e. if the destination task is executed at instants at which the source task is also executed), PLECS inserts code equivalent to a Zero Order Hold (p. 1133) that operates with the slower sample time.

If the source task is *slower* than the destination task and the sample time of the source task is an integer multiple of the sample time of the destination task (i.e. if the source task is executed at instants at which the destination task is also executed), PLECS inserts code equivalent to a Delay (p. 639) that operates with the slower sample time.

In all other cases, PLECS uses a double buffer with a semaphore to transfer the data.

Note

The insertion of a Zero Order Hold or a Delay ensures that data is always exchanged in a deterministic manner. The drawback is that it introduces latency. If this is not desired, you can insert a Task Transition block (p. 1012) in front of the receiving block inside the destination task. This causes PLECS to transfer the data using a double buffer instead.
↪ *Added in PLECS 4.6.*

Task Transitions Between Different Cores

For task transitions between different cores, the write operation in the source task can occur simultaneously with the read operation in the destination task. PLECS uses a double buffer with two semaphores to guarantee that simultaneous read and write operations never access the same buffer.

Task Transitions Between Different CPUs

For task transitions between different CPUs, PLECS delegates the data transfer to external functions that must be provided by the target framework.

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 (p. 977)). 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**, PLECS will automatically generate code for the subsystem and use the compiled code instead of the model representation when you start a simulation.

In connection with the "Traces" feature of the scopes (see "Adding Traces" (p. 123)), this allows you to easily verify the fidelity of the generated code against a normal simulation.

Demo Model

- The demo model **"Plant Code Generation: Three-Phase 6-Pulse Thyristor Converter"** shows how to generate code from a physical model and simulate the model in CodeGen mode.
- Find it in PLECS under **Window** > **Demo Models** > Power Supplies.

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" (p. 347).

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"). 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" (p. 358).

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.

Table 11.1: Comparison of Code Generation Targets

	RSim Target	Real-Time Target
Purpose	Rapid, non-real-time simulations.	Real-time simulations.
Technique	A compressed description of the circuit schematic is embedded in the code and interpreted at run time.	Signal and state-space equations are inlined as ANSI C code.
Limitations	None	Limited support for naturally commutated devices and non-linear components.

continues on next page

Table 11.1 – continued from previous page

	RSim Target	Real-Time Target
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 *build time*, the generated code will run without a license. Otherwise, a PLECS license is required at *run time*.	Requires a PLECS Coder license at *build time*.

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” (p. 342).

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.

Table 11.2: Library Files for Rapid Simulations

Platform	Library Files
Windows 64-bit	`plecs/bin/win64/plecsrsim.dll`
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” (p. 58).

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.

Real-Time Code Execution

A typical application of C code generation from PLECS models is to execute the generated code on hardware targets with a fixed step size and deterministic timing. Such targets typically interact with external hardware through physical I/O, such as analog and digital inputs and outputs. This enables controllers, plant models, or complete systems to be tested under conditions that are close to those of a hardware prototype.

When C code is generated from a physical system (p. 342), it is typically intended to run on a real-time simulator hardware, such as the RT Box. This use case is referred to as "Real-Time Simulations". C code generated from PLECS subsystems without physical systems typically represents control logic and can be executed on general-purpose embedded targets, such as microcontroller platforms.

> **Note**
>
> This manual does not cover target-specific details of real-time code execution and "Real-Time Simulations" on dedicated hardware. Refer to the documentation provided with the corresponding target support package. To install target support packages, see Coder Configuration (p. 65).

Real-Time Simulations

The simulation of a physical system is called *real-time* if simulated time and actual elapsed time progress at the same rate. This condition applies when a simulation model is executed as generated C code (p. 341) on dedicated real-time simulator hardware, such as the RT Box, but not to offline simulations running in PLECS on a PC.

On a real-time simulator, the simulation is advanced with a fixed time step, and the calculations for each step must be completed within the corresponding time interval. For example, with a fixed time step of $1\,\mu$s, defined in the Scheduling tab (p. 351) of the Coder options, the simulator must complete all calculations within $1\,\mu$s of elapsed time.

In contrast to "Offline Simulations", a fixed real-time step does not allow events such as zero crossings to be located precisely in time. The accuracy of a real-time simulation is therefore directly related to the resolution provided by the fixed time step. To keep the time step small, model complexity and, in particular, the number of ideal switches should be minimized. Dedicated modeling

techniques such as Sub-cycle Averaging (p. 489) can reduce model complexity while preserving the relevant switching behavior. Alternatively, parts of the model can be assigned to dedicated solvers that resolve switching behavior at a very fine time resolution, as with the Nanostep solver (p. 363) on the RT Box.

A typical use case for real-time simulation is testing external hardware, such as controllers. The controller exchanges signals with the simulator as it would with a real hardware plant. Ideally, the simulator represents the physical plant accurately enough that the controller behaves the same as it would with the actual hardware. This use case is commonly referred to as hardware-in-the-loop (HIL) testing.

Offline Simulations

The counterpart to *real-time* simulations are *offline* simulations, which are the usual way of using PLECS on the user's PC. In contrast to real-time simulations running on a dedicated hardware target, offline simulations running on the PC are not tied to actual elapsed time. They may take as much or as little time as needed to advance the model, and can therefore run faster or slower than real time.

Offline simulations can use solver settings and event-handling techniques that are not possible under hard real-time constraints. For example, variable-step solvers (p. 135) can adapt the step size according to the model dynamics and tolerance settings. For event handling, PLECS can interpolate backward in simulated time to locate zero-crossing events more accurately than would be possible with a fixed real-time step.

PLECS is designed to simulate typical real-time concepts also in offline simulations. Features that are primarily intended for dedicated hardware targets, such as code generation or target-specific library components, therefore provide an offline implementation as well. For example, "Simulating a Subsystem in CodeGen Mode" (p. 354) allows the real-time behavior of generated code to be emulated in an offline simulation. The simulation results can be compared to a variable-step simulation before the code is deployed to a real-time hardware target.

Real-time Code Execution on the RT Box

The RT Box is Plexim's real-time simulation platform designed to execute generated C code from PLECS models. The code may represent a physical system as described in "Code Generation for Physical Systems" (p. 342), control logic, or both.

For control code, the same concepts apply to the RT Box as to other general-purpose embedded targets. The following sections therefore focus on concepts specific to “Real-Time Simulations” (p. 360) of physical systems.

> **Note**
>
> This manual does not cover target-specific details of real-time code execution and “Real-Time Simulations” (p. 360) on RT Box. Refer to the documentation provided with the RT Box target support package. To install target support packages, see Coder Configuration (p. 65).

RT Box Solvers

The RT Box offers three solvers for real-time simulation of physical systems. Depending on the RT Box model, the CPU, FlexArray, and multiple Nanostep solvers are available. The suitable solver depends on the topology of the simulated system and the required simulation time step.

CPU

The CPU solver supports real-time simulation of the broadest range of PLECS library components, including components from electrical, mechanical, magnetic, thermal, and control domains. It is available on all RT Box models.

Time step

The achievable CPU time step depends on model complexity and is $T_{sim} \geq 1\,\mu\text{s}$.

Workflow

Code generated for an RT Box target is executed on the CPU solver by default with no additional solver settings.

When simulating power converters on CPU, it is recommended to use the Sub-cycle average (p. 489) configuration, available for all components in the “Power Modules” (p. 488) library section and for several components in the “Nanostep” (p. 493) section.

Switch model

By default, switches in generated code for the CPU solver are modeled as ideal switches. For components that provide the **Switch model (CPU code generation)** parameter, non-ideal switch models can be used by setting this parameter to `Non-ideal`. See “Ideal and Non-Ideal Switch Models in Electric Circuits” (p. 344) for details.

FlexArray

The FlexArray solver is an FPGA-based solver intended for fast real-time simulation of electrical circuits. It is available on RT Box 2, 3 and 4.

Time step

The achievable FlexArray time step depends on model complexity and is $T_{sim} \geq 50\,\text{ns}$.

Workflow

Code generated from electrical systems is assigned to the FlexArray solver by selecting `FlexArray` in Target of Electrical Model Settings (p. 675) block. Code of control logic and other physical domains continues to run on the CPU.

When simulating power converters on FlexArray, it is recommended to use the Sub-cycle average (p. 489) configuration, available for all components in the "Power Modules" (p. 488) library section and for several components in the "Nanostep" (p. 493) section.

Switch model

The FlexArray solver uses the non-ideal switch model (p. 344) for switches in generated code.

Nanostep

The Nanostep solver is an FPGA-based solver for selected switched-mode power converter topologies from the "Nanostep" (p. 493) library section. It is available on all RT Box models.

Time step

RT Box CE and RT Box 1 provide one Nanostep solver with a fixed time step of $7.5\,\text{ns}$. RT Box 2, 3 and 4 provide two Nanostep solvers, each with a fixed time step of $4\,\text{ns}$.

At every Nanostep solver step, the gate signals are sampled, the state variables are updated, and zero crossings are detected.

Workflow

To use the Nanostep solver, build the converter using components from the "Nanostep" (p. 493) library section and select the `Nanostep / Switched` Configuration, where applicable. To probe signals from Nanostep components use dedicated Nanostep Probe and Nanostep Scope components available in the RT Box target support packages.

Solver weight

Each Nanostep solver has a capacity of three solver weight units. A

Nanostep component (p. 493) with weight 1 uses one third of this capacity and a component with weight 3 uses one complete Nanostep solver. For example, an RT Box 2 provides two Nanostep solvers and can therefore execute up to six Nanostep components with weight 1, or up to two components with weight 3.

Switch model

The Nanostep solver simulates switches using the ideal switch model (p. 344).

12

Functional Mockup Interface

↪ *Added in PLECS 4.7.*

The **Functional Mock-up Interface (FMI)** is an open standard for exchanging and coupling simulation models between different engineering tools. It defines a common format in which a model can be exported from one tool as a **Functional Mock-up Unit (FMU)**—a file that contains the model description, parameters, and compiled code or equations. Other tools that support FMI can import and run these FMUs, typically in order to connect them to other models. The FMI standard is used in areas such as control design, system simulation, and engineering analysis, where models from different domains or software environments need to interact. By providing a consistent way to share and execute models, FMI helps avoid tool-specific integrations and makes it easier to reuse models across projects and development stages.

FMI Standard

The FMI standard[8] defines three interface types:

- The **Model Exchange (ME)** interface exposes an ordinary differential equation system to the solver of the importing tool. The integration algorithm of the importing tool is responsible for advancing time, setting states, handling events, etc.
- The **Co-Simulation (CS)** interface is designed to couple two separate solvers that are synchronized at distinct communication times. The FMU can either just serve as a communication link between the importing tool and the exporting tool (tool-coupling) or contain both a model and its solver as runnable code.
- The **Scheduled Execution (SE)** interface provides support for the concurrent computation model partitions on a single computational resource. This interface is not supported by PLECS.

FMI Version

PLECS implements versions 2.0 and 3.0 of the FMI standard. Version 3.0 introduces the following features:

- More integer data types and a 32-bit float type. (Note that PLECS does not support 64-bit integers.)
- Advanced co-simulation features (see below).

PLECS supports both the import and export of Co-Simulation and Model Exchange FMUs. To import an FMU that was exported from another tool, use the FMU block (p. 686). The remainder of this chapter describes the export of FMUs from PLECS.

> ↪ *Changed in PLECS 5.0:* Added support for Co-Simulation FMUs and the FMI 3.0 standard.

[8] https://fmi-standard.org

Common Concepts for FMU Export

An FMU file is a ZIP archive that contains several standardized components:

- **Model description**: an XML file that defines the exposed model variables (such as inputs, outputs, states, parameters) and model capabilities
- **Platform binaries**: platform-specific compiled libraries conforming to the FMI Application Programming Interface (API) for different operating systems that implement the model behavior
- **Resources**: Optional additional files
- **Documentation**: Optional documentation

PLECS can export any atomic subsystem to an FMU (see “Virtual and Atomic Subsystems” (p. 978)). The input and output terminals of the subsystem determine the inputs and outputs of the FMU. If the subsystem has a mask, the mask parameters define the FMU parameters (see “Mask Dialog” (p. 98)), and the mask description defines the FMU description.

When you export an atomic subsystem to an FMU, PLECS will embed the subsystem schematic in a compressed and encrypted model file inside the FMU resources folder. The FMU will parse and execute this model at runtime.

PLECS Model Exchange FMU

When you export a subsystem to a Model Exchange FMU, PLECS must discretize the model so that the FMU is executed with a fixed sample time. The platform binaries contain the code required to execute such a discretized model.

> **Note**
>
> Exporting a Model Exchange FMU from PLECS requires a PLECS Coder license. The resulting FMU can be used on a target computer without the need to install PLECS, and no license is required to run the FMU.

Limitations

Discrete-Variable Sample Times

Since PLECS discretizes a models when exporting it to an ME-FMU, the model cannot contain components that have a variable sample time.

Thermal Description Parameters

Thermal description parameters are inlined and cannot be modified in the exported FMU.

Masked Subsystems

The shared FMU libraries contain a basic MATLAB parser for interpreting simple initialization commands of masked subsystems (see “Initialization Commands” (p. 103)). This parser supports only a limited subset of the MATLAB language.

During an FMU export, PLECS checks whether the exported model contains masked subsystems whose mask initialization commands cannot be executed by the internal MATLAB interpreter. In such a case PLECS will show a warning message, remove the mask initialization commands and instead replace all parameter expressions of all components below the mask with their evaluated values, i.e. inline them. As a consequence, the behavior of such inlined subsystems can no longer be modified in the exported FMU via FMU parameters.

This limitation applies for instance to the following components from the PLECS library:

- Diode with Reverse Recovery (p. 646)

- Non-Excited Synchronous Machine (p. 850)
- Synchronous Reluctance Machine (p. 1007)
- Thyristor with Reverse Recovery (p. 1028)
- Transmission Line (3ph) (p. 1071)
- all components from the “Power Modules” library (p. 488)
- all components from the “Nanostep” library (p. 493)

PLECS Co-Simulation FMU

↪ *Added in PLECS 5.0.*

PLECS exports a *tool-coupling* Co-Simulation FMU. The platform binaries implement a communication link between the importing tool and a PLECS installation that must be running on the target computer. When a simulation is started by the importing tool, the FMU sends the embedded model file to PLECS and synchronizes the simulation progress between PLECS and the importing tool. To enable the co-simulation mode in PLECS, check the option **Enable FMU Co-Simulation endpoint** on the **General** page of the **PLECS Preferences** dialog (see "Configuring PLECS" (p. 60)).

> **Note**
>
> Exporting a Co-Simulation FMU from PLECS does not require a separate license. However, to use the resulting FMU, PLECS must be installed on the target computer, and a PLECS license is required at runtime.

FMU with Discrete Inputs

If you export a PLECS Co-Simulation FMU with discrete input signals such as gate signals, it is recommended to select version 3.0 of the FMI standard. This will allow the importing tool to utilize the **Event Mode** in order to properly handle the discontinuities introduced by gate signal changes.

Limitations

Thermal Description Parameters

Thermal description parameters are inlined and cannot be modified in the exported FMU.

Exporting an FMU

To export an atomic subsystem to an FMU, select the Subsystem block, then choose **Export FMU...** from the **Edit > Subsystem** menu or the block's context menu. This will open the **FMU Export** dialog that allows you to configure the FMU as described below. When you click on **Export**, the FMU is created.

General Options

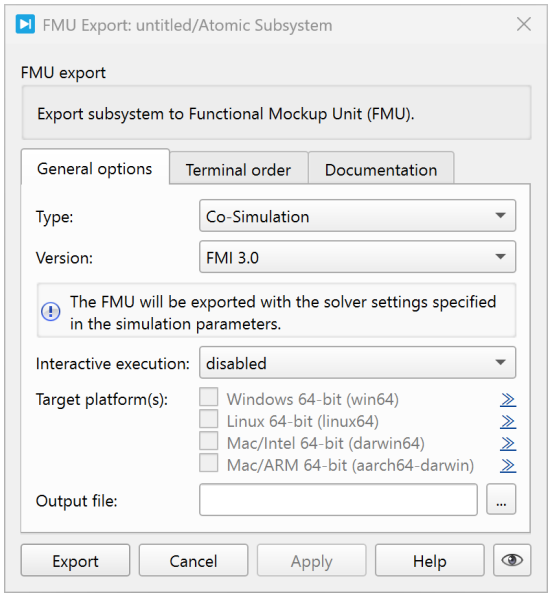


Fig. 12.1: General Options tab of FMU Export dialog

Type
: This parameter lets you choose between the **Co-Simulation** and **Model Exchange** types.

Version
: This parameter lets you choose between the FMI standard versions 2.0 and 3.0. Note that the importing tool must support the same version in order to use the FMU.

Sample time
: This parameter is applicable only to Model Exchange FMUs. It speci-

fies the sample time used to discretize the exported subsystem. It is recommended to run a simulation with a fixed-step solver using this sample time before the export to verify the proper behavior of the discretized model.

Interactive execution
This parameter is applicable only to Co-Simulation FMUs. If this is set to **enabled** and the importing tool supports it, PLECS will open the schematic of the FMU model at runtime and allow the importer to interact e.g. with scopes.

Target platform(s)
These checkboxes let you choose the binary platform(s) on which you intend to run the exported FMU. PLECS will embed the corresponding shared libraries into the FMU.

Before you can export an FMU for the first time, you need to install the shared library files on your computer. This is indicated by blue arrows next to the checkbox items as shown below. When you click on any of them, a dialog opens that lets you download and install the required package automatically.

Output file
This parameter specifies the file path of the exported FMU.

Terminal Order

On this page you can configure the order in which the input and output terminals of the subsystem appear in the exported FMU.

Documentation

Icon file
This parameter allows you to specify a PNG image file that is included as a model icon in the exported FMU.

Documentation
This parameter allows you to provide documentation that is included in the exported FMU. You can choose to include a single HTML file or a folder with HTML and resource files.

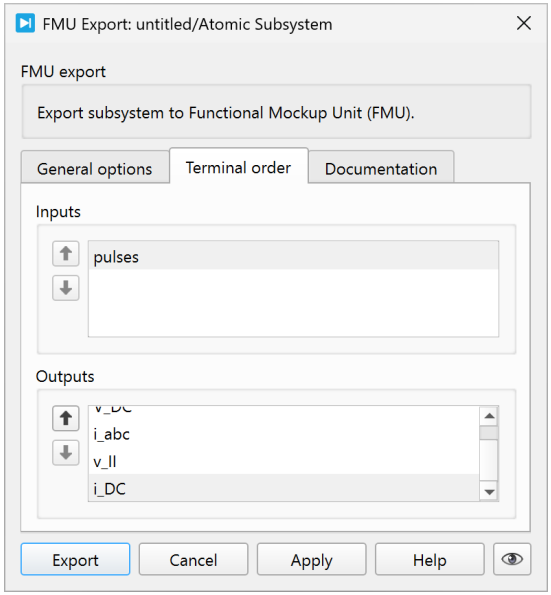


Fig. 12.2: Terminal Order tab of FMU Export dialog

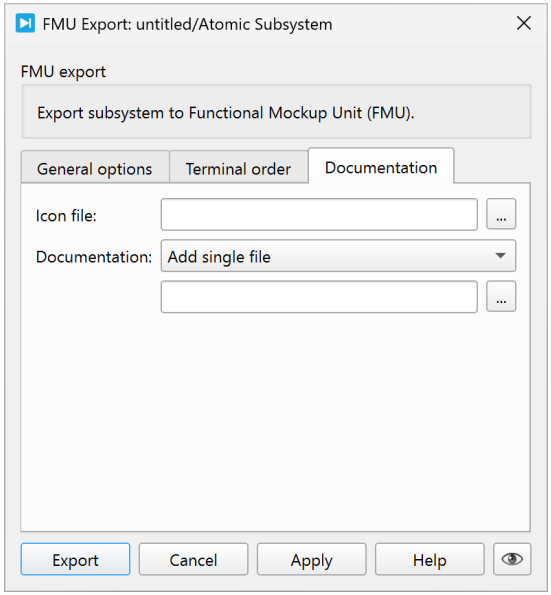


Fig. 12.3: Documentation tab of FMU Export dialog

13

Diff Tool

↪ *Added in PLECS 4.7.*

The Diff tool can be used to compare two PLECS models or two subsystems of PLECS models. The result of a Diff is a side-by-side comparison of the two models showing all the differences between them. Depending on the kind of change, a suitable view is used, e.g. a schematic to show inserted or deleted components or a parameter dialog to show changed parameter values. The Diff browser can be used to navigate through the differences. A "relevance" value for each difference helps you to identify the most important changes.

This chapter is divided into four sections. The first section describes how to select two models or subsystems and compute a Diff for them. The second section gives an overview on how the result of the Diff is presented to the user. The third section explains the algorithms PLECS uses to compute the Diff. Finally, the fourth section lists the limitations of the Diff tool.

How to Compute a Diff

Computing a Diff in PLECS Standalone

The Diff tool can compare either two complete PLECS models or two subsystems contained in one or two PLECS models. To compute a Diff, choose the option **Compare** in the **File** menu. This opens the Diff dialog.

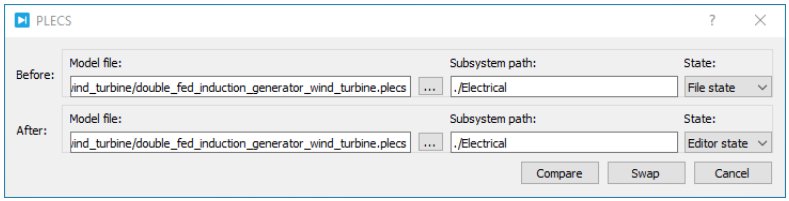


Fig. 13.1: Standalone Diff dialog

This dialog has the two rows “Before” and “After” to specify the two entities that should be compared. You can use drag&drop to specify a model or a subsystem for comparison. Each row in the Diff dialog contains the following fields:

Model file
: This field is used to specify the path of the model file. Depending on the models that are currently open in PLECS, this field is already filled with a reasonable suggestion. You can also use drag&drop or the **Open file browser** button (**...**) to choose a file.

Subsystem path
: This field can be used to specify a subsystem for the comparison. If you want to compare subsystems you have to specify a subsystem for both the “before” and “after” row. You cannot compare a model with a subsystem. The subsystem path should also contain the name of the model, which can be abbreviated by a dot (.). If you use only the model name or a dot or if you leave this field empty, the whole model will be compared. Depending on the subsystems that are currently selected in PLECS, this field is already filled with a reasonable suggestion. You can also drag&drop a subsystem from a schematic window.

State
: This is a drop-down list that can specify either one of the values **File state** or **Editor state**. If you choose **File state**, the state the model has in the model file is used for comparison. If you choose **Editor state**, the

unsaved changes of an open model are used for comparison. This can be useful to review changes before saving a model.

Differences are defined as editing steps from the “before” model to the “after” model. A component present in the “before” model but missing in the “after” model will be displayed as a deletion. If you decide that you would like the differences to be presented the other way around, use the **Swap** button to swap all fields in the “before” and “after” row.

Once you have filled out all fields, press the **Compare** button to start the Diff computation. A progress bar will show the current state of the Diff computation. If you want to abort the Diff computation, press **Cancel**. Once the Diff computation has finished, the Diff result window will show the result.

Command Line Interface

↪ *Added in PLECS 4.9.*

PLECS Standalone can be used as an external diff viewer in version control systems such as Git. To compare two files, use the following command line syntax:

```
plecs -diff leftModelFile rightModelFile
```

or on macOS:

```
PLECS.app/Contents/MacOS/PLECS -diff leftModelFile rightModelFile
```

Computing a Diff in PLECS Blockset

The Diff tool can compare either two complete PLECS circuits contained in one or two Simulink models or two subsystems contained in one or two PLECS circuits. To compute a Diff, choose the option **Compare** in the **File** menu. This opens the Diff dialog.

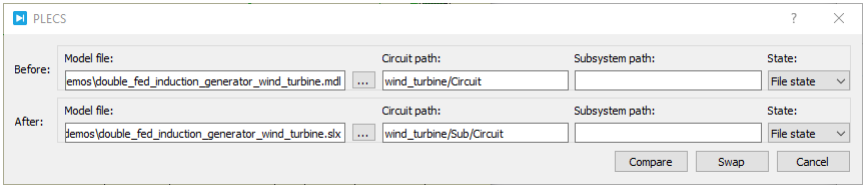


Fig. 13.2: Blockset Diff dialog

This dialog has the two rows “Before” and “After” to specify the two entities that should be compared. You can use drag&drop to specify a model or a subsystem for comparison. Each row in the Diff dialog contains the following fields:

Model file
: This field is used to specify the path of the Simulink model file. Depending on the models that are currently open, this field is already filled with a reasonable suggestion. You can also use drag&drop or the **Open file browser** button (**...**) to choose a file. Valid model file extensions are **mdl** and **slx**.

Circuit path
: This field is used to specify the path of the PLECS Circuit within the Simulink model. It should start with the Simulink model name and end with the PLECS Circuit name. Depending on the models that are currently open, this field is already filled with a reasonable suggestion.

Subsystem path
: This field can be used to specify a subsystem for the comparison. If you want to compare subsystems you have to specify a subsystem for both the “before” and “after” row. You cannot compare a circuit with a subsystem. The subsystem path should also contain the name of the circuit, which can be abbreviated by a dot (.). If you use only the circuit name or a dot or if you leave this field empty, the whole circuit will be compared. Depending on the subsystems that are currently selected in PLECS, this field is already filled with a reasonable suggestion. You can also drag&drop a subsystem from a schematic window.

State
: This is a drop-down list that can specify either one of the values **File state** or **Editor state**. If you choose **File state**, the state the circuit has in the model file is used for comparison. If you choose **Editor state**, the unsaved changes of an open model are used for comparison. This can be useful to review changes before saving a model.

Differences are defined as editing steps from the “before” model to the “after” model. A component present in the “before” circuit but missing in the “after” circuit will be displayed as a deletion. If you decide that you would like the differences to be presented the other way around, use the **Swap** button to swap all fields in the “before” and “after” row.

Once you have filled out all fields, press the **Compare** button to start the Diff computation. A progress bar will show the current state of the Diff computation. If you want to abort the Diff computation, press **Cancel**. Once the Diff computation has finished, the Diff result window will show the result.

How to Interpret the Results

The result of computing a Diff is shown in the Diff result window. This window contains a Diff browser and two views, one for the “before” model and one for the “after” model.

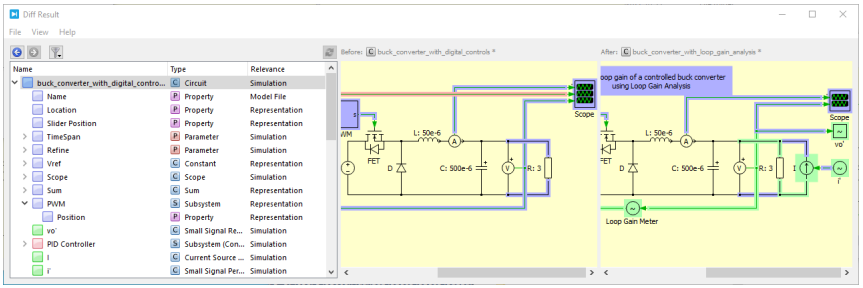


Fig. 13.3: Diff result window

The Diff Browser

The Diff browser can be used to navigate through the differences between the two models. It is similar to the Circuit Browser shown in schematic windows (see “Circuit Browser” (p. 131)). However, there are three important differences:

1. The Diff browser shows not only components but also all other kinds of items stored in a model, e.g. parameters, properties like the positions of components, analyses and simulation scripts
2. The Diff browser only shows the items that actually changed
3. The Diff browser merges both models into one tree, i.e. an item in the tree represents the corresponding item in both models (if it exists)

There are cases where an item is located in different schematics in the “before” and “after” models. For example, if a resistor is in the root schematic in the “before” model and within a subsystem in the “after” model, it will appear twice in the Diff browser.

The Diff browser shows the three columns **Name**, **Type** and **Relevance**.

Name

This column shows the name of the corresponding item. The hierarchical structure of the model is shown as a tree. Use the little arrows to navigate into nested items. The colored icons next to the item names illustrate the kind of change:

- A red icon means that the corresponding item was deleted from the "before" model
- A green icon means that the corresponding item was inserted into the "after" model
- A blue icon means that the corresponding item was modified

For renamed items, both the "before" and "after" names are shown, with an arrow (→) between them.

Type

This column shows the type of the corresponding item. The type is also illustrated by the icon. There are the following types:

- C PLECS Circuit
- S Subsystem
- C Component (other than a Subsystem)
- C Connection
- A Annotation
- T Terminal
- P Parameter
- S Setting
- A Component assertion
- P Scope plot
- A XY-plot axis
- F Fourier plot
- P Probe (entry for a component in a Probe block or a mask probe)
- M Mask probe (entry in the list of probes for a masked subsystem)
- P Simulink probe (entry in a Simulink Probe block)
- P Netlist probe (entry in a PLECS Spice Netlist block)
- S State machine state (includes entry points and junctions)
- T State machine transition
- A State machine annotation

- State machine parameter
- Analysis
- Simulation script
- Property (of an item of any type)

This list is exhaustive, i.e. every data item in a model has one of the above types.

> **Note**
>
> The item types in the Type column of the Diff browser are distinct from the component types shown in the Type column of the Circuit browser.

Relevance

This column shows a "relevance" for each item shown in the Diff browser. The relevance gives an estimation for the importance of a change. It can have the following values:

Simulation
: The difference may have an impact on the result of a simulation. Example: changing a component parameter value.

Representation
: The difference does not influence the simulation, but how a model is displayed in the GUI. Example: changing the position of a component in the schematic.

Model file
: The difference does not influence the simulation and is not directly visible in the GUI either, but it is visible in the model file. Example: changing the PLECS version in which the model is saved.

Tool Buttons

The Diff browser also shows the following tool buttons:

Previous View

Use this button to return to the previous item that was shown in the views.

Next View

Use this button to go to the next item in the list of visited items.

Filter Options

Use this button to filter items according to their relevance (see “Relevance”).

Refresh

Use this button to recompute the Diff after one of the compared original models changed.

The Diff Views

To the right of the Diff browser, there are two views that show the currently selected item in the appropriate context, e.g., a changed component is shown in the parent schematic, a changed parameter is shown in the parameter dialog. Each view shows the path of the currently visible item at the top. By clicking on an element of the path, you can navigate to the corresponding item.

By default, the left view shows the “before” model and the right view shows the “after” model. You can swap the two views in the **General** tab of the PLECS preferences dialog (see “Configuring PLECS” (p. 60)). Note that swapping the views only changes their location in the window and has no influence on the roles of the “before” and “after” models.

As in the Diff browser, items deleted from the “before” model are marked in red, items inserted into the “after” model are marked in green and modified items are marked in blue. A spotlight is used to facilitate finding the changed item.

You can navigate through the models as in a normal Schematic window. However, editing is disabled, because the Diff result is only valid for the particular state the models were in when you pressed the **Compare** button.

How Diff Works

Classical Diff algorithms (like Miller & Myers, 1985) operate on linear lists. This works well for continuous text but cannot account for the hierarchical structure of tree-like data. A PLECS model is a collection of nested linear and hierarchical data. To compute a useful Diff between two PLECS models, we therefore employ an algorithm that uses a linear Diff algorithm (Miller & Myers, 1985) for linear structures and a hierarchical Diff algorithm (Zhang & Shasha, 1989 and Paassen, 2018) for hierarchical structures and applies them recursively to nested data as appropriate. For example, the hierarchical algorithm is used to compare the trees of subsystems and components, and within a C-Script block, the linear algorithm is used to compare each C code snippet.

The result of the algorithm is a mapping of items in the "before" model to items in the "after" model. Items that are mapped to each other are considered to be "the same" item. Items that are in the mapping and identical are reported to be unchanged. Items that are in the mapping and not identical are reported to be modified. Items that are not in the mapping but present in the "before" model are reported to be deleted. Items that are not in the mapping but present in the "after" model are reported to be inserted.

References

- W. Miller and E. W. Myers, "A file comparison program", Softw: Pract. Exper., 15: 1025-1040, 1985, https://doi.org/10.1002/spe.4380151102
- K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance between Trees and Related Problems", SIAM Journal of Computing, 18(6): 1245-1262, 1989
- B. Paassen, "Revisiting the tree edit distance and its backtracing: A tutorial", 2018, https://arxiv.org/abs/1805.06869

Limitations

Model Conversion

PLECS computes Diffs based on the internal representation of a model rather than the text contents of a model file. If a model is not open yet, it will be loaded into the internal representation before the comparison. If the model file was saved with a different version of PLECS, this internal representation may differ from the model file due to differences between the PLECS versions. This means that the Diff shown in PLECS may not contain certain differences that are present in the model files. PLECS shows a warning if the versions of the compared models do not match.

Non-Intuitive Results

The result shown in the Diff result window may sometimes not correspond to what you would consider the "correct" result. There are mainly two reasons for this:

1. While there are many ways to edit a model, the Diff algorithm (see "How Diff Works") detects only three distinct edit operations: deletions, insertions and modifications. Also, the order of the items and their parent-child relationships within the models must be preserved in order for an item to be considered the same item in the "before" and "after" models.

2. The ordering of the items in the model is not directly visible in the GUI. This corresponds to the order in which the items are stored in the model file and has nothing to do with the coordinates of a component within a schematic.

An example of a non-intuitive result may be the following: Consider a model containing only a resistor and a diode in the root schematic, in this order. Now the user adds a subsystem and moves the diode into it. As expected, the Diff result will be an inserted subsystem in the "after" model and a modified diode, located in the root schematic in the "before" model and within the subsystem in the "after" model. However, if the user, instead of moving the diode into the subsystem, moves the resistor into the subsystem, the Diff result will be a deleted resistor in the "before" model and an inserted subsystem and inserted nested resistor in the "after" model.

The reason for the second, unexpected result is that inserting a subsystem and moving the resistor into it changes the order of the components. Because the subsystem is added at the end of the list of components in the schematic, and the resistor is removed from the beginning of the same list, moving the resistor into the subsystem changes its position relative to the diode. The resistor can

therefore not be part of the mapping and is detected as deleted and inserted instead of just modified.

Though situations like the one described above exist, the result that the Diff algorithm produces is always correct in the sense that it represents a minimal valid sequence of deletions, insertions and modifications that lead from the “before” model to the “after” model.

14

SPICE

↪ *Added in PLECS 5.0.*

PLECS Spice is a separately licensed feature available only for PLECS Standalone. PLECS is already equipped with a schematic editor, its popular scope, and a user-friendly user interface. The ingredients included in the SPICE bundle that turns the standard PLECS software into a fully-featured SPICE analysis tool are:

Modified Nodal Analysis
: The set of equations solved in a SPICE simulation is obtained using MNA, which produces differential algebraic equations (DAE) that include Kirchhoff's current law (KCL) explicitly as algebraic constraint.

Mixed-Signal Solver
: Differential algebraic solvers capable of simulating state-space and MNA models together.

Netlist Parser
: SPICE models are often distributed as netlists. A netlist is a text-based description of a circuit. The netlist component in PLECS Spice supports various dialects to enable the import of netlists from varied sources.

Compact Models
: Netlists provided by manufacturers as accurate models of real devices often rely on parametrization of built-in, well-established, semiconductor device models. These compact models are highly optimized for SPICE simulation.

The strength of the standard PLECS software lies in its judicious use of ideal switches: by avoiding the computational burden of simulating the switching transients of detailed semiconductor models, it delivers exceptional speed and robustness for system-level integration. But although ideal switch models are sufficient and well-suited for system-level analysis, simulating detailed switching dynamics is valuable for informing semiconductor selection and validating

component choices before procurement. To meet this need, Plexim offers PLECS Spice, a bundle of circuit simulation tools for the analysis of systems involving detailed component models described by SPICE netlists. The feature enables users to analyze parasitic effects and detailed device-level behavior without leaving the PLECS development environment.

PLECS Spice introduces the Netlist component (p. 845), which allows to import and connect SPICE netlists to other components in a schematic. The import tool makes it easy to integrate semiconductor SPICE models provided by manufacturers to your converter designs. Since the solvers included in PLECS Spice are capable of simulating schematics containing both PLECS and SPICE components, the controls for this converter can be created using the PLECS control domain. With the capabilities of both modeling approaches unified in one platform, users no longer need to build the same model twice.

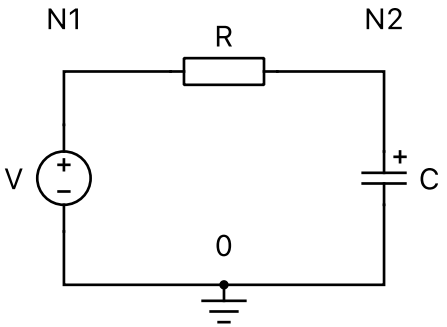
How PLECS Spice Works

Section “Physical Model Equations” (p. 34) describes how the standard PLECS software simulates physical models using piecewise linear state-space equations. The approach is well-suited for circuits involving mostly linear electrical components, but falls short in the presence of highly nonlinear semiconductor models.

This section briefly describes the inner workings of the transient analysis in SPICE. Using a simple example, it introduces the concept of modified nodal analysis (MNA), shows how MNA is used to derive the differential algebraic equations (DAE), and explains how netlists are used to describe electric circuits.

Modified Nodal Analysis

Example RC Circuit



The connections between the components of a circuit are called **nodes**. In the above example, the nodes are labeled N1, N2 and 0. The node labeled 0 is connected to ground. The **nodal voltage** of a node connected to ground is defined as zero (reference node). The nodal voltage of any other node is defined as the potential difference (voltage) between that node and the ground. The algebraic relationship between the current and voltage of a component is called **branch equation**.

In a state-of-the-art implementation of SPICE such as PLECS Spice, a sophisticated algorithm is used to build the MNA equations efficiently. Nevertheless, the MNA recipe described below conveys sufficiently well what it’s algorithm achieves:

1. At every node, write the equation provided by Kirchhoff's current law (KCL).
2. Substitute currents (except for capacitors, inductors and sources) using branch equations.
3. Write the branch equations in terms of nodal voltages using Kirchhoff's voltage law (KVL).
4. Add the identities provided by the sources.
5. Add the capacitor and inductor equations.

In the resulting set of equations, the unknown variables are

- Nodal voltages (excluding the ground)
- Currents in voltage sources
- Currents in capacitors
- Currents in inductors

> **Note**
>
> In PLECS Spice, a connected SPICE electrical circuit *must* have an Electrical Ground (p. 673), because it is used as reference node for the nodal voltages. To specify a ground, simply connect the ground component to a node.

Let's apply the MNA recipe to the example RC circuit.

Step 1 Apply KCL at the nodes N1 and N2

$$\begin{aligned} i_V - i_R &= 0 \\ i_R - i_C &= 0 \end{aligned}$$

Step 2 Substitute the resistor branch equation $i_R = v_R/R$

$$\begin{aligned} i_V - \frac{1}{R} v_R &= 0 \\ \frac{1}{R} v_R - i_C &= 0 \end{aligned}$$

Step 3 By KVL, the voltage across the resistor satisfies $v_R = v_{\mathrm{N1}} - v_{\mathrm{N2}}$

$$i_V - \frac{1}{R}(v_{\mathrm{N1}} - v_{\mathrm{N2}}) = 0$$
$$\frac{1}{R}(v_{\mathrm{N1}} - v_{\mathrm{N2}}) - i_C = 0$$

Step 4 Append the source equation $v_1 = V$

$$i_V - \frac{1}{R}(v_{\mathrm{N1}} - v_{\mathrm{N2}}) = 0$$
$$\frac{1}{R}(v_{\mathrm{N1}} - v_{\mathrm{N2}}) - i_C = 0$$
$$v_{\mathrm{N1}} = V$$

Step 5 Append the capacitor equation $C\dot{v}_C = i_C$

$$i_V - \frac{1}{R}(v_{\mathrm{N1}} - v_{\mathrm{N2}}) = 0$$
$$\frac{1}{R}(v_{\mathrm{N1}} - v_{\mathrm{N2}}) - i_C = 0$$
$$v_{\mathrm{N1}} = V$$
$$\frac{d}{dt}Cv_{\mathrm{N2}} = i_C$$

General Form

By grouping all the unknowns into a single vector $\mathbf{x}$, the MNA equations can be written as the differential algebraic equation system (DAE)

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}) = \mathbf{f}(\mathbf{x})$$
$$\mathbf{0} = \mathbf{g}(\mathbf{x}, t)$$

where

$\mathbf{q}(\mathbf{x}) \equiv$ electric charges and magnetic fluxes
$\mathbf{f}(\mathbf{x}) \equiv$ capacitor currents and inductor voltages
$\mathbf{g}(\mathbf{x}, t) \equiv$ KCL equations at every node

Operating Point

The above DAE system of equations is solved by PLECS Spice during a **transient analysis**. Unlike for state-space systems, initial conditions $\mathbf{x}_0$ must satisfy the algebraic constraint (KCL)

$$\mathbf{0} = \mathbf{g}(\mathbf{x}_0, t_0).$$

The process of finding valid initial conditions is called **operating point analysis**. The resulting vector is called **operating point**. Unless initial conditions are provided, a SPICE operating point typically represents the DC steady-state of the circuit at t_0, where capacitor currents and inductor voltages are zero (all time-derivatives are zero).

By default, a transient analysis always starts with an operating point analysis. "SPICE Simulation Parameters" (p. 406) provide the option to configure or even skip the operating point analysis completely, but it is generally *not recommended* to do so. Simulating a physical system with inconsistent initial conditions can lead to unphysical solutions.

Benefits

The state-space approach adopted by PLECS is computationally very efficient for piecewise linear physical systems. By comparison with our earlier example, the ordinary differential equation (ODE) for the RC circuit is the single equation

$$\frac{d}{dt}v_C = \frac{1}{RC}V - \frac{1}{RC}v_C.$$

However, for circuits with algebraic dependencies, this ODE approach relies on an *indirect* method to resolve "Algebraic loops" (p. 36) that often fails to converge in the presence of strong nonlinearities. The DAE formulation in PLECS Spice solves algebraic constraints of physical models *directly* as part of an implicit numerical method. This direct approach is significantly more robust and is essential for handling typical SPICE semiconductor models.

Differential Algebraic Equation Solver

SDIRK and ESDIRK Methods

In PLECS Spice, we provide two 3rd-order accurate implicit Runge-Kutta methods to solve the DAE system of equations obtained from MNA. The numerical methods are augmented with dedicated circuit-tailored convergence helpers that make them perfectly suited for the simulation of SPICE models.

SDIRK (Singly Diagonally Implicit Runge-Kutta) and ESDIRK (Explicit first stage SDIRK) numerical methods solve each time step using multiple stages. At each stage the information from previous stages is combined with that of the current stage to increase the quality and accuracy of the approximation. The data from these stages is also used to evaluate a polynomial interpolant for refinement and an error estimate for stepsize control.

Because no information from previous time steps is necessary during this process, Runge-Kutta methods are called one-step methods. The key benefit of a one-step method is that it is inherently self-starting. This property is crucial for integrating hybrid schematics containing both standard PLECS and SPICE models.

Standard PLECS components may introduce physical and signal discontinuities that must be precisely located during zero-crossing detection using polynomial interpolation. After such an event, the algorithm cannot rely on memory from a different system topology. To remain performant in the presence of frequent ideal switching, the solver must be able to compute the next time step using only information from the current state, a task for which one-step methods are perfectly suited.

SDIRK
: This method was specifically designed for DAEs of index higher than 1. It provides excellent stability and should be used when (E)SDIRK shows signs of instability.

(E)SDIRK (Hybrid ESDIRK/SDIRK method)
: This is the *default* method chosen by PLECS Spice when the solver parameter (p. 135) is set to `auto` in the presence of a SPICE model. It swaps automatically between ESDIRK and SDIRK and is faster than pure SDIRK for most well-behaved systems.

Both SDIRK and ESDIRK are L-stable with L-stable embedded error estimators, ensuring robust behavior when solving stiff equations.

Mixed-Signal Hybrid Simulation

The DAE solvers can simulate complex mixed-signal hybrid systems that *combine standard PLECS and SPICE models in a single schematic*. The only restriction is that SPICE and standard PLECS physical models must be connected through sources and meters, or in other words through the control domain.

This feature enables a powerful *top-down* modeling workflow. A system initially designed and simulated entirely with fast ideal PLECS models can be refined by converting ideal switches to device-specific SPICE netlists. The controls can remain in standard PLECS.

> **Note**
>
> A connected electrical circuit is a SPICE physical model if it contains a Netlist (p. 845) or Netlist File Reference component (p. 847). All the standard PLECS components of a SPICE physical model must be compatible (p. 409) with SPICE.

> **Warning**
>
> Electrical grounds in PLECS electrical circuits are not connected to electrical grounds in SPICE electrical circuits.

Netlists

Format

A SPICE netlist is a text-based description of an electric circuit. The format is widely adopted by manufacturers to provide device-level models of specific semiconductor devices. While there are slight differences between the dialects of popular SPICE solvers, they all rely on comparable syntaxes. This allows importing netlists from varied sources into the PLECS Spice Netlist component (p. 845).

The syntax supported by PLECS Spice is summarized in “Netlist Cheat Sheets” (p. 411) and covered in detail in “Netlist Syntax Reference” (p. 429).

Example Netlist

The netlist description of the above “Example RC Circuit” (p. 389) is simple.

```
.SUBCKT RC_Circuit N2

R N1 N2 1
V N1 0 1
C N2 0 1e-4
.IC V(N2)=0

.ENDS
```

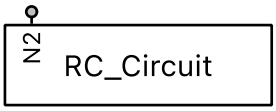
It defines

- a $1\,\Omega$ resistor identified as `R` connected to nodes `N1` and `N2`
- a $1\,\mathrm{V}$ voltage source identified as `V` connected to the node `N1` and ground
- a $100\,\mu\mathrm{F}$ capacitor identified as `C` connected to node `N2` and ground
- an initial nodal voltage of zero at node `N2`

> **Note**
>
> When a node in a netlist is named `0`, it always refers to ground.

Remark. By setting the initial nodal voltage at `N2` to zero with `.IC V(N2)=0`, we match the standard PLECS state-space formulation behavior that sets the initial voltage across the capacitor to zero. The nodal voltage at N2 would otherwise be set to one by the operating point analysis (p. 392), because the capacitor current is initialized to zero in the operating point (its DC steady-state). Accordingly, constant nodal voltages would be observed during transient simulation.

The circuit description starts with `.SUBCKT RC_Circuit N2` and is completed by `.ENDS`. These instructions specify that the subcircuit is named `RC_Circuit` and that `N2` is an exposed node. Both the name and the exposed node are visually reflected in the Netlist component (p. 845) of the schematic:



`N2` can be electrically connected to other components of the schematic. For example, connecting a voltmeter to `N2` and an Electrical Ground (p. 673) measures the nodal voltage at `N2`.

> **Note**
>
> More than one node can be exposed. For example, `.SUBCKT RC_Circuit N1 N2` creates two terminals labeled `N1` and `N2` for the Netlist component (p. 845) on the PLECS schematic. These terminals correspond to the exposed nodes `N1` and `N2`. Their labels can be changed independently of the netlist via the "Terminals Tab" (p. 401).

Compact Models

In addition to sources and passive elements, SPICE solvers provide a set of MNA compatible models for standard semiconductor elements such as mosfets, diodes and bipolar junction transistors. These models, also referred to as **compact models**, are designed to capture the fundamental electrical behavior of nonlinear devices while also retaining reasonably low complexity. They are made out of a mixture of physics-based modeling and empirical corrections.

To achieve optimal simulation performance, compact models are tightly integrated in the implementation of the solver. Customization of the model equations by the user is not possible, but every compact model defines a set of parameters that can be tuned to match the electrical response of the model to the response of the physical device. This restriction permits tailored optimization and the use of convergence helpers to increase robustness of the simulation. In PLECS Spice, classical compact models were improved to guarantee continuity of key physical quantities.

```
M drain gate source source mosfetModel
.MODEL mosfetModel NMOS VTO=3.5 KP=100 ; more parameters can be specified␣
↪here
```

Using PLECS Spice

Netlist Component

Fundamentals

To include a SPICE model in a schematic, we use the Netlist component (p. 845). It enables both writing circuit descriptions manually and importing a netlist (p. 400) from a file.

For example, by manually copying the netlist (p. 394) corresponding to the "Example RC Circuit" (p. 389) in the dialog of a netlist component, we obtain the schematic depicted in Fig. 14.1.

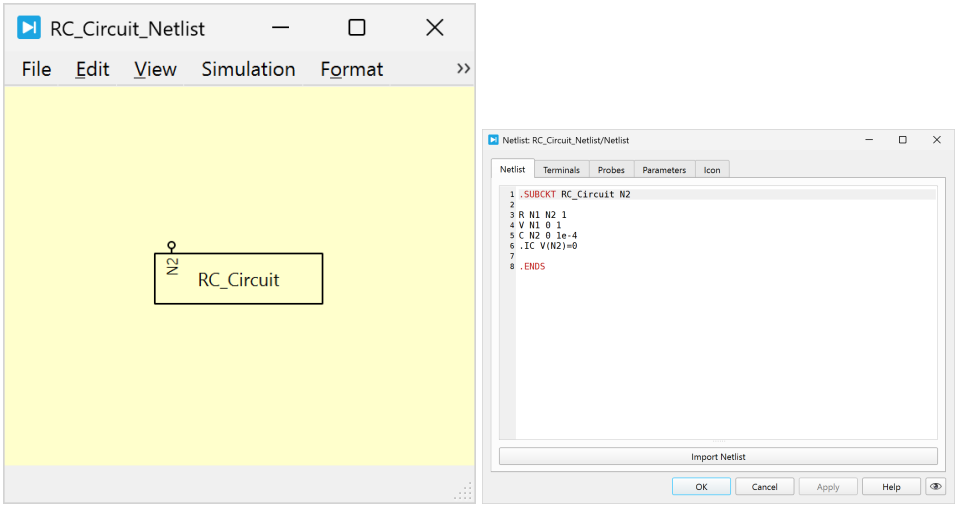


Fig. 14.1: Netlist description of an RC circuit

The first subcircuit located at the top of a netlist (the one defined by the first `.SUBCKT` and its corresponding `.ENDS`) is called the **main subcircuit**. The name of the main subcircuit is shown on the Netlist component (p. 845) in the schematic. Similar to a standard PLECS subsystem (p. 977), a Netlist component (p. 845) can be resized by dragging one of its corners.

Nodes listed after the main subcircuit name in the `.SUBCKT` statement are exposed as terminals on the PLECS schematic. They can be used to connect SPICE models to other compatible components (p. 409). To move a terminal, hold **Shift** and drag it along the edges of the component.

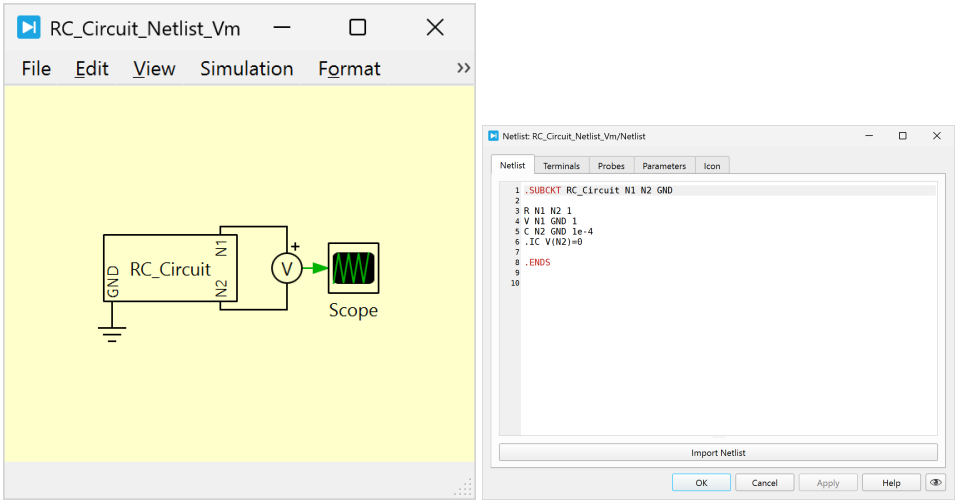


Fig. 14.2: Netlist with exposed terminals and ground connection

> **Note**
>
> Some netlists do not reference ground (node `0`) explicitly, but expose instead a terminal (often called `GND`) that must be grounded in the schematic using an Electrical Ground (p. 673).

> **Warning**
>
> Electrical Grounds (p. 673) in PLECS electrical circuits are not connected to electrical grounds in SPICE electrical circuits.

PLECS Spice solvers are capable of simulating mixed-signal hybrid (p. 394) schematics that contain both standard PLECS and SPICE electrical circuits.

As an example, let us study the difference between the reverse recovery of a detailed device-level SPICE diode model and that of a parametrized Diode with Reverse Recovery (p. 646) from the standard PLECS library.

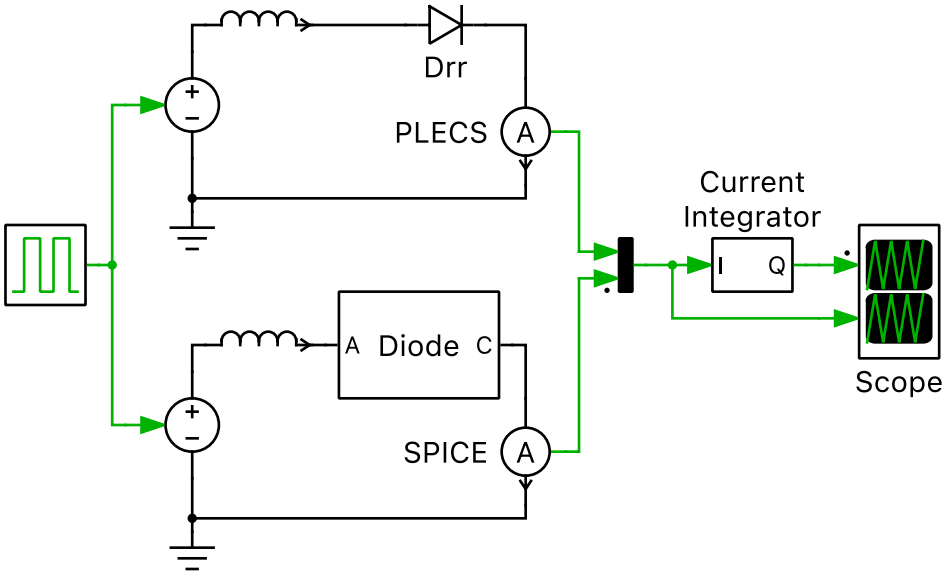


Fig. 14.3: Comparison of a SPICE diode model with the standard PLECS diode with reverse recovery. The top circuit contains an ideal diode. The bottom circuit contains a SPICE model. Both circuits are driven and measured by the same standard PLECS controls.

SPICE Highlighting

After running a PLECS Spice simulation, the Ⓢ icon appears in the bottom right corner of the schematic window. Pressing it will open the diagnostics window and highlight the electrical circuits that are solved using the "Modified Nodal Analysis" (p. 389) formulation. Parts of the schematics that are not highlighted are solved using the standard PLECS state-space formulation (p. 34).

The Netlist Dialog

The Netlist (p. 845) component dialog provides five tabs for configuring a netlist, customizing terminal labels, defining probe signals, setting parameters and adding an icon.

Netlist Tab

A SPICE netlist description is included in a Netlist (p. 845) component using the *Netlist* tab. It can be written/edited directly in the provided text dialog or imported form a file using using the **Import Netlist** button.

Writing Netlists

Netlists must follow the ".SUBCKT Statement" (p. 473) syntax introduced in "Fundamentals" (p. 397). The "Netlist Cheat Sheets" (p. 411) provide a quick overview of the format.

Importing Netlists

Click on the import option of the netlist component dialog to import a netlist from a file. PLECS Spice will parse the selected file and provide a drop-down menu with all available subcircuits (starting with `.SUBCKT`), models (starting with `.MODEL`) as well as the top level circuit (terminated with `.END`).

The importer automatically populates the text field based on your selection:

- If you choose a subcircuit: It is copied to the top of the text field (as the main subcircuit), and all its dependencies are automatically appended.
- If you choose a model: It is automatically wrapped within a new main subcircuit.
- If you choose the top level circuit: It is wrapped into a subcircuit. All dependencies are automatically appended. You will have to manually list the nodes that you want to expose.

The preview window is used to fix errors occurring during the import process.

> **Note**
>
> Importing a netlist will overwrite previously written/imported content.

Note

When the dependency detection encounters errors or parametric subcircuit names, you are prompted to choose between two options:

- Append all detected dependencies: Includes only dependencies up to the error or with the current parameterization.
- Append entire file: Includes all content from the file, guaranteeing no dependencies are missed. This results in larger file size and slower simulation initialization.

Diagnostics

After parsing the netlist, PLECS Spice displays diagnostics for syntax violations (p. 429) and additional debugging information, as shown in Figure Fig. 14.8 on p. 420.

Diagnostic messages appear in a table below the netlist editor. Click a message to navigate to the corresponding line in the text field. For messages referring to multiple lines, clicking the message repeatedly cycles through the relevant text snippets. This feature is particularly useful to *debug* a SPICE model or adapt the syntax of a netlist to PLECS Spice.

Messages are categorized as errors, warnings, or informational notes:

Errors must be resolved before PLECS Spice can simulate.

Warnings indicate that PLECS Spice may automatically adapt to an unsupported netlist statement (for example,by ignoring an unrecognized parameter).

Information messages provide additional context without requiring action.

Note

When importing a netlist, the parser is unaware of parameter overwrites. In rare cases, this can lead to a diagnostic that is inconsistent with the one obtained after parameters are evaluated or at the start of a simulation.

Terminals Tab

The *Terminals* tab allows renaming terminals of the Netlist component (p. 845) corresponding to exposed nodes without modifying the netlist itself. This is useful to assign standardized or meaningful names to schematic terminals.

Examples:

- `1`, `2`, `3` → `drain`, `gate`, `source`
- `D`, `G`, `S` → `drain`, `gate`, `source`
- `1`, `2`, `3` → `Vin`, `Vout`, `GND`

Probes Tab

The *Probes* tab enables you to define probe signals for the Netlist component (p. 845). These probe signals can directly be used in a Probe (p. 116) or in a Masked Configurable Subsystem (p. 104), making it easier to switch between configurations and keeping probes valid.

We currently support five types of probes:

- Terminal voltage probes measure the potential difference between two terminals of the Netlist component (p. 845).
- Terminal current probes measure the current flowing **in** through the specified terminal of the Netlist component (p. 845).
- Nodal voltage probes measure the nodal voltage at an internal node of the netlist.
- Element voltage probes measure the potential differences across two-node elements of a netlist.
- Element current probes measure the current in a two-node element in the netlist.

The terminal-based probes offer a selection dialog based on the terminal names in the “Terminals Tab”. The internal probes selection tool offers a tree view of the nodal voltages and the two-terminal elements that exist in the parsed netlist. This tree structure represents the hierarchy how subcircuits call each other.

Parameters Tab

The *Parameters* tab allows you to overwrite the default values of optional parameters defined in the main subcircuit (specified after `PARAMS:` in the subcircuit declaration), e.g.

```
.SUBCKT netlist n1 n2 PARAMS: a=1 b=2 c=3
```

Overwrite values are parsed using the Octave engine. They can reference variables from your schematic’s Model Initialization Commands (p. 143). This enables parameterization of device models and parameter sweeps.

The *Parameters Tab* also supports vectorized inputs (e.g., `[1 2 3]`), which vectorizes the Netlist component (p. 845) in the same way as standard PLECS components (see "Vectorization" (p. 77)).

Icon Tab

You can also define a custom icon for your netlist component via the *Icon* tab. As an example, let's add an icon for the diode in Fig. 14.3 on p. 399.

The diode icon is constructed from a triangle and three individual straight line elements and is scripted using the Lua Language (p. 107). Furthermore, it is possible to display an arbitrary netlist name via the command: `Icon:text("My netlist name")`. A detailed description of the input dialog and the available *Lua* commands is available here (p. 84).

Netlist File Reference Component

The Netlist File Reference component (p. 847) is a variant of the Netlist component (p. 845) that does not store the netlist in an editor. Instead, the netlist is imported at runtime from a file.

Advantages of the Netlist File Reference component (p. 847)

- The file and imported model / subcircuit can be specified and changed using variables.
- Modifications to the referenced file affect all dependent simulations.
- Simplifies "Building libraries for semiconductor device families" (p. 405).
- Avoids storing netlists as hard copies in the .plecs file.

Advantages of the Netlist component (p. 845)

- Diagnostics (p. 401) are available in the netlist editor, including warnings and infos.
- Node names and parameter names in the "Terminals Tab" (p. 401) and "Parameters Tab" are automatically updated upon editing the main subcircuit.
- Internal probes are supported.
- Modifications in the netlist editor are local and do not affect other simulations.

- The .plecs file is self-contained and can be shared without having to send additional files.

In most cases, importing a netlist (p. 400) into the Netlist component (p. 845) is the recommended workflow.

Best Practices

Use Configurable Subsystems

To add device-level detail using SPICE models without abandoning your existing PLECS schematic, use Configurable Subsystems (p. 620) to switch between PLECS and SPICE implementations of the same circuit.

Consider a buck converter with controls in the digital domain.

To add SPICE device models, we wrap the power stage components in a configurable subsystem with two configurations:

1 **PLECS configuration**: Uses ideal PLECS components (MOSFET, diode)

2 **SPICE configuration**: Uses netlist components with detailed device models

Using this configurable subsystem approach, we add the netlist components alongside the original PLECS components.

The active configuration can be controlled directly in the schematic or through a variable defined in the initialization commands or in a script, making it easy to compare ideal and detailed simulations.

This approach is particularly powerful because:

- The digital control section remains unchanged between configurations
- Switching between ideal and detailed simulations is quick and easy
- By holding traces, you can plot both simulations in a single scope
- There is no need to maintain separate model files or rebuild circuits

Use Masked Subsystems

When using configurable subsystems, creating a custom mask (p. 84) to visually represent the devices contained in the subsystem, makes the schematic more intuitive.

In the mask editor, probe signals defined in the netlist can be accessed and used to define “Mask Probe Signals” (p. 104). Probing the subsystem returns the MOSFET’s channel voltage from either the PLECS component or SPICE netlist, depending on the current configuration.

Use Model References

When reusing a SPICE device in a circuit, use the Model Reference component (p. 824) with Netlist components (p. 845) to automatically propagate netlist, parameter, or probe changes to all instances. Wrapping the netlist into a subsystem allows mask parameters to be set independently for each model reference to the subsystem.

Use Custom Libraries

For frequently reused netlists across circuits, create custom "Libraries" (p. 72) to maintain a collection of SPICE device models for easy project integration.

To create a custom library element:

- Import the subcircuit of interest into a Netlist (p. 845) component.
- Change the terminal labels, define probes signals and define an icon.
- Save this component in your custom library.

Building libraries for semiconductor device families

Semiconductor manufacturers often bundle the SPICE models of several devices into a single file. Such a file typically contains several top-level SPICE subcircuits along with helper circuits.

We recommend the following approach to generate a custom library for a family of semiconductor devices:

- Use the Netlist File Reference (p. 847) component and use a variable for the input field *Name*.
- Set up the terminals and define probes.
- Wrap this component into a masked subcircuit (p. 84) with icon.
- Set up the mask dialog by defining a new string-valued combo box (p. 98) parameter with the aforementioned variable name.
- Store all SPICE subcircuit names of interest as combobox options.
- Save this subcircuit in a PLECS library next to the netlist file.

Example Model

- See the example model **"Netlist File Reference for Device Families"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

SPICE Simulation Parameters

With a PLECS Spice license, additional settings are accessible in the "Simulation Parameters" (p. 135).

New Solver Options

On the *Solver* page of the "Simulation Parameters" (p. 135), new SDIRK and ESDIRK solvers (p. 393) are available for "Mixed-Signal Hybrid Simulation" (p. 394) of hybrid PLECS and SPICE models; these appear under the *Solver* setting when *Solver Type* is set to *Variable-step*. When the *Solver* is set to *auto* and any circuit part requires the modified nodal analysis formulation (p. 389), (E)SDIRK is automatically selected as default solver.

As a general rule of thumb, (E)SDIRK is usually faster, but SDIRK is typically more stable.

> **Attention**
>
> Selecting either of these new solvers manually does **not** force a circuit to be solved the SPICE way using "Modified Nodal Analysis" (p. 389). This is purely dependent on the components.

SPICE Simulation Parameters

On the new *SPICE* page in the "Simulation Parameters" (p. 135), you can change the following:

Transient Analysis Options

Transient regularization

These settings allow to regularize the simulation, potentially increasing stability and reducing oscillations by artificially damping the equations. Note that these values only apply to electrical circuits that are solved using the Modified Nodal Analysis formulation.

> **Attention**
>
> Setting these parameters alters the circuit that is simulated and it will behave slightly different from the circuit in the schematic.

You can select one of the presets `Off`, `Low` and `High`, or define your own parameters by selecting `Custom`. The following regularization parameters are hidden when `Off` is selected.

Minimum conductance to ground
A small conductance added from every node to ground, preventing ill-conditioned matrices when solving the equations. Typical values range from `1e-12` to `1e-9`.

Parallel conductance to capacitance ratio
Adds a parallel conductance across all capacitors. The conductance depends on the capacitance:

$$G_{\text{par}} = \text{ratio} \cdot C \quad \text{or} \quad R_{\text{par}} = \frac{1}{\text{ratio} \cdot C}$$

Typical values range from `1e-14` to `1e-9`.

Inductor series resistance
A small resistance is inserted in series with every inductor. Common values range from `1e-6` to `1e-3`.

Parallel resistance to inductance ratio
Adds a parallel resistor across all inductors. The resistance depends on the inductance:

$$R_{\text{par}} = L \cdot \text{ratio}$$

Typical values range from `1e14` to `1e9`. No resistor is placed for a value of `0`.

The resistances are applied to inductors and capacitors as seen in Fig. 14.18 on p. 428.

Max. number of iterations
Limits the number of Newton iteration per implicit stage. When the solver does not converge within the iteration limit, the current step size is reduced and a smaller one is attempted.

Max. number of threads
This sets the number of threads the solver may use in the current simulation. An upper limit is given by the *Thread limit* in the PLECS preferences (p. 60). At most, the number of physical CPU cores on your machine are used.

> **Note**
>
> The options specified in "Variable-Step Solver Options" (p. 136) also **do affect** the transient analysis during SPICE simulations.

Operating Point Analysis Options

Max. number of iterations
: This limits the number of Newton iterations during the operating point analysis (p. 392).

Tolerances
: The relative and absolute tolerances specify the acceptable local integration error during the operating point analysis (p. 392). Additionally we offer the option to set a tolerance for nodal voltages computed during the analysis. Note that these tolerances by default have different scales.

Bypass operating point
: When set to *on*, the operating point analysis (p. 392) is skipped and the simulation starts from zero. Initial conditions are ignored. This setting is **not** recommended.

Initial conditions
: When set to *off*, user-provided initial conditions are ignored. The solver performs a standard SPICE operating point analysis (p. 392).

  When set to *on*, the operating point analysis (p. 392) attempts to find a consistent state that also satisfies the initial conditions given by "PLECS Components Compatible with SPICE" and the .IC statements in netlists.

Parameter Settings

.temp
: The default temperature used for SPICE circuits. See "Temperature (.TEMP)" (p. 415) for details.

Diagnostics

Several settings are available for configuring how diagnostics are displayed.

Netlist parser details
: When set to *show*, warnings and info messages from netlists are listed with other information in the diagnostics window of PLECS, accessible from the bottom right corner of a schematic. Regardless the setting, this

information is always accessible in the dialog of the Netlist component (p. 845).

Invalid netlist function evaluation
Netlists may contain user-defined functions. In some ill composed netlists, these functions need to be evaluated outside of their domain, e.g. the logarithm might be evaluated using a negative number. When set to *ignore*, we extend the domain of all functions to $\mathbb{R}$. See "Mathematical Functions" (p. 430) for details.

Incompatible initial condition
Both "PLECS Components Compatible with SPICE" and .IC statements (p. 470) may specify initial conditions. In the beginning of the simulation we analyze whether satisfying these initial conditions is feasible. A capacitor parallel to a voltage source, for example, can not specify an initial voltage across. If there are contradicting initial conditions, we automatically ignore some of them to fix the system. All of those are listed in the schematics diagnostics when not set to *ignore*.

PLECS Components Compatible with SPICE

List of Compatible Electrical Components

The following standard PLECS components are natively supported in SPICE circuits (see "Mixed-Signal Hybrid Simulation" (p. 394)):

- **Connectivity**
  - Electrical Port (p. 676)
  - Electrical Label (p. 674)
  - Wire Multiplexer (p. 1128)
  - Wire Selector (p. 1129)
  - Electrical Ground (p. 673)
- **Sources**
  - Voltage Source (Controlled) (p. 1119)
  - Voltage Source DC (p. 1122)
  - Voltage Source AC (p. 1120)
  - Voltage Source AC (3-Phase) (p. 1121)
  - Current Source (Controlled) (p. 632)
  - Current Source DC (p. 634)

    - Current Source AC (p. 633)
- **Meters**
    - Ammeter (p. 558)
    - Voltmeter (p. 1123)
    - Meter (3-Phase) (p. 819)
- **Passive Components**
    - Resistor (p. 905)
    - Inductor (p. 779)
    - Capacitor (p. 597)
    - Mutual Inductor (p. 843)
    - Variable Resistor (p. 1109)
    - Variable Inductor (p. 1099)
    - Variable Capacitor (p. 1092)
    - Variable Resistor with Variable Inductor (p. 1114)
    - Variable Resistor with Constant Inductor (p. 1111)
    - Variable Resistor with Variable Capacitor (p. 1112)
    - Variable Resistor with Constant Capacitor (p. 1110)
    - Electrical Algebraic Component (p. 671)
- **Manual Switches**
    - Manual Switch (p. 802)
    - Manual Double Switch (p. 800)
    - Manual Triple Switch (p. 803)
- **Transformer**
    - Ideal Transformer (p. 744)
    - Linear Transformer (2 Windings) (p. 789)
    - Linear Transformer (3 Windings) (p. 791)
    - Mutual Inductance (2 Windings) (p. 839)
    - Mutual Inductance (3 Windings) (p. 841)
- **Electronics**
    - Op-Amp (p. 856)

Note

The Inductor (p. 779) and Capacitor component (p. 597) provide fields for specifying initial conditions. For SPICE simulations, explicit initial condition specification is optional. You may leave these fields empty or set them to `NaN`. The operating point analysis (p. 392) will resolve unspecified initial conditions according to standard SPICE behavior for Capacitor (p. 436) and Inductor statements (p. 449) without an explicit `IC` parameter.

Additional Limitations

- All the connections of wire multiplexers (p. 1128) and wire selectors (p. 1129) share the same mathematical formulation. Thus, it is not possible to create a bus bundling wires from standard PLECS and SPICE circuits.
- During a mixed-signal simulation, live changes to parameters and manual switch states are not permitted. Attempting to do so, e.g., via a script (p. 314), the RPC-interface (p. 323), or the user interface, raises an error.

Example Model

- See the example model **"Creating simple diodes in PLECS Spice"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Netlist Cheat Sheets

This section provides examples explaining how to use netlists in PLECS Spice.

Listing 14.1: Comments and Parameters

```
* This is a comment.

* Subcircuit definitions start with .SUBCKT and end with .ENDS.
* The following subcircuit is called "main" and has nodes "n1" and "n2".
.SUBCKT main n1 n2; The semicolon starts an inline comment.

.PARAM a=100; This defines the parameter "a".
.PARAM c=1e-6 r=1e3; This defines parameters "c" and "r".
.PARAM l={a*c}; Expressions in curly braces are evaluated.

R1 n1 0 1e3; This is a 1 kiloohm resistor between nodes n1 and ground.
```

(continues on next page)

(continued from previous page)

```
* The node "0" always denotes ground.
R2 n2 0 {r}; The resistance of "R2" is given by the parameter "r".
C1 n1 n3 {c}; This is a capacitor. Node "n3" is internal and not exposed on␣
↪the schematic.
L1 n3 n1 {l}; This is an inductor.
.ENDS
```

Listing 14.2: Compact Models

```
* Subcircuits can have parameters.
* The main subcircuit parameters can be changed in the PLECS UI of the␣
↪netlist component.
.SUBCKT main d g s a c PARAMS: is=1e-9 vto=3.5

* Define compact models using .MODEL statements.
.MODEL dio D; This model called "dio" loads the default diode model.
.MODEL mos1 NMOS; This model called "mos1" loads the default NMOS model.
.MODEL mos2 NMOS IS={is} VTO={vto} LEVEL=3; This model sets additional␣
↪parameters.

* Add circuit elements using the previously defined models.
D1 a c dio; This creates a diode based on the model "dio".
D2 a c dio; We can create several elements with the same model "dio".
M1 d g s s mos1; The last MOSFET node - bulk - is typically connected to␣
↪source.

.ENDS
```

Listing 14.3: Sources and Functions

```
.SUBCKT main n1 n2 n3 n4 n5 n6 n7 n8

.PARAM a=1 b=2
.FUNC fun(x, y) = {cos(x) * y + b}; This defines a function with variables␣
↪x and y.

V1 n1 0 1; This is a 1 Volt DC voltage source.
I1 n2 0 1; This is a 1 Ampere current source.

* Sources with user-defined current or voltage use the letter B.
B1 n3 0 V={sin(TIME)}; The voltage of this source is given by sin(TIME),␣
↪where TIME is the current simulation time.
B2 n4 0 V={fun(TIME, a)}; This evaluates the previously defined function as
↪"cos(TIME)*1+2".

B3 n5 0 I={V(n8)}; The current of this source is given by the nodal voltage␣
```

(continues on next page)

(continued from previous page)

```
↪V(n8).
B4 n6 0 I={I(V1)}; The current of this source is given by the current of␣
↪voltage source "V1".

B5 n7 0 V={if(V(n8)>0, tanh(V(n8)-1)/2, 0)}; When V(n8)>0 then tanh(V(n8)-
↪1)/2 else 0.
* To mitigate convergence issues the equation above is designed to be␣
↪continuous and differentiable.

.ENDS
```

Listing 14.4: Calling Subcircuits

```
* This is the main subcircuit.
.SUBCKT main n1 n2 n3 n4 n5

.PARAM a=0
.PARAM c=3

X1 n1 n2 helper1; This creates an instance of the "helper1" subcircuit␣
↪between nodes "n1", "n2".
X2 n3 0 helper1; Several instances can be created. The ground node "0" can␣
↪be used here.
X3 n4 n5 helper2 PARAMS: b=4; Specifying parameters overwrites the defaults␣
↪defined in the subcircuit.

* The three lines above create a total of six resistors:
* 1 Ohm between n1 n2 (R1 in X1)
* 1 Ohm between n3 and ground (R1 in X2)
* 1 Ohm between n4 and n5 (R1 in X3)
* 4 Ohm between n4 and n5 (R2 in X3) (X3 overwrites the subcircuit␣
↪parameter b)
* 3 Ohm between n4 and n5 (R3 in X3)
* 10 Ohm between n4 and ground (R4 in X3)

.ENDS

* This is a helper subcircuit.
.SUBCKT helper1 local1 local2
R1 local1 local2 1
* Nodes from the main subcircuit such as "n1" cannot be accessed here.
.ENDS

* This is parameterizable helper subcircuit.
.SUBCKT helper2 n1 n2 PARAMS: a=1 b=2
* Nodes "n1" and "n2" are local and not connected to the nodes "n1", "n2"␣
↪of the main subcircuit.
* Parameters, functions, and models from the calling subcircuit are also␣
```

(continues on next page)

(continued from previous page)

```
↪accessible here
R1 n1 n2 {a}
R2 n1 n2 {b}
R3 n1 n2 {c}; Since c is not defined here, c=3 from the subcircuit creating␣
↪this instance is used.
R4 n1 0 10; The ground node "0" is the same globally.
.ENDS
```

For technical details on using netlists, we refer to “Netlist Syntax Reference” (p. 429).

Differences to Text-Based SPICE Simulators

PLECS Spice offers “Mixed-Signal Hybrid Simulation” (p. 394), coupling PLECS and SPICE circuits via a single intuitive user interface. To allow for this tight integration, we refrain from creating an intermediate netlist file before parsing. Such files often contain additional solver settings, plotting and measurement instructions, or sweep parameters. In PLECS Spice, netlists are solely used for describing subcircuit models in the Netlist component (p. 845).

In the following, we explain how to use the PLECS UI instead of text-based commands.

Parameter Sweeps (.STEP)

PLECS Spice offers two different approaches to carry out parameter sweeps:

- “Simulation Scripts” (p. 313) to simulate a circuit sequentially for several parameter values
- “Vectorization” (p. 77) to simulate several parameter values simultaneously.

By exposing netlist parameters in the “Parameters Tab” (p. 402), you can sweep over them like over any other parameter in standard PLECS.

> **Warning**
>
> Using “Vectorization” (p. 77) for a sweep parameter multiplies the system size by the number of sweep values and can severely affect simulation speed. Therefore, it is recommended to use the scripting functionality for carrying out parameter sweeps in large simulations.

Temperature (.TEMP)

Compact models in SPICE behave differently depending on the temperature of the circuit. In PLECS Spice, you can set this temperature globally in the "Parameter Settings" (p. 408). To sweep temperature, you can define a variable for `.temp` and vary that variable in a Simulation Script (p. 313).

Example Model

- See the example model **"Temperature Sweep in PLECS Spice"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Note

If you define a parameter called `temp` in a subcircuit, it will overwrite the global setting for all user-defined functions in this subcircuit and all referenced subcircuits. However, compact models will still read the global temperature setting, unless you explicitly pass `TEMP={temp}` as instance parameter when creating the semiconductor element.

Measuring Quantities of Interest (.MEAS)

In PLECS Spice, you can directly probe voltages and currents using Voltmeters (p. 1123), Ammeters (p. 558), and via the "Probes Tab" (p. 402). You can use the PLECS Scope (p. 942) to analyze these measurements or process them via the control domain in standard PLECS.

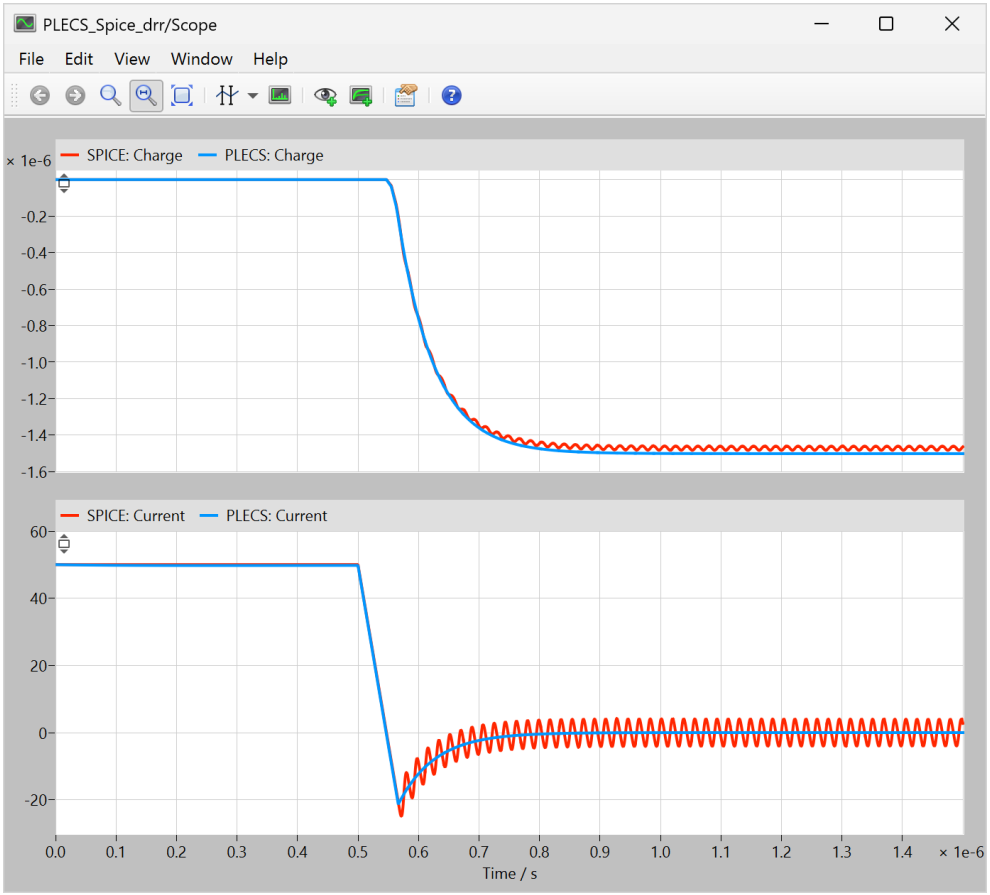


Fig. 14.4: Scope measurements from the schematic shown in Fig. 14.3 on p. 399.

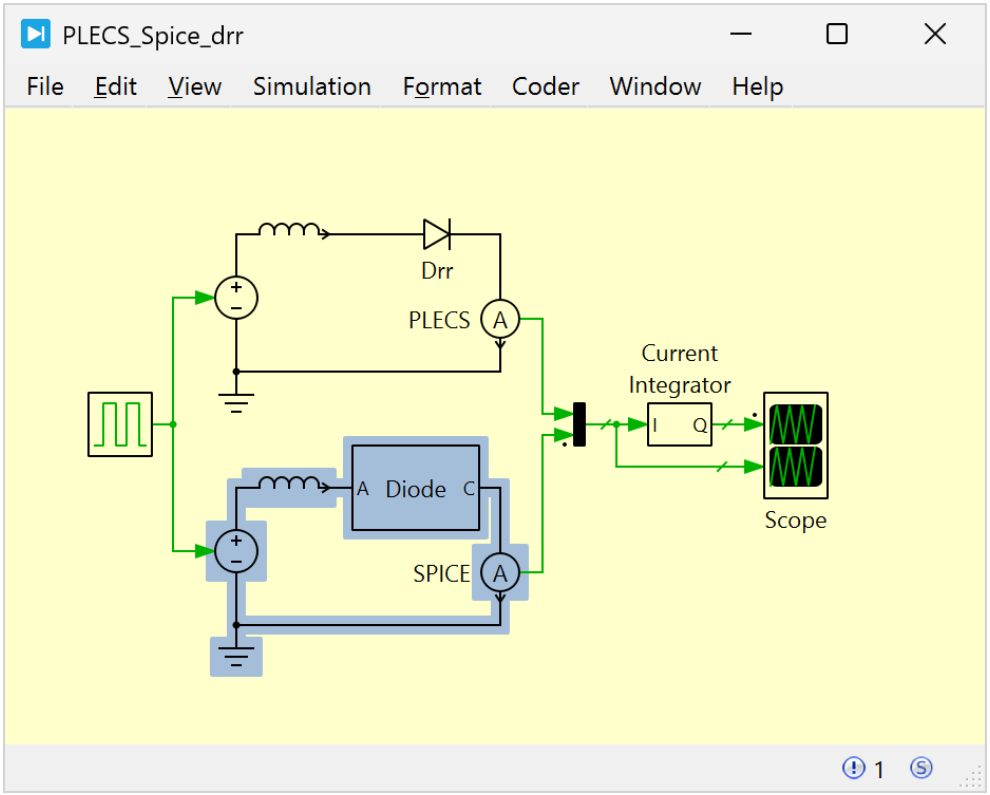


Fig. 14.5: Pressing the SPICE icon highlights physical models that are simulated using MNA

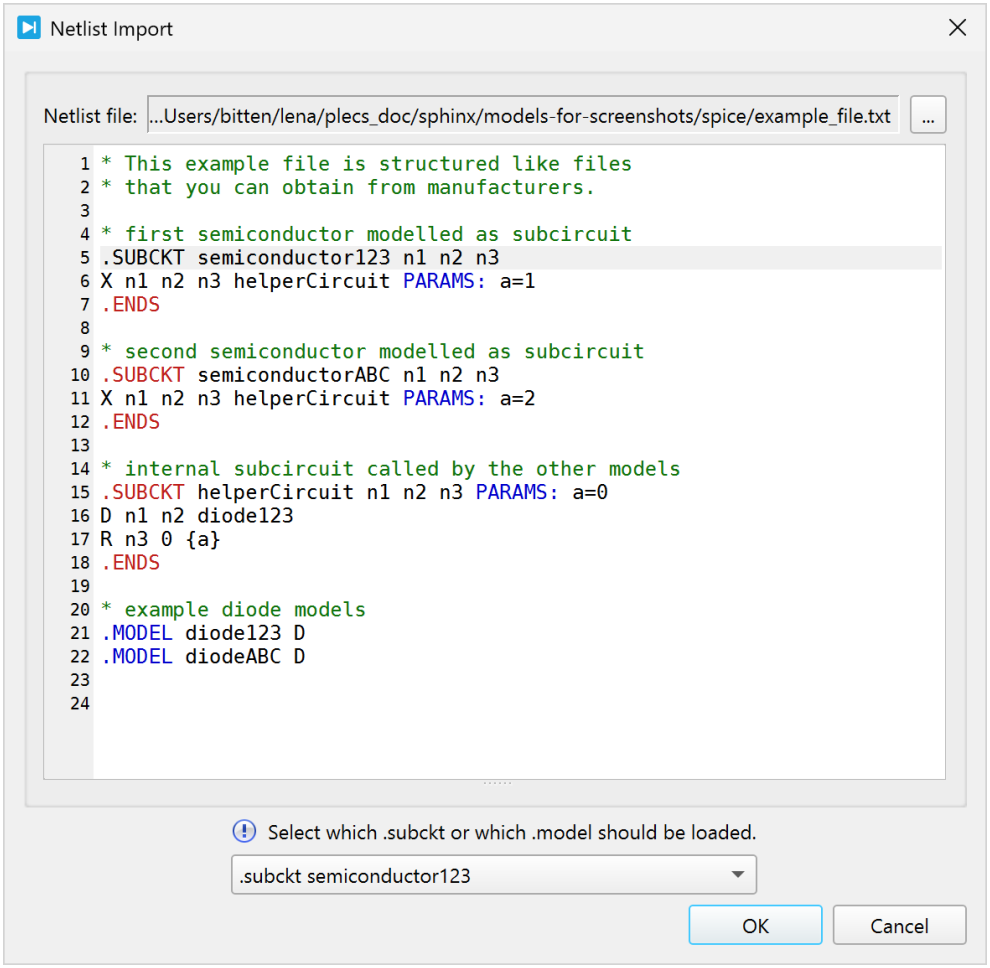


Fig. 14.6: A netlist import dialog with an example netlist.

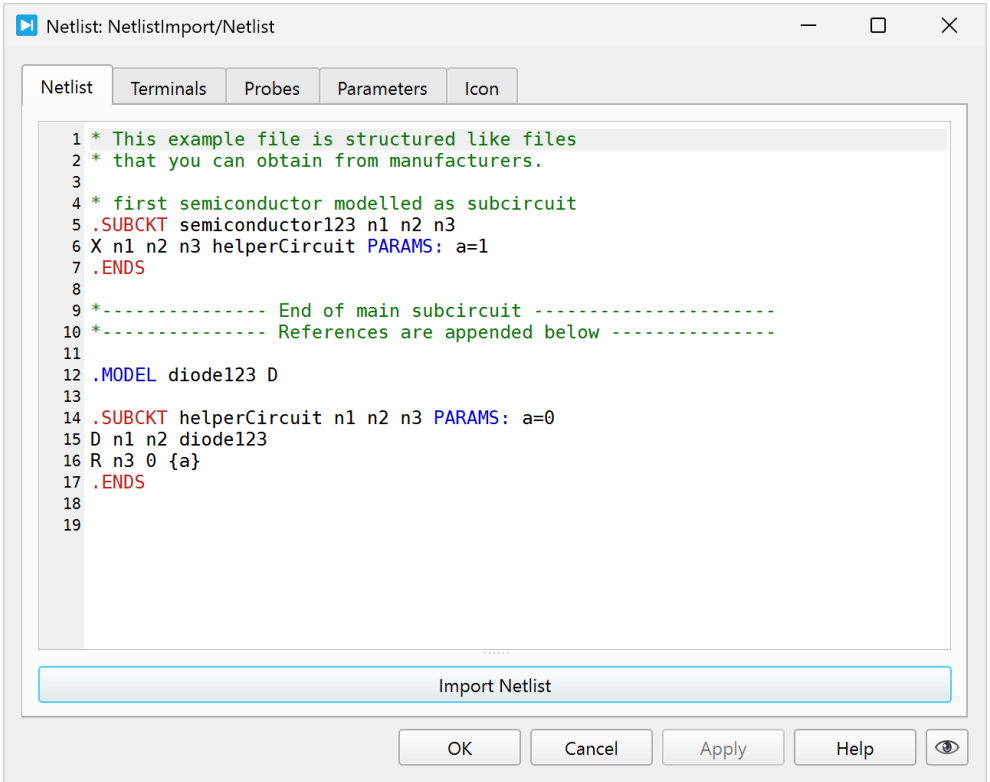


Fig. 14.7: After importing *semiconductor123* in Fig. 14.6.

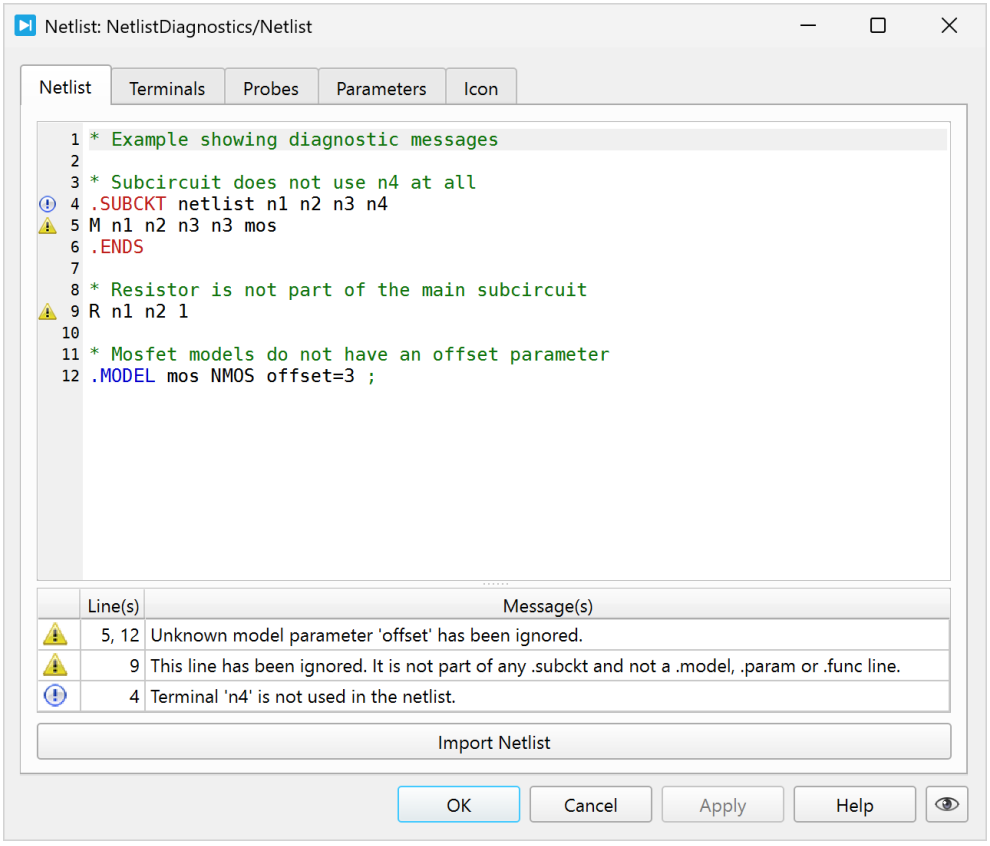


Fig. 14.8: Netlist showing several diagnostics.

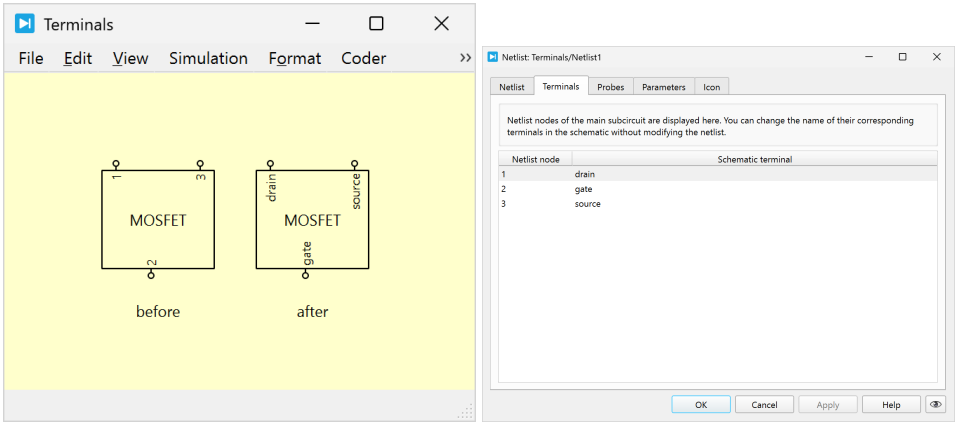


Fig. 14.9: Renaming netlist terminals.

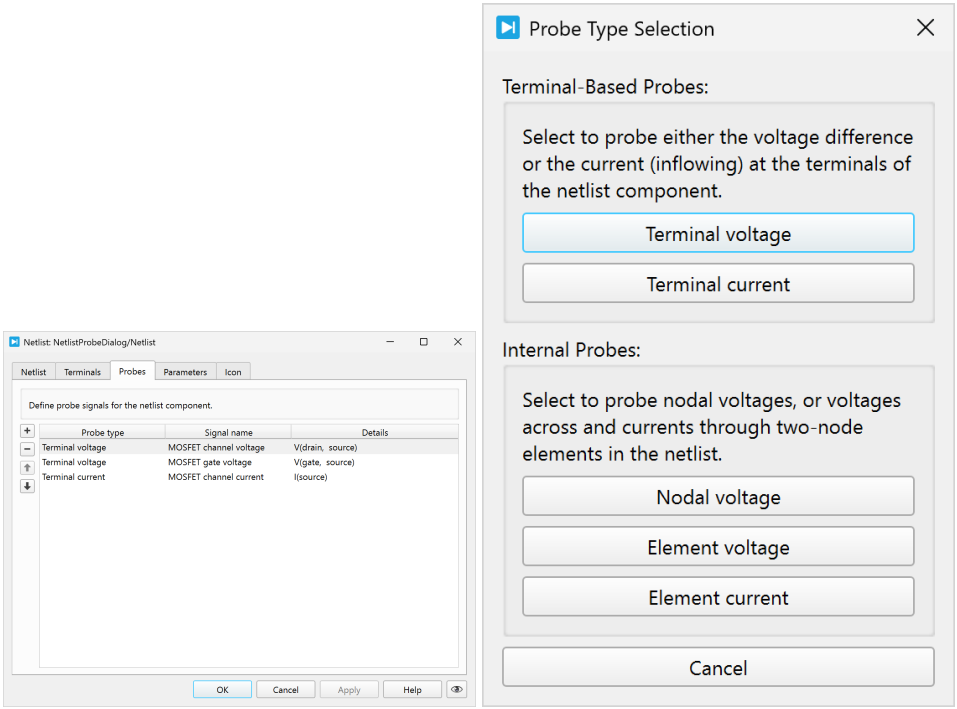
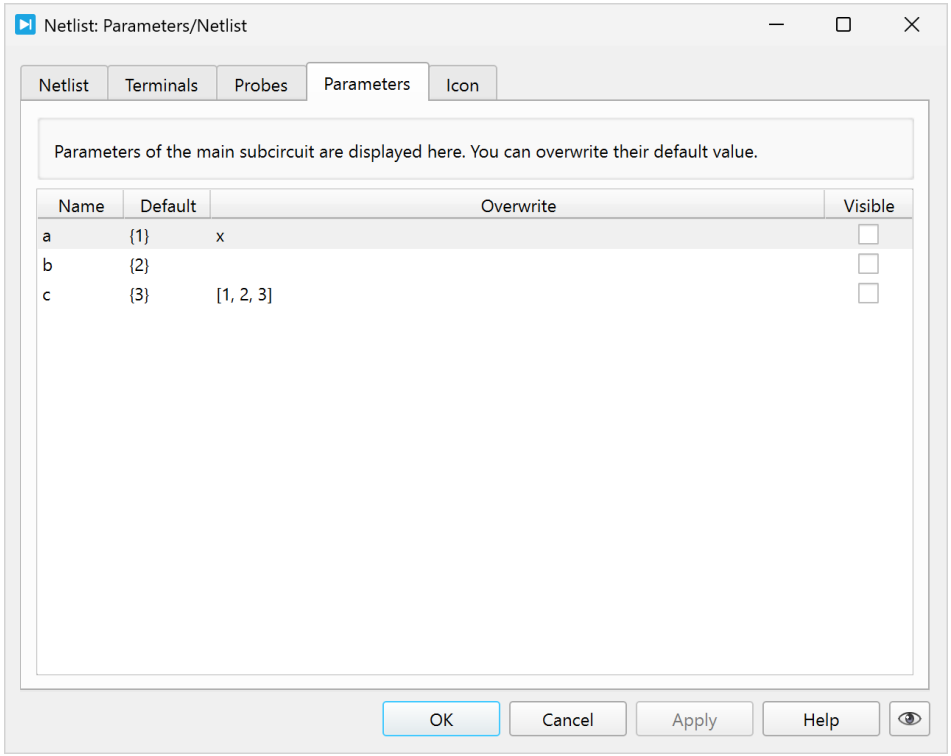
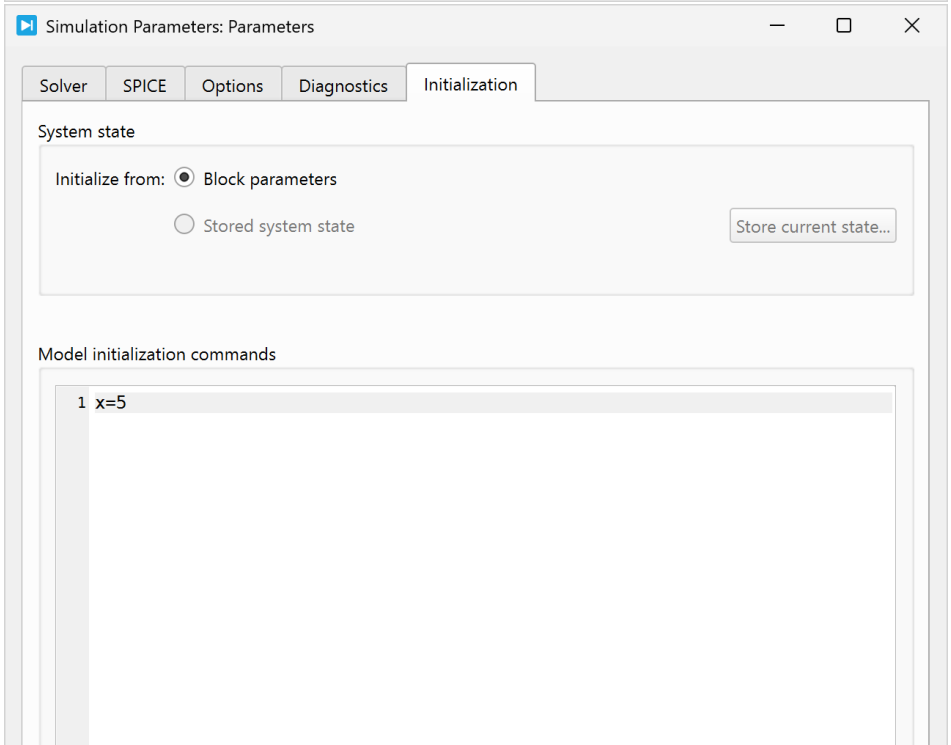


Fig. 14.10: Probing netlist voltages and currents.


Netlist: Parameters/Netlist
Netlist
Terminals
Probes
Parameters
Icon
Parameters of the main subcircuit are displayed here. You can overwrite their default value.
Name
Default
Overwrite
Visible
a
{1}
x
b
{2}
c
{3}
[1, 2, 3]
OK
Cancel
Apply
Help


Simulation Parameters: Parameters
Solver
SPICE
Options
Diagnostics
Initialization
System state
Initialize from:
Block parameters
Stored system state
Store current state...
Model initialization commands
1 x=5

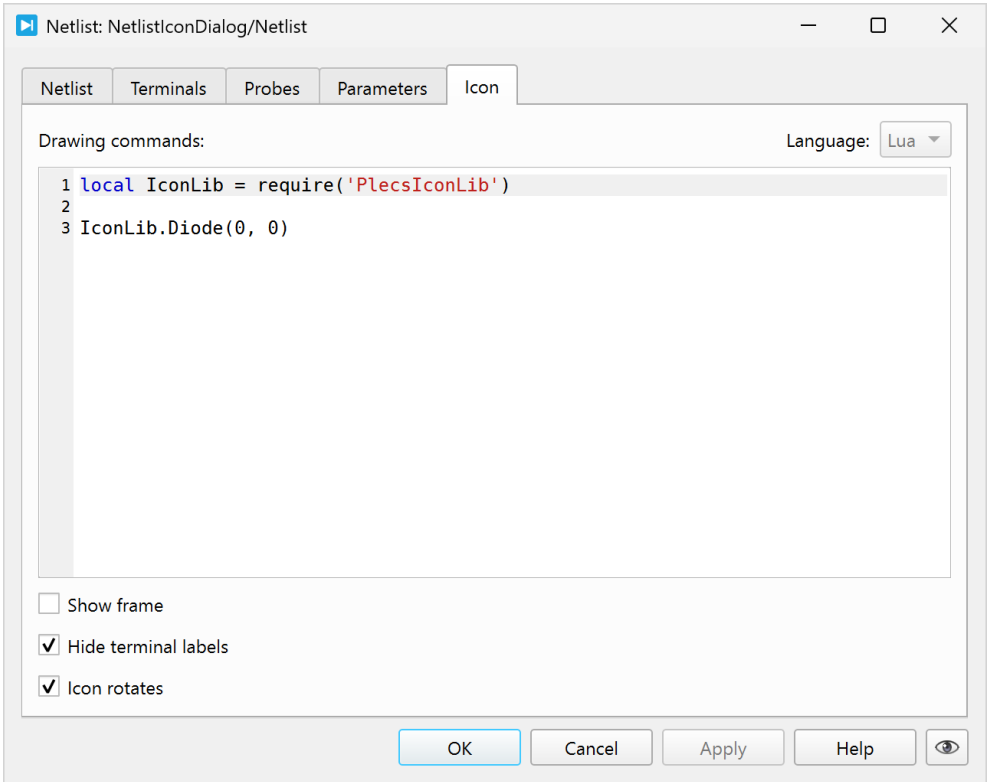


Fig. 14.12: Defining a custom diode icon.

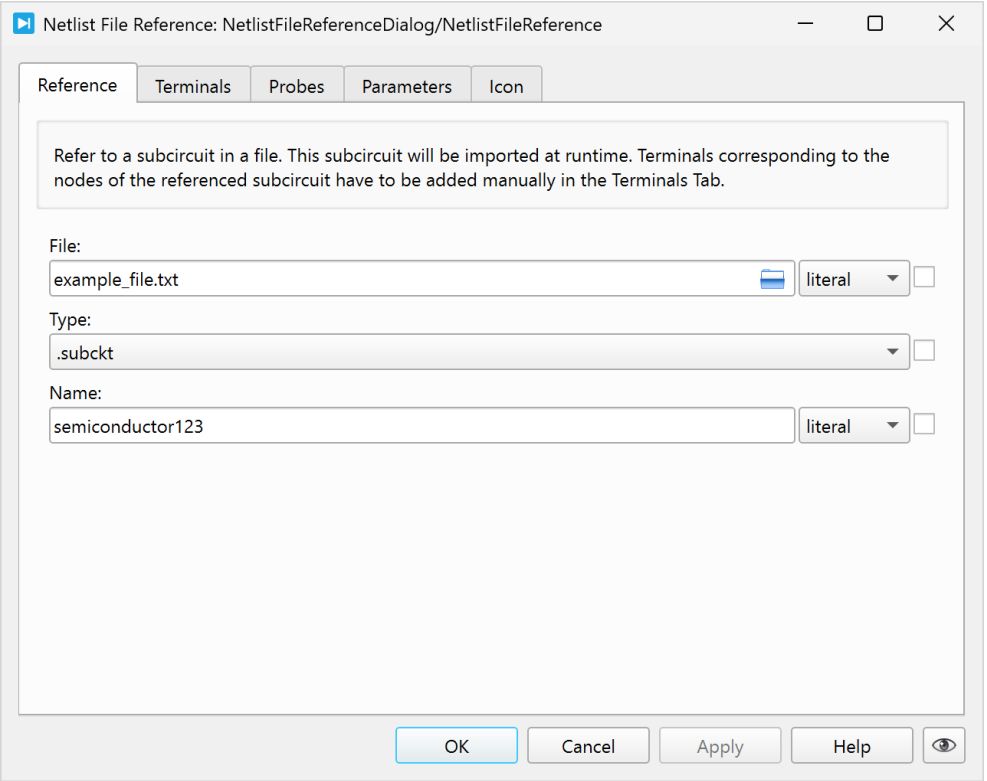


Fig. 14.13: Netlist file reference setup to match the import depicted in Fig. 14.7 on p. 419.

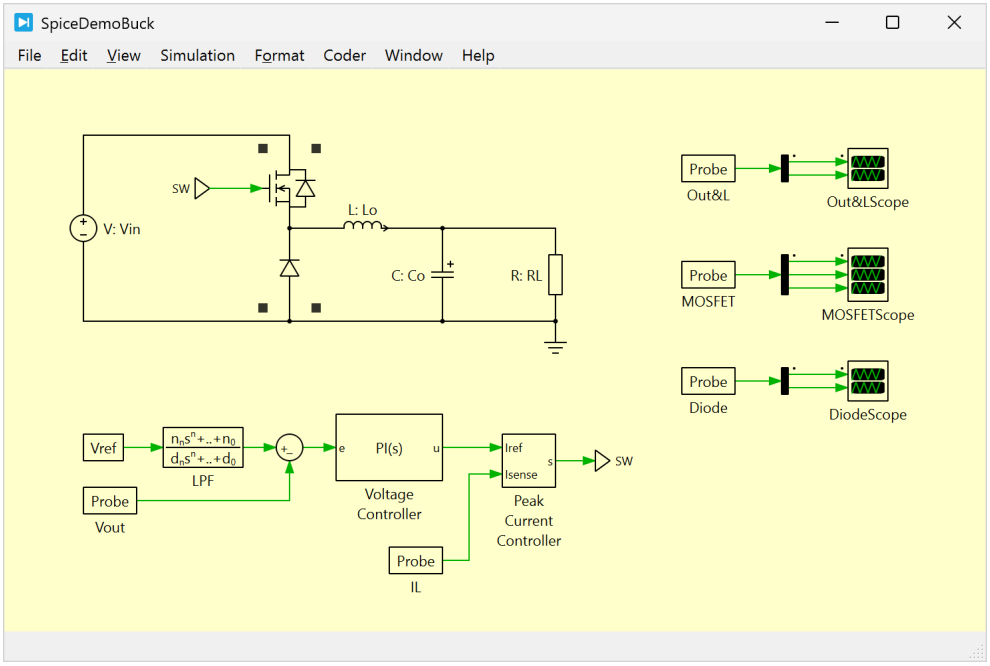


Fig. 14.14: PLECS Buck converter with peak current controller

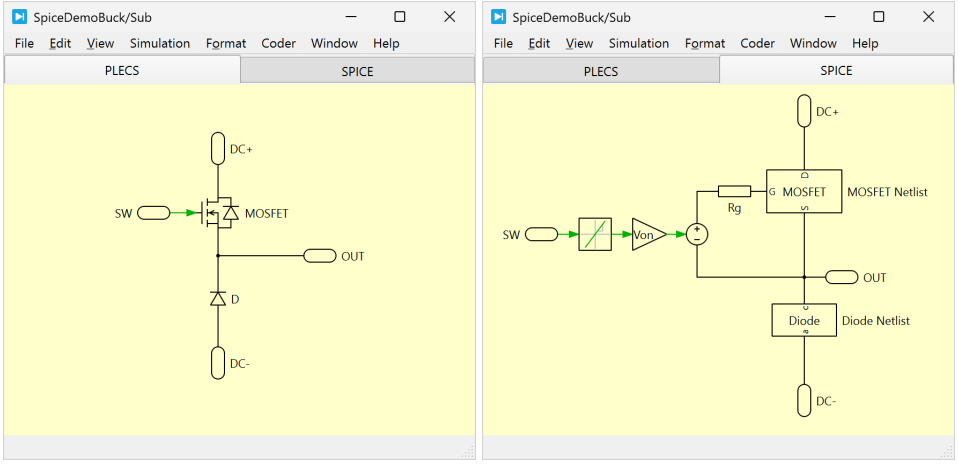


Fig. 14.15: Configurable subsystem for switching between PLECS and SPICE configurations.

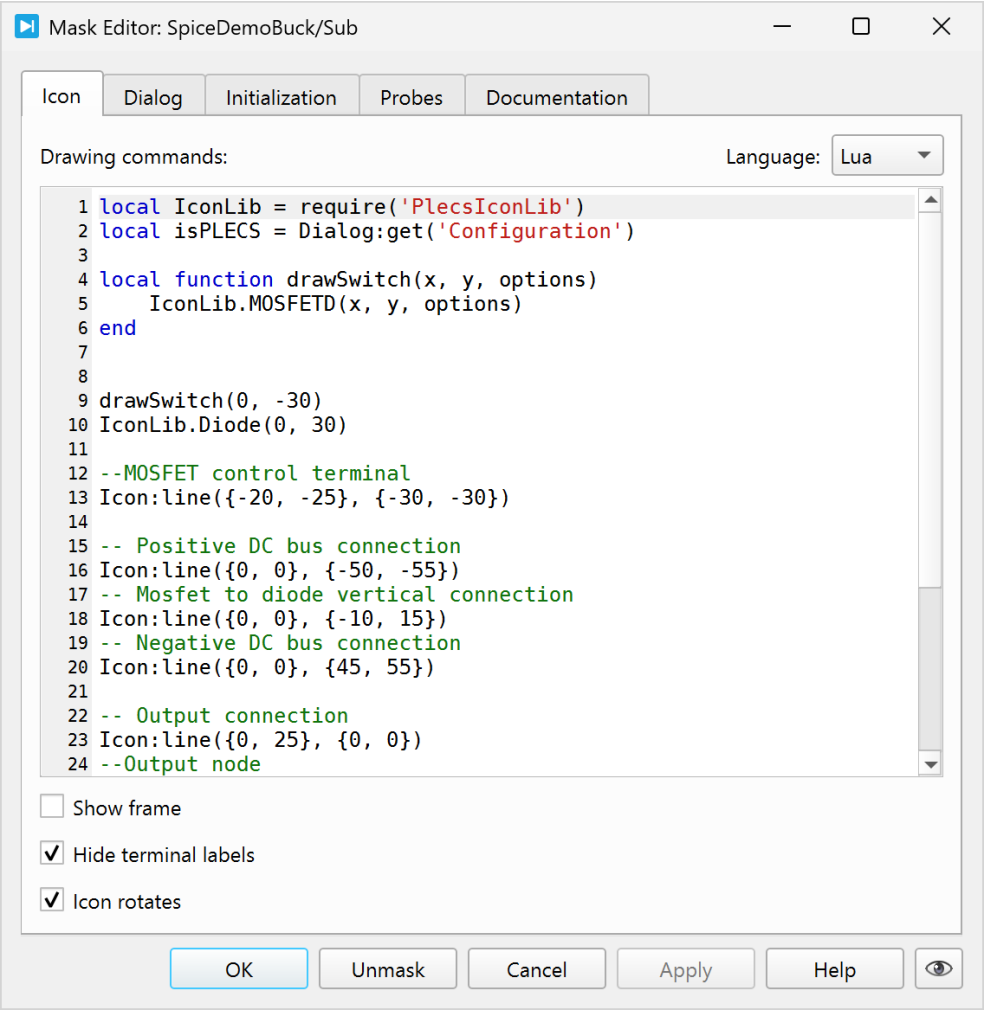


Fig. 14.16: A masked subsystem can be have an icon representing its contents.

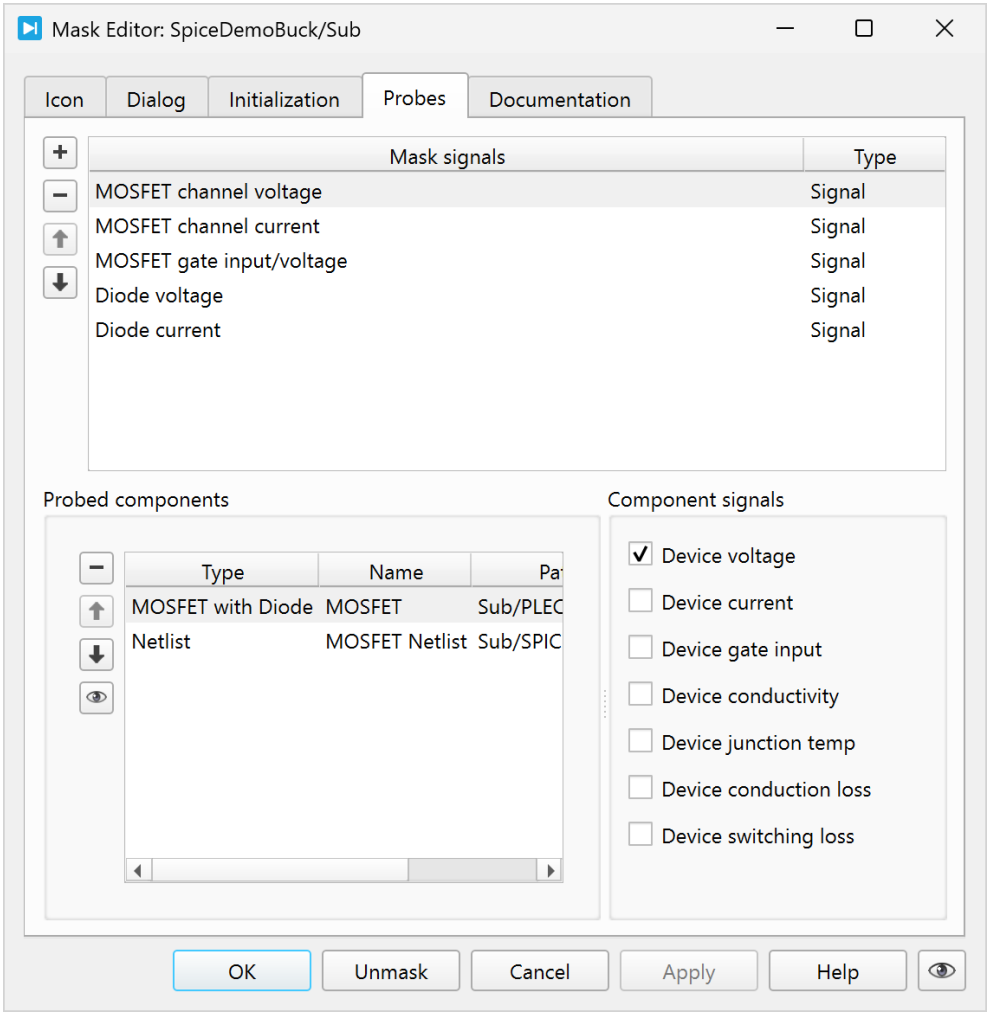


Fig. 14.17: The *Probes* tab of a masked subcircuit, defining Mask signals by referencing components from its configurations.

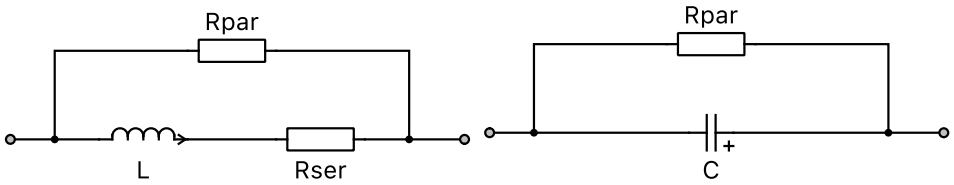


Fig. 14.18: Parallel and series resistances applied to inductors and capacitors when using the SPICE regularization options.

Netlist Syntax Reference

> **Note**
>
> Example netlists introducing most of the netlist syntax are available as “Netlist Cheat Sheets” (p. 411).

The following is a technical documentation of the supported syntax.

The **Recommended** code snippets throughout this section contain clear and unambiguous netlist syntax. To see supported legacy syntax required for compatibility, refer to the online documentation or the help viewer in PLECS.

Fundamentals

Syntax conventions

Recommended

- All netlist statements are case-insensitive.
- Any line starting with `*` is treated as a comment.
- Anything after a `;` is treated as a comment.
- Statements can be continued on the next line using the + character.
- Use spaces for separation. Use = to assign parameters.

```
* This is a comment.
.PARAM a=1; This is also a comment.
R1 n1 N1 1; Since n1=N1, this resistor is short-circuited.
R2 n1 n2
+ {1e6*12}; this defines the resistance of R2
```

Nodes

Node names can be integers or alphanumeric strings, with node `0` always reserved for ground. We recommend to use descriptive names that clearly distinguish themselves from the names of circuit elements, parameters and functions to make the netlist more readable.

Constants

Recommended

Constant numbers should be specified using scientific notation.

```
R1 n1 n2 4.356e4
```

Expressions

Recommended

Instead of assigning a constant value, mathematical expressions can be used. These mathematical expressions should be put into curly braces. Expressions can evaluate parameters (p. 472) and functions (p. 470).

```
.PARAM a=1 b={a} c={a+b}
.FUNC f(x,y)={x+y}
R1 n1 n2 {a}
R2 n1 n2 {f(b,c)}
```

Predefined Parameters and Functions

PLECS Spice provides a rich set of built-in mathematical operators, functions, and constants that you can use in expressions. In user defined source expressions, certain circuit values can be accessed (see “Circuit State Access” (p. 432)).

Mathematical Operators

All standard arithmetic and comparison operators are supported:

```
.PARAM a={1+2}      ; Addition
.PARAM b={1-2}      ; Subtraction
.PARAM c={1*2}      ; Multiplication
.PARAM d={1/2}      ; Division
.PARAM e={1+(2*3)} ; Curly braces for grouping

* Both ** and ^ perform exponentiation:
.PARAM a={3**2}     ; 3 squared
.PARAM b={3^0.5}    ; square root of 3
```

> **Note**
>
> Some SPICE versions treat ^ as logical XOR. In PLECS Spice ^ is always exponentiation.

Mathematical Functions

All standard functions are available:

```
* Trigonometric functions (angles in radians)
sin(x)    cos(x)    tan(x)
asin(x)   acos(x)   atan(x)   atan2(y,x)

* Hyperbolic functions
sinh(x)   cosh(x)   tanh(x)

* Exponential and logarithm
exp(x)                    ; e^x
log(x)   ln(x)            ; Natural logarithm (base e)
log10(x)                  ; Logarithm base 10

* Power and root
pow(x,y)                  ; x^y
sqrt(x)                   ; Square root
abs(x)                    ; Absolute value

* Rounding and extrema
floor(x)  ceil(x)  round(x) sign(x)
min(x,y)  max(x,y)
```

Note

To avoid convergence issues, we ensure that the domains of all functions are extended to the real numbers $\mathbb{R}$. Further, for extremely large or undefined values, we modify the following functions:

- `exp(x)` for values of `x` exceeding approximately 70.
- `log(x)`, `ln(x)` and `log10(x)` for values of `x` below approximately 10^{-30}.
- `pow(x,y)` for `x < 0` and `y` not an integer.

Via the "invalid netlist function evaluation" setting in "SPICE Simulation Parameters" (p. 406), you can throw warnings or errors when this occurs.

SPICE-Specific Functions

Certain special functions are commonly used in SPICE:

```
limit(x,y,z)
* Sorts the values x, y, z and returns the value in the middle.
* This is equivalent to clamping/limiting x such that is not larger than y␣
↪and not smaller than z.

table(x, a1, b1, a2, b2, ...)
* Linear interpolation through points (a1,b1), (a2,b2), ... evaluated at x.
* The values a1, a2, a3, ... must be in ascending or descending order.
```

(continues on next page)

(continued from previous page)

```
pwr(x,y)  ;= pow(abs(x), y)
pwrs(x,y) ;= sign(x) * pwr(x,y)
```

Logical Expressions

Recommended

You can use `if(condition, true_value, false_value)` or `condition ? true_value : false_value` to either use `true_value` when `condition` is true or `false_value` when `condition` is false.

The `condition` should be the result of a comparison `== != < > <= >=`. Multiple comparison results can be combined using `&` as "and" or using `|` as "or".

```
.FUNC custom_max(a,b)={if(a>b, a, b)}
.FUNC custom_min(a,b)={a<b ? a : b}
.FUNC custom_limit(x,y,z)={if(x>=y & x<=z, x, if(x<y && y<z, y, z))}
```

> **Note**
>
> Some SPICE versions support `~` or `!` as "not". Neither of these is currently supported in PLECS Spice.

Constants

Recommended

The parameter `pi` gives access to the mathematical constant $\pi \approx 3.14159$.

```
.PARAM freq=50
.PARAM omega={2*pi*freq}
```

Circuit State Access

In function based source expressions (p. 435), you can access voltages, currents and the simulation time.

```
* Voltage measurements
V(n1)                  ; Voltage at node relative to ground
V(n1, n2)              ; Voltage difference V(n1) - V(n2)

* Current measurements
I(V1)                  ; Current through voltage source V1
```

(continues on next page)

(continued from previous page)

```
* Simulation time
TIME                    ; Current simulation time

* Time derivatives of voltages and currents
ddt(V(n))               ; dV/dt
ddt(I(L1))              ; dI/dt
* Note: ddt() can be applied to arbitrary expressions, but this is not␣
↪recommended.
* Note: ddt() uses numerical differentiation and may introduce numerical␣
↪noise.
```

Currents can be accessed for the following elements:

- all voltage sources
- "(R) Resistor Statement" (p. 462)
- "(I) Current Source Statement" (p. 442) with constant value
- "(B) Function-Based Source Statement" (p. 435)
- "(G) Voltage-Controlled Current Source Statement" (p. 441)
- "(F) Current-Controlled Current Source Statement" (p. 441)

Scopes and Shadowing

Outside the Main Subcircuit

As explained in "Fundamentals" (p. 397), PLECS Spice treats the first subcircuit in a netlist component as main subcircuit. In the global scope outside of this main subcircuit, you can define **.SUBCKT**, **.MODEL**, **.PARAM** and **.FUNC**. Anything defined like this can be used in the main subcircuit or overwritten there. If a definition is unused, it is not parsed and syntax errors will not be reported.

Other text outside of the main subcircuit will be ignored.

```
.SUBCKT main n1 n2
R1 n1 n2 {a} ; This accesses the parameter a defined outside of the main␣
↪subcircuit.
.ENDS

.PARAM a=1

R2 n 0 1; This will be ignored.

.SUBCKT unused m1 m2
* This subcircuit is unused - anything in here is not parsed.
.ENDS
```

Scopes

When defining **.SUBCKT**, **.MODEL**, **.PARAM** and **.FUNC** inside of a subcircuit, these definitions are only accessible in its scope, i.e. between the subcircuit's **.SUBCKT** and **.ENDS**.

```
.SUBCKT main n1 n2
X n1 n2 helper
* The parameter b defined in helper is not accessible here.
.ENDS

.SUBCKT helper m1 m2
.PARAM b=2
R1 m1 m2 {b}
.ENDS
```

You may only define each **.SUBCKT**, **.MODEL**, **.PARAM** and **.FUNC** once per scope. Any **.SUBCKT**, **.MODEL**, **.PARAM** and **.FUNC** definitions within a subcircuit can be freely placed and can be used before they are defined.

```
.PARAM a=1;

.SUBCKT main n1 n2
R n1 n2 {a} ; Uses a=2 even though it is defined after this line.
.PARAM a=2 ; This overwrite the a=1 from outside the main subcircuit.
.ENDS
```

Shadowing

When X statements (p. 469) are used to instantiate a subcircuit, this instantiated subcircuit has access to all definitions available in the subcircuit where the X statement is located. Inside the instantiated subcircuit these definitions can be overwritten. Such shadowing does not apply to node names. Node names are always local to their subcircuit. Shadowing also works recursively when instances of subcircuits create other instances of subcircuits.

```
.SUBCKT main n1 n2
.PARAM a=1
.PARAM b=2
X n1 n2 helper
.ENDS

.SUBCKT helper m1 m2
R1 m1 0 {a}; Resistance is 2 using b=2 from the main subcircuit.
.PARAM a=3; This overwrites a to be 3 while inside this subcircuit.
R2 m1 0 {a}; Resistance is 3.
.ENDS
```

Subtleties of Shadowing

Whenever definitions contain an expression (p. 430), they will be evaluated only in the scope of their actual use. When using `PARAMS:` in X statements (p. 469), this causes an evaluation in the scope of the X statement.

```
.SUBCKT sub1 n
.PARAM a=1 b={a}
X1 n sub2
X2 n sub2 PARAMS: b={b}
.ENDS

.SUBCKT sub2 n
.PARAM a=2
R1 n 0 {b}
.ENDS
```

In the netlist above:

- `X1` creates a resistor with resistance `{b}` evaluated in `sub2`. This evaluates to `2` as `b=a=2` in the scope of `sub2`.
- `X2` creates a resistor with the resistance value `b={b}` passed after `PARAMS:`. This is evaluated in `sub1` and evaluates to `1` as `b=a=1` in the scope of `sub1`.

Note

To avoid shadowing subtleties, we recommend to:

- Define each `.SUBCKT`, `.MODEL`, and `.FUNC` only once and in the global scope.
- When passing parameters to subcircuits, only pass parameters that are listed with a default value after `PARAMS:` in the subcircuit definition (p. 473).

List of Statements

Each non-empty line represents a statement whose type is determined by its first character(s):

(B) Function-Based Source Statement

Function-based sources allow you to define the output voltage or current using arbitrary expressions (p. 430). The output can depend on node voltages, device currents, time, and parameters (p. 432).

Voltage Source

Recommended

```
Bname pos neg V={function body}
```

The `function body` has to be an expression (p. 430).

Current Source

Recommended

```
Bname pos neg I={function body}
```

The `function body` has to be an expression (p. 430).

The evaluated function body may reference the state of the circuit (p. 432).

Examples

```
B1 n1 0 I={I(V1) + ddt(V(n4))}
B2 n2 0 V={sin(V(n5, n6))}
V1 n3 0 1
```

(C) Capacitor Statement

Capacitors in PLECS Spice support initial conditions, temperature dependence, and parasitic series/parallel resistances. Furthermore, nonlinear capacitors with user-defined charge functions are supported.

Linear Capacitor

Recommended

```
Cname pos neg capacitance <IC=ic> <TC1=tc1> <TC2=tc2> <TEMP=temp> <M=m>

Cname pos neg modelName <capacitance> <L=l> <W=w> <IC=ic> <TC1=tc1>␣
↪<TC2=tc2> <TEMP=temp> <M=m>
.MODEL modelName C <NAME=value ...>
```

- The device type `C` must be specified in the model statement.
- The capacitance can also be set as the model parameter `CAP`. If it's defined in both the model and the instance, the instance value is used.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

Instance Parameters

Parameter	Description	Default Value
IC	Initial voltage across the capacitor	*none*
TC1	Temperature coefficient for CAP (first order correction)	$0\,^{\circ}\mathrm{C}^{-1}$
TC2	Temperature coefficient for CAP (second order correction)	$0\,^{\circ}\mathrm{C}^{-2}$
TEMP	Device temperature	$27\,^{\circ}\mathrm{C}$
M	Number of parallel devices	1
L	Length of the semiconductor capacitor	$0\,\mathrm{m}$
W	Width of the semiconductor capacitor	DEFW

- The default value for L is not permitted. Thus, a nonzero value must be set either in the instance statement or as a model parameter.

Model Parameters

Parameter	Description	Default Value
T_ABS	Device temperature	$27\,^{\circ}\mathrm{C}$
CAP	Capacitance	$0\,\mathrm{F}$
CJ	Bottom junction capacitance	$0\,\mathrm{F/m^2}$
CJSW	Sidewall junction capacitance	$0\,\mathrm{F/m}$
DEFW	Default width of the semiconductor capacitor	$10^{-6}\,\mathrm{m}$
NARROW	Narrowing due to the etching process	$0\,\mathrm{m}$

- You can specify the instance parameters TC1, TC2, L, and W as model parameters too. If both are provided, the instance values take priority.
- The instance parameter TEMP overwrites the model parameter T_ABS.
- When the capacitance is specified either in the instance statement or through

the parameter CAP, the parameters L, W, CJ, CJSW, DEFW, and NARROW are ignored.

- The parameters CAP, CJ, and CJSW are temperature-corrected using TC1 and TC2.

Linear Capacitor with Parasitic Elements

PLECS Spice supports a model of a linear capacitor with equivalent series resistance (RSER) and an additional resistance (RPAR) connected in parallel with the series combination of the ideal capacitor and the series resistor.

Recommended

```
Cname pos neg capacitance <M=m> <RSER=rser> <RPAR=rpar>
```

Instance Parameters

Parameter	Description	Default Value
M	Number of parallel devices	1
RSER	Series resistance	$0\,\Omega$
RPAR	Parallel resistance	$+\infty\,\Omega$

> **Note**
>
> - Initial conditions are not supported for linear capacitors with parasitic elements.

Charge-based Capacitor

Recommended

```
Cname pos neg Q={function body}
```

Examples

```
Csnub n1 n2 100n
Cesr n3 n4 10u RSER=0.01
Cvar n5 n6 Q={1e-9*V(n5,n6)*(1+0.1*V(n5,n6)^2)}
```

(D) Diode Statement

Recommended

```
Dname pos neg modelName <<AREA>=area> <IC=vd> <TEMP=temp> <M=m>
.MODEL modelName D <NAME=value ...>
```

- The diode type `D` must be specified in the model statement.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

Instance Parameters

Parameter	Description	Default Value
`AREA`	Device area (number of parallel devices)	1
`IC`	Initial voltage across the diode	*none*
`TEMP`	Device temperature	27 °C
`M`	Number of parallel devices	1

Model Parameters

Parameter	Value	Default Value
T_ABS	Device temperature	27 °C
TNOM	Nominal temperature	27 °C
IS	Reverse saturation current	10^{-14} A
XTI	Temperature exponent for IS	3
EG	Band gap energy	1.11 eV
N	Emission coefficient	1
BV	Reverse breakdown voltage	∞ V
IBV	Reverse breakdown current	10^{-3} A
NBV	Emission coefficient for the reverse breakdown current	1
IKF	High-level injection knee current	∞ A
TIKF	Temperature coefficient for IKF (first order correction)	$0\,°C^{-1}$
ISR	Recombination saturation current	0 A
NR	Emission coefficient for the recombination current	1
RS	Series resistance	0 Ω
TRS1	Temperature coefficient for IKF (first order correction)	$0\,°C^{-1}$
TRS2	Temperature coefficient for IKF (second order correction)	$0\,°C^{-2}$
VJ	Junction potential	1 V
M	Junction grading coefficient	0.5
CJO / CJO	Zero-bias junction capacitance	0 F
FC	Forward-bias depletion capacitance coefficient	0.5
TT	Forward transit time	0 s

- The instance parameter `TEMP` overwrites the model parameter `T_ABS`.
- The parameters representing currents and capacitances (starting with `I` and `C`) are multiplied by the `AREA` parameter.
- The parameters representing resistances (starting with `R`) are divided by the `AREA` parameter.

> **Note**
>
> - Some parameters encountered in semiconductor models are provided as additional information and do not affect the simulation. PLECS Spice ignores the following parameters: `mfg`, `Iave`, `Vrev`.

Examples

```
.MODEL diodeModel D IS=1e-12 N=1.05 RS=0.02 TT=5n CJO=100p BV=100
D1 n1 n2 diodeModel
```

(E) Voltage-Controlled Voltage Source Statement

Recommended

You can model voltage-controlled sources using “(B) Function-Based Source Statement” (p. 435). This is purely for supporting legacy syntax.

(F) Current-Controlled Current Source Statement

Recommended

You can model current-controlled sources using “(B) Function-Based Source Statement” (p. 435). This is purely for supporting legacy syntax.

(G) Voltage-Controlled Current Source Statement

Recommended

You can model voltage-controlled sources using “(B) Function-Based Source Statement” (p. 435). This is purely for supporting legacy syntax.

(H) Current-Controlled Voltage Source Statement

Recommended

You can model current-controlled sources using “(B) Function-Based Source Statement” (p. 435). This is purely for supporting legacy syntax.

(I) Current Source Statement

Define a current source either as constant DC source or by using specific parametrized functions. If you want to use a different source function, you can use “(B) Function-Based Source Statement” (p. 435).

```
Iname pos neg <DC> dcvalue
Iname pos neg [function]
```

See “Source Statement Functions” for `[function]` syntax. If no function is specified, the source is treated as DC source with `dcvalue=0`.

> **Note**
>
> - We do not support AC-related arguments in sources. Please use PLECS blocks for AC analysis.
> - We only support sources that provide either a `dcvalue` or a `[function]`. Some SPICE version allow to provide both simultaneously, which is equivalent to putting two sources in series.

Examples

```
I1 n 0 1
I2 n 0 SIN 0 1 50
```

Source Statement Functions

PULSE Function

```
PULSE initialValue pulsedValue <delayTime <riseTime <fallTime <pulseWidth␣
↪<period>>>>>
```

> **Note**
>
> At `delayTime`, the pulse will start at `initialValue` and change to `pulsedValue` within `riseTime`. Consequently, a pulse without delay, rise and fall time `PULSE 0 1 0 0 0` will output `0` in the very first simulation step and `1` in the next. This differs from the PLECS pulse (p. 895) which starts in high-state.

Table 14.1: PULSE Parameters

Parameter	Default	Description
`initialValue`	*required*	Initial value before pulse
`pulsedValue`	*required*	Value during pulse
`delayTime`	0	Absolute time after which the pulse starts (not phase delay)
`riseTime`	0	Time for rising edge transition
`fallTime`	0	Time for falling edge transition
`pulseWidth`	∞	Duration of the pulse at pulsedValue
`period`	∞	Period of the pulse waveform

SIN/SINE Function

```
SIN<E> offset amplitude <frequency <delay <damping>>>
```

Table 14.2: SIN Parameters

Parameter	Default	Description
`offset`	*required*	DC offset
`amplitude`	*required*	Amplitude of sine wave
`frequency`	0	Frequency, in hertz (Hz)
`delay`	0	Absolute time after which the sine starts (not phase delay)
`damping`	0	Damping factor

Note

Our default frequency of 0 differs from most other SPICE versions where $1/T_{\text{stop}}$ is used.

SFFM Function

```
SFFM offset amplitude <frequency <modulationIndex <signalFrequency>>>
```

Table 14.3: SFFM Parameters

Parameter	Default	Description
`offset`	*required*	DC offset
`amplitude`	*required*	Carrier amplitude
`frequency`	0	Carrier frequency, in hertz (Hz)
`modulationIndex`	0	Modulation index
`signalFrequency`	0	Signal frequency, in hertz (Hz)

Note

Our default frequency of 0 differs from most other SPICE versions where $1/T_{\text{stop}}$ is used.

EXP Function

```
EXP initialValue pulsedValue <riseDelayTime <riseTimeConstant␣
↪<fallDelayTime <fallTimeConstant>>>>
```

Table 14.4: EXP Parameters

Parameter	Default	Description
`initialValue`	*required*	Initial value
`pulsedValue`	*required*	Pulsed value
`riseDelayTime`	0	Delay before rising exponential starts
`riseTimeCon-stant`	0	Time constant for rising exponential
`fallDelayTime`	0	Delay before falling exponential starts
`fallTimeCon-stant`	0	Time constant for falling exponential

> **Note**
>
> Our defaults for riseTimeConstant, fallDelayTime, and fallTimeConstant differ from most SPICE versions where $1/T_{\text{step}}$ is used.

PWL Function

```
PWL t1 v1 t2 v2 <t3 v3 <...>>
```

Piecewise linear interpolation with value `v1` at time `t1`, `v2` at time `t2`, etc.

Table 14.5: PWL Requirements

Requirement	Description
Time ordering	Times must be strictly increasing: $t_1 < t_2 < t_3 < \ldots$
Minimum points	At least one time-value pair required

(J) JFET Statement

We support the standard SPICE junction field-effect transistor (JFET) model[9][10]. We have made minor improvements to the PN-junction currents in

[9] P. Antognetti and G. Massobrio, "Semiconductor Device Modeling with Spice", 2nd ed., McGraw-Hill, Inc., USA, 1993.

[10] A. E. Parker and D. J. Skellern, "An improved FET model for computer simulators," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 9, no. 5, pp. 551-

reverse bias to ensure continuity of the modeling equations.

Recommended

```
Jname drain gate source modelName <<AREA>=area> <IC=vds,vgs> <TEMP=temp>␣
↪<M=m>
.MODEL modelName NJF <NAME=value ...>
```

- The JFET device type `NJF` or `PJF` must be specified in the model statement.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

Instance Parameters

Parameter	Description	Default Value
`AREA`	Device area (number of parallel devices)	1
`IC`	Initial condition voltages (V_{ds}, V_{gs}).	*none*
`TEMP`	Device temperature	27 °C
`M`	Number of parallel devices	1

- When the `IC` parameter is given, initial values for both the drain-source (V_{ds}) and the gate-source voltage (V_{gs}) must be specified.

553, May 1990

Model Parameters

Parameter	Value	Default Value
T_ABS	Device temperature	$27\,^{\circ}\mathrm{C}$
TNOM	Nominal temperature	$27\,^{\circ}\mathrm{C}$
VTO	Threshold voltage	$-2\,\mathrm{V}$
BETA	Transconductance	$10^{-4}\,\mathrm{A/V^2}$
LAMBDA	Channel length modulation	$0\,\mathrm{V^{-1}}$
B	Parker-Skellern fit parameter (Doping tail parameter)	1
IS	Reverse saturation current for the gate-source and gate-drain junctions	$10^{-14}\,\mathrm{A}$
XTI	Temperature exponent for IS	1
EG	Band gap energy	$1.11\,\mathrm{eV}$
N	Emission coefficient for the gate-source and gate-drain junctions	1
PB	Gate junction potential	$1\,\mathrm{V}$
M	Gate junction grading coefficient	0.5
CGS	Gate-source zero-bias junction capacitance	$0\,\mathrm{F}$
CGD	Gate-drain zero-bias junction capacitance	$0\,\mathrm{F}$
FC	Forward-bias depletion capacitance coefficient	0.5
RS	Source resistance	$0\,\Omega$
RD	Drain resistance	$0\,\Omega$

- The instance parameter TEMP overwrites the model parameter T_ABS.
- The parameters representing currents and capacitances (starting with I and C) are multiplied by the AREA parameter.
- The parameters representing resistances (starting with R) are divided by the AREA parameter.
- The parameter BETA is multiplied by the device area.

Note

- Some parameters encountered in semiconductor models are provided as additional information and do not affect the simulation. PLECS Spice ignores the following parameters: `mfg`.

Examples

```
J1 drain gate source nJFet
.MODEL nJFet NJF VTO=-2 BETA=1e-3 LAMBDA=0.01 RD=10 RS=10
```

References

(K) Mutual Inductor Coupling Statement

```
Kname L1name L2name <L3name <...>> k
```

The mutual inductor coupling models magnetic coupling between two or more inductors.

- `Kname` is the unique identifier for the coupling.
- `L1name`, `L2name`, etc. are the names of the inductors to be coupled.
- `k` is the coupling coefficient, where $0 \leq k \leq 1$.
  - $k = 0$ means no coupling (inductors are independent)
  - $k = 1$ means perfect coupling (ideal transformer)
  - typical values 0.95 to 0.99

The mutual inductance M between inductors is calculated as:

$$M = k\sqrt{L_1 \cdot L_2}$$

For coupling more than two inductors, the same coupling coefficient k applies to all pairs.

Examples

```
L1 n1 n2 100u
L2 n3 n4 400u
K1 L1 L2 0.98
```

```
L1 n1 n2 100u
L2 n3 n4 400u
L3 n5 n6 900u
K1 L1 L2 L3 0.95
```

(L) Inductor Statement

Inductors in PLECS Spice support initial current conditions, temperature dependence, and parasitic series/parallel resistances.

Linear Inductor

Recommended

```
Lname pos neg inductance <IC=ic> <TEMP=temp> <M=m>

Lname pos neg modelName <inductance> <IC=ic> <TEMP=temp> <M=m>
.MODEL modelName L <NAME=value ...>
```

- The device type `L` must be specified in the model statement.
- The inductance can also be set as the model parameter `IND`. If it's defined in both the model and the instance, the instance value is used.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

Instance Parameters

Parameter	Description	Default Value
`IC`	Initial current through the inductor	*none*
`TEMP`	Device temperature	27 °C
`M`	Number of parallel devices	1

Model Parameters

Parameter	Description	Default Value
T_ABS	Device temperature	$27\,^{\circ}\mathrm{C}$
IND	Inductance	$0\,\mathrm{H}$
TC1	Temperature coefficient for IND (first order correction)	$0\,^{\circ}\mathrm{C}^{-1}$
TC2	Temperature coefficient for IND (second order correction)	$0\,^{\circ}\mathrm{C}^{-2}$

- The instance parameter TEMP overwrites the model parameter T_ABS.

Linear Inductor with Parasitic Elements

PLECS Spice supports a model of a linear inductor with equivalent series resistance (RSER) and an additional resistance (RPAR) connected in parallel with the series combination of the ideal inductor and the series resistor.

Recommended

```
Lname pos neg inductance <M=m> <RSER=rser> <RPAR=rpar>
```

Instance Parameters

Parameter	Description	Default Value
M	Number of parallel devices	1
RSER	Series resistance	$0\,\Omega$
RPAR	Parallel resistance	$+\infty\,\Omega$

Note

- Initial conditions are not supported for linear inductors with parasitic elements.
- You may not couple inductors with parasitic contributions using the mutual inductor coupling statement (`K`).

Examples

```
L1 mid out 5u
```

(M) MOSFET Statement

PLECS Spice supports a modified version of the standard Level 1 and Level 3 MOSFET model as well as a model for a vertically-diffused MOSFET (VDMOS).

Classic SPICE MOSFET models

The static equations for the supported `LEVEL=1` and `LEVEL=3` MOSFET models are identical to a larger extent to those of the original Spice3f5 models. We have made minor modeling improvements to make the equations continuous, notably for the bulk junctions and for the evaluation of the threshold voltage. Regarding the large-signal model, we do not support the Meyer model of the intrinsic charges because it does not conserve charge. Instead, a simplified version of the BSIM3v3.3 charge model is implemented (`capMod=2`, 0/100 charge partition).

Recommended

```
Mname drain gate source bulk modelName <<L>=l> <<W>=w> <AD=ad> <AS=as>␣
↪<PD=pd> <PS=ps> <NRD=nrd> <NRS=nrs> <IC=vds,vgs,vbs> <TEMP=temp> <M=m>
.MODEL modelName NMOS <NAME=value ...>
```

- The MOSFET device type `NMOS` or `PMOS` must be specified in the model statement.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).
- When importing a MOSFET model, the bulk node will be set equal to the source node

Instance Parameters

Parameter	Description	Default Value
L	Channel length	$10^{-4}\,\mathrm{m}$
W	Channel width	$10^{-4}\,\mathrm{m}$
AD	Drain diffusion area	$0\,\mathrm{m}^2$
AS	Source diffusion area	$0\,\mathrm{m}^2$
PD	Drain diffusion perimeter	$0\,\mathrm{m}$
PS	Source diffusion perimeter	$0\,\mathrm{m}$
NRD	Number of drain squares	1
NRS	Number of source squares	1
IC	Initial condition voltages ($V_{\mathrm{ds}}, V_{\mathrm{gs}}, V_{\mathrm{bs}}$).	*none*
TEMP	Device temperature	$27\,^{\circ}\mathrm{C}$
M	Number of parallel devices	1

- When the IC parameter is given, initial values for the drain-source (V_{ds}), the gate-source (V_{gs}), and bulk-source voltage (V_{bs}) must be specified.

Model Parameters

Parameter	Description	Default Value
T_ABS	Device temperature	$27\,^{\circ}\mathrm{C}$
TNOM	Nominal temperature	$27\,^{\circ}\mathrm{C}$
LEVEL	Specifies the level of the MOSFET model	1
VTO / VTO	Zero-bias threshold voltage	$0\,\mathrm{V}$
PHI	Surface-inversion potential	$0.6\,\mathrm{V}$

continues on next page

Table 14.6 – continued from previous page

Parameter	Description	Default Value
KP	Transconductance	$2 \times 10^{-5}\,\mathrm{A/V^2}$
GAMMA	Bulk threshold parameter	$0\,\mathrm{V^{0.5}}$
UO / U0	Surface mobility	$600\,\mathrm{cm^2/(Vs)}$
NSS	Surface state density	$0\,\mathrm{cm^{-2}}$
NSUB	Substrate doping	$0\,\mathrm{cm^{-3}}$
TPG	Type of the gate material	1
LD	Lateral diffusion length	$0\,\mathrm{m}$
TOX	Gate insulation oxide thickness	$10^{-7}\,\mathrm{m}$
CGSO	Gate-source overlap capacitance per channel width	$0\,\mathrm{F/m}$
CGDO	Gate-drain overlap capacitance per channel width	$0\,\mathrm{F/m}$
CGBO	Gate-bulk overlap capacitance per channel length	$0\,\mathrm{F/m}$
IS	Reverse saturation current for the bulk-drain and bulk-source junctions	$10^{-14}\,\mathrm{A}$
JS	Reverse saturation current density for the bulk-drain and bulk-source junctions	$0\,\mathrm{A/m^2}$
XTI	Temperature exponent for IS and JS	1
N	Emission coefficient for the bulk-drain and bulk-source junctions	1
PB	Junction potential for the bulk-drain and bulk-source junctions	$0.75\,\mathrm{V}$
MJ	Grading coefficient for the bulk-drain and bulk-source junctions	0.5
CBD	Bulk-drain zero-bias junction capacitance	$0\,\mathrm{F}$

continues on next page

Table 14.6 – continued from previous page

Parameter	Description	Default Value
CBS	Bulk-source zero-bias junction capacitance	$0\,\mathrm{F}$
CJ	Zero-bias junction capacitance per unit area	$0\,\mathrm{F/m^2}$
MJSW	Grading coefficient for the sidewall junction capacitance	0.5 (LEVEL=1) 0.33 (LEVEL=3)
CJSW	Zero-bias sidewall junction capacitance	$0\,\mathrm{F/m}$
FC	Forward-bias depletion capacitance coefficient	0.5
TT	Forward transit time for the bulk-drain and bulk-source junctions	$0\,\mathrm{s}$
RSH	Channel sheet resistance	$0\,\Omega$
RD	Drain resistance	NRD x RSH
RS	Source resistance	NRS x RSH
RG	Gate resistance	$0\,\Omega$
RB	Bulk resistance	$0\,\Omega$
RDS	Drain-source resistance	$+\infty\,\Omega$

- When TPG it set to 1 (-1), it is assumed that the gate material has opposite (same) doping than the substrate. For an aluminum gate, set TPG to zero.
- You can specify the instance parameters L, W, AD, AS, PD, PS, NRD, and NRS as model parameters too. If both are provided, the instance values take priority.
- The instance parameter TEMP overwrites the model parameter T_ABS.

Additional Parameters for the LEVEL=1 model

Parameter	Description	Default Value
LAMBDA	Channel length modulation	$0\,\mathrm{V^{-1}}$

Additional Parameters for the LEVEL=3 model

Parameter	Description	Default Value
DELTA	Width effect on threshold voltage	0
ETA	Static feedback on the threshold voltage	1
KAPPA	Saturation field factor	0.2
NFS	Fast surface state density	$0\,\mathrm{cm}^{-2}$
THETA	Mobility modulation	$0\,\mathrm{V}^{-1}$
VMAX	Maximum carrier drift velocity	$+\infty\,\mathrm{m/s}$
XJ	Metallurgical junction depth	$0\,\mathrm{m}$

Note

We assume a different default value for VMAX than the original SPICE solvers. You can set VMAX=0 to reproduce the standard SPICE behavior.

VDMOS model

A simple model for a VDMOS, based on the static LEVEL=1 MOSFET equations and an explicitly modeled body diode, is supported. While the gate-source capacitance is assumed constant, the gate-drain capacitance is modelled empirically using a nonlinear equation.

Recommended

```
Mname drain gate source modelName <IC=vds,vgs> <TEMP=temp> <M=m>
.MODEL modelName VDMOS <nchan> <NAME=value ...>
```

- The VDMOS device type VDMOS must be specified in the model statement. The polarity of the device is set with the nchan (default) or the pchan flag.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

Instance Parameters

Parameter	Description	Default Value
`IC`	Initial condition voltages (V_{ds}, V_{gs}).	*none*
`TEMP`	Device temperature	27 °C
`M`	Number of parallel devices	1

- When the `IC` parameter is given, initial values for both the drain-source (V_{ds}), and the gate-source (V_{gs}) voltage must be specified.

Model Parameters

Parameter	Description	Default Value
TNOM	Nominal temperature	$27\,^{\circ}\mathrm{C}$
VTO	Zero-bias threshold voltage	$0\,\mathrm{V}$
KP	Transconductance	$1\,\mathrm{A/V^2}$
LAMBDA	Channel length modulation	$0\,\mathrm{V^{-1}}$
MTRIODE	Conductance multiplier in the linear regime	1
SUBSHIFT	Shift in gate-source voltage of the subthreshold model	$0\,\mathrm{V}$
KSUBTHRES	Subthreshold parameter	$0.1\,\mathrm{V}$
CGDMAX	Maximum value of the gate-drain capacitance	$0\,\mathrm{F}$
CGDMIN	Minimum value of the gate-drain capacitance	$0\,\mathrm{F}$
A	Gate-drain capacitance parameter	1
CGS	Gate-source capacitance	$0\,\mathrm{F}$
RD	Drain resistance	$0\,\Omega$
RS	Source resistance	$0\,\Omega$
RG	Gate resistance	$0\,\Omega$
RDS	Drain-source resistance	$+\infty\,\Omega$
IS	Reverse saturation current (body diode)	$10^{-14}\,\mathrm{A}$
XTI	Temperature exponent for IS (body diode)	3
EG	Band gap energy (body diode)	$1.11\,\mathrm{eV}$
N	Emission coefficient (body diode)	1
BV	Reverse breakdown voltage (body diode)	$\infty\,\mathrm{V}$
IBV	Reverse breakdown current (body diode)	$10^{-10}\,\mathrm{A}$
NBV	Emission coefficient for the reverse breakdown current (body diode)	1
VJ	Junction potential (body diode)	$0.8\,\mathrm{V}$
M	Junction grading coefficient (body diode)	0.5
CJO	Zero-bias junction capacitance (body diode)	$0\,\mathrm{F}$
FC	Forward-bias depletion capacitance coefficient	0.5

Note

- Some parameters encountered in semiconductor models are provided as additional information and do not affect the simulation. PLECS Spice ignores the following parameters: `mfg`, `Vds`, `Ron`, `Qg`.

Examples

```
M1 d g s s nmosModel TEMP=125
.MODEL nmosModel NMOS LEVEL=1 VTO=3.5 KP=100 RD=0.05 RS=0.02 CGSO=1n␣
↪CGDO=500p
```

(Q) BJT Statement

The standard SPICE bipolar junction transistor (BJT) model based on the Gummel-Poon equations is supported[11][12]. We have made minor improvements to the PN-junction currents in reverse bias to ensure continuity of the modeling equations. The excess phase of the forward current is modeled using a companion model[13].

Recommended

```
Qname collector base emitter <substrate> modelName <<AREA>=area> <IC=vbe,
↪vce> <TEMP=temp> <M=m>
.MODEL modelName NPN <NAME=value ...>
```

- The `substrate` node is optional and connected to ground unless specified.
- The BJT device type `NPN` or `PNP` must be specified in the model statement.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).
- When importing a BJT model, the substrate node will be omitted.

[11] H. K. Gummel and H. C. Poon, "An integral charge control model of bipolar transistors," *Bell System Technical Journal*, vol. 49, no. 5, pp. 827-852, May-June 1970

[12] P. Antognetti and G. Massobrio, "Semiconductor Device Modeling with Spice", 2nd ed., McGraw-Hill, Inc., USA, 1993.

[13] *VBIC - Vertical Bipolar Intercompany Model*, https://designers-guide.org/vbic/

Instance Parameters

Parameter	Description	Default Value
AREA	Device area (number of parallel devices)	1
IC	Initial condition voltages (V_{be}, V_{ce}).	*none*
TEMP	Device temperature	27 °C
M	Number of parallel devices	1

- When the IC parameter is given, initial values for both the base-emitter (V_{be}) and the collector-emitter voltage (V_{ce}) must be specified.

Model Parameters

Parameter	Description	Default Value
T_ABS	Device temperature	27 °C
TNOM	Nominal temperature	27 °C
IS	Reverse saturation current	10^{-16} A
XTI	Temperature exponent for IS	3
EG	Band gap energy	1.11 eV
NF	Emission coefficient for the forward current	1
NR	Emission coefficient for the reverse current	1
BF	Ideal forward current gain	100
BR	Ideal reverse current gain	1
XTB	Temperature exponent for BF and BR	0
VAF / VA	Forward Early voltage	$+\infty$ V
VAR / VB	Reverse Early voltage	$+\infty$ V

continues on next page

Table 14.7 – continued from previous page

Parameter	Description	Default Value
IKF	Roll-off corner current for the forward current gain	$+\infty$ A
IKR	Roll-off corner current for the reverse current gain	$+\infty$ A
ISC	Reverse saturation current for the base-collector leakage junction	0 A
ISE	Reverse saturation current for the base-emitter leakage junction	0 A
NC	Emission coefficient for the base-collector leakage junction	2
NE	Emission coefficient for the base-emitter leakage junction	1.5
VJC	Base-collector junction potential	0.75 V
MJC	Base-collector junction grading coefficient	0.33
CJC	Base-collector zero-bias junction capacitance	0 F
XCJC	Fraction of the base-collector capacitance connected to the internal base node	1
VJE	Base-emitter junction potential	0.75 V
MJE	Base-emitter junction grading coefficient	0.33
CJE	Base-emitter zero-bias junction capacitance	0 F
VJS	Substrate-collector junction potential	0.75 V
MJS	Substrate-collector junction grading coefficient	0
CJS	Substrate-collector zero-bias junction capacitance	0 F
FC	Forward-bias depletion capacitance coefficient	0.5
TR	Reverse transit time	0 s

continues on next page

Table 14.7 – continued from previous page

Parameter	Description	Default Value
TF	Forward transit time	$0\,\mathrm{s}$
VTF	Voltage giving the base-collector voltage dependence of the forward transit time.	$+\infty\,\mathrm{V}$
ITF	Forward transit time fall-off current	$0\,\mathrm{A}$
XTF	Forward transit time bias dependence coefficient	1
PTF	Excess phase	0°
RC	Collector resistance	$0\,\Omega$
RE	Emitter resistance	$0\,\Omega$
RB	Zero-bias base resistance	$0\,\Omega$
RBM	Minimum base resistance at high current	RB
IRB	Current when the base resistance drops to (RB + RBM)/2	$+\infty\,\mathrm{A}$
TRC1	Temperature coefficient for RC (first order correction)	$0\,^\circ\mathrm{C}^{-1}$
TRC2	Temperature coefficient for RC (second order correction)	$0\,^\circ\mathrm{C}^{-2}$
TRB1	Temperature coefficient for RB (first order correction)	$0\,^\circ\mathrm{C}^{-1}$
TRB2	Temperature coefficient for RB (second order correction)	$0\,^\circ\mathrm{C}^{-2}$
TRE1	Temperature coefficient for RE (first order correction)	$0\,^\circ\mathrm{C}^{-1}$
TRE2	Temperature coefficient for RE (second order correction)	$0\,^\circ\mathrm{C}^{-2}$
TRM1	Temperature coefficient for RBM (first order correction)	$0\,^\circ\mathrm{C}^{-1}$

continues on next page

Table 14.7 – continued from previous page

Parameter	Description	Default Value
TRM2	Temperature coefficient for RBM (second order correction)	$0\,°\mathrm{C}^{-2}$

- The instance parameter TEMP overwrites the model parameter T_ABS.
- The parameters representing currents and capacitances (starting with I and C) are multiplied by the AREA parameter.
- The parameters representing resistances (starting with R) are divided by the AREA parameter.

Note

- For the parameters VAF, VAR, VTF, IKF, IKR, and IRB, a value of 0 is interpreted as $+\infty$.
- Some parameters encountered in semiconductor models are provided as additional information and do not affect the simulation. PLECS Spice ignores the following parameters: mfg, BVceo, Vceo, Ic, Icrating.

Examples

```
Q1 collector base emitter npnModel
.MODEL npnModel NPN IS=1e-14 BF=50 VAF=100 IKF=1
+ TF=10n TR=100n CJE=100p CJC=50p RC=1 RE=0.5 RB=5
```

References

(R) Resistor Statement

Resistors in PLECS Spice support temperature dependence and variable resistance based on expressions. Unlike fixed PLECS resistor blocks, SPICE resistors can model temperature effects and semiconductor resistor geometries.

Linear Resistor

Recommended

```
* Simple temperature-dependent resistor
Rname pos neg resistance <TC1=tc1> <TC2=tc2> <TEMP=temp> <M=m>

* Resistor with a model definition
Rname pos neg modelName <resistance> <L=l> <W=w> <TC1=tc1> <TC2=tc2>␣
↪<TEMP=temp> <M=m>
.MODEL modelName R <NAME=value ...>
```

- The device type `R` must be specified in the model statement.
- The inductance can also be set as the model parameter `RES`. If it's defined in both the model and the instance, the instance value is used.
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

Instance Parameters

Parameter	Description	Default Value
`TC1`	Temperature coefficient for RES (first order correction)	$0\,^{\circ}\mathrm{C}^{-1}$
`TC2`	Temperature coefficient for RES (second order correction)	$0\,^{\circ}\mathrm{C}^{-2}$
`TEMP`	Device temperature	$27\,^{\circ}\mathrm{C}$
`L`	Length of the semiconductor resistor	$0\,\mathrm{m}$
`W`	Width of the semiconductor resistor	`DEFW`
`M`	Number of parallel devices	1

- The default value for `L` is not permitted. Thus, a nonzero value must be set either in the instance statement or as a model parameter.

Model Parameters

Parameter	Description	Default Value
T_ABS	Device temperature	27 °C
RES	Resistance	0 Ω
RSH	Sheet resistance	0 Ω/sq.
DEFW	Default width of the semiconductor resistor	10^{-6} m
NARROW	Narrowing due to the etching process	0 m

- You can specify the instance parameters TC1, TC2, L, and W as model parameters too. If both are provided, the instance values take priority.
- The instance parameter TEMP overwrites the model parameter T_ABS.
- When the resistance is specified either in the instance statement or through the parameter RES, the parameters L, W, RSH, DEFW, and NARROW are ignored.
- Both RES and RSH are temperature-corrected using TC1 and TC2.
- The default value for RSH is not permitted and a nonzero value must be chosen.

Variable Resistor

Recommended

```
Rname pos neg R={function body}
```

> **Warning**
>
> The variable resistor is implemented as a current source with a user-defined function. Hence, the resistance function must never evaluate to zero.

Examples

```
Rg 1 2 4.7 TC1=0.001 TEMP=90

Vctrl ctrl 0 PULSE 0 1 0 1e-6 1e-6 0.5 1
Rload out 0 R={10+5*V(ctrl)}
```

(S) Voltage Controlled Switch Statement

The voltage-controlled switch monitors a voltage difference to control its switch resistance. Contrary to PLECS, it is not possible to simulate ideal switches (zero on-resistance, infinite off-resistance) in SPICE.

```
Sname pos neg posMeasure negMeasure modelName
.MODEL modelName SW <NAME=value ...>
```

- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

PLECS Spice supports two types of voltage-controlled switch models:

- Models without hysteresis and continuous resistance transitions
- Models with hysteresis and discontinuous resistance transitions

Model Parameters

Continuous resistance model

Parameter	Value	Default Value
`Von`	Control value at which the switch is fully closed	$1\,\mathrm{V}$
`Voff`	Control value at which the switch is fully opened	$0\,\mathrm{V}$
`Ron`	Resistance when the switch is closed	$1\,\Omega$
`Roff`	Resistance when the switch is open	$10^6\,\Omega$

- The switch resistance transitions smoothly between R_{off} and R_{on} over the control voltage range $[V_{\mathrm{off}}, V_{\mathrm{on}}]$.

Hysteresis model

Parameter	Value	Default Value
`Vt`	Threshold voltage	$0.5\,\mathrm{V}$
`Vh`	Hysteresis voltage	$1\,\mathrm{V}$
`Ron`	Resistance when the switch is closed	$1\,\Omega$
`Roff`	Resistance when the switch is open	$10^6\,\Omega$

- The switch changes its internal state from off to on (closes) when the control voltage V_c reaches $V_\mathrm{t} + |V_\mathrm{h}|$ and opens again at $V_\mathrm{c} = V_\mathrm{t} - |V_\mathrm{h}|$.

Model selection

- The continuous resistance model is automatically used when either `Von` or `Voff` is given and all parameters related to hysteresis are ignored.
- When using the hysteresis model, it is recommended to specify both `Vt` and `Vh` to avoid compatibility issues with other SPICE solvers.

Examples

```
S1 drain source gate source SW_model
.MODEL SW_model SW Von=5 Voff=0.5 Ron=0.01 Roff=1Meg
```

(V) Voltage Source Statement

Define a voltage source either as constant DC source or by using specific parametrized functions. If you want to use a different source function, you can use “(B) Function-Based Source Statement” (p. 435).

```
Vname pos neg <DC> dcvalue
Vname pos neg [function]
```

See “Source Statement Functions” (p. 442) for `[function]` syntax. If no function is specified, the source is treated as DC source with `dcvalue=0`.

Note

- We do not support AC-related arguments in sources. Please use PLECS blocks for AC analysis.
- We only support sources that provide either a `dcvalue` or a `[function]`. Some SPICE version allow to provide both simultaneously, which is equivalent to putting two sources in series.

Examples

```
V1 n 0 1
V2 n 0 SIN 0 1 50
```

(W) Current Controlled Switch Statement

The current-controlled switch monitors the current through another circuit element to control its switch resistance. Contrary to PLECS, it is not possible to simulate ideal switches (zero on-resistance, infinite off-resistance) in SPICE.

```
Wname pos neg Vname modelName
.MODEL modelName CSW <NAME=value ...>
```

- The above statement probes the current of a device with the name `Vname`. Allowed device types are voltage sources (`V`), controlled sources (`E`, `H`), and function-based voltage sources (`B`).
- You can customize this model by providing a list of model parameter (name-value format) at the end of the .MODEL statement (p. 471).

PLECS Spice supports two types of current-controlled switch models:

- Models without hysteresis and continuous resistance transitions
- Models with hysteresis and discontinuous resistance transitions

Model Parameters

Continuous resistance model

Parameter	Value	Default Value
`Ion`	Control value at which the switch is fully closed	$1\,\mathrm{A}$
`Ioff`	Control value at which the switch is fully opened	$0\,\mathrm{A}$
`Ron`	Resistance when the switch is closed	$1\,\Omega$
`Roff`	Resistance when the switch is open	$10^6\,\Omega$

- The switch resistance transitions smoothly between R_{off} and R_{on} over the control voltage range $[I_{\mathrm{off}}, I_{\mathrm{on}}]$.

Hysteresis model

Parameter	Value	Default Value
`It`	Threshold current	$0.5\,\mathrm{A}$
`Ih`	Hysteresis current	$1\,\mathrm{A}$
`Ron`	Resistance when the switch is closed	$1\,\Omega$
`Roff`	Resistance when the switch is open	$10^6\,\Omega$

- The switch changes its internal state from off to on (closes) when the control current I_{c} reaches $I_{\mathrm{t}} + |I_{\mathrm{h}}|$ and opens again at $I_{\mathrm{c}} = I_{\mathrm{t}} - |I_{\mathrm{h}}|$.

Model selection

- The continuous resistance model is automatically used when either `Ion` or `Ioff` is given and all parameters related to hysteresis are ignored.
- When using the hysteresis model, it is recommended to specify both `It` and `Ih` to avoid compatibility issues with other SPICE solvers.

Examples

```
Vsense sense_in sense_out 0

W1 load_pos load_neg Vsense CSW_model
.MODEL CSW_model CSW Ion=10 Ioff=0.1 Ron=0.01 Roff=1Meg
```

(X) Subcircuit Instance Statement

X statements allow you to instantiate subcircuits (p. 473). Be aware of "Scopes and Shadowing" (p. 433).

Recommended

```
Xname n1 <n2 ...> subName <PARAMS: <p1=val1 ...>>
```

- Creates an instance of the subcircuit `subName`.
- The nodes in the X statement are the nodes in the current scope. Inside the instantiated `.SUBCKT` call they correspond to the node names of the `.SUBCKT` statement.
- You can override some or all of the `.SUBCKT` statement parameters values by specifying them after `subName`. Any parameter not overwritten uses the default value of the `.SUBCKT` statement. Any parameter passed to a subcircuit in an X statement immediately evaluates the parameter value before passing it on (see "Subtleties of Shadowing" (p. 435)).

> **Note**
>
> We recommend to always specify all parameters that you want to pass to a subcircuit in both the `.SUBCKT` and the `X` statement.

Examples

```
.SUBCKT main n1 n2 n3
X1 n1 helper1
X2 n1 helper2 PARAMS: a=3
.ENDS

.SUBCKT helper1 m1
* the node called n1 in the main subcircuit is now accessible as m1
.ENDS

.SUBCKT helper2 m1 PARAMS: a=1 b=2
```

(continues on next page)

(continued from previous page)

```
* The instance created by X2 sets a to 3 instead of 1.
R1 m1 0 {a}
R2 m1 0 {b}
.ENDS
```

.FUNC Statement

Functions allow you to define reusable mathematical expressions. Variables in function definitions are replaced by values when evaluated.

Recommended

```
.FUNC fname(<x1<,x2<,x3...>>>)={ function body }
```

Where:

- `fname` - Function name used for calling this function
- `x1, x2, x3, ...` - Variables (up to 15) in parentheses
- `{ function body }` - Expression (p. 430) in curly braces that can contain the variables

Functions and Parameters (p. 472) should not depend on themselves in infinite loops. For clarity, we recommend to name nodes, parameters and functions differently. The names of "Predefined Parameters and Functions" (p. 430) must not be used as parameter name.

Examples

```
.FUNC f(x,y)={sin(x+y)}
.FUNC g(x,y,z)={f(x,y)+z}
.PARAM a=7
.FUNC h()={6*a}
R1 n 0 {f(1,2)} ; sin(1+2)
R2 n 0 {g(3,4,5)+h()} ; sin(3+4)+5+6*7
```

.IC Statement

Recommended

`.IC` statements can be used to specify initial conditions for certain capacitors and inductors. This is purely for supporting legacy syntax, as initial conditions can be directly specified in the "(C) Capacitor Statement" (p. 436) or "(L) Inductor Statement" (p. 449).

.MODEL Statement

`.MODEL` is used to access provided "Compact Models" (p. 396) and to set specific parameters.

Recommended

```
.MODEL modelname modeltype <p1=val1 <p2=val2 <...>>>
```

Where:

- `modelname` - A unique name for this model
- `modeltype` - The type of device (see Table 14.8)
- `p1=val1, p2=val2, ...` - Device-specific parameters. Values can be "Constants" (p. 429) or "Expressions" (p. 430) in curly braces.

`.MODEL` does not create any instances of circuit elements. They need to be referenced in their corresponding statements ("(D) Diode Statement" (p. 439), "(M) MOSFET Statement" (p. 451), . . .). Refer to these statements to find a list of all model parameters available for a particular model type.

The underlying compact models typically depend on "Temperature (.TEMP)" (p. 415).

Table 14.8: Supported .MODEL Types

`modeltype`	Description
`D`	Diode
`NMOS`	N-type MOSFET
`PMOS`	P-type MOSFET
`NPN`	NPN BJT
`PNP`	PNP BJT
`NJF`	N-type JFET
`PJF`	P-type JFET
`SW` or `VSwitch`	Voltage-controlled switch
`CSW` or `ISwitch`	Current-controlled switch
`R` or `RES`	Semiconductor resistor
`C`	Semiconductor capacitor

Examples

```
.MODEL mosfet NMOS
.MODEL diode D IS=1e-15 RS=0.1 CJO=1e-11 TT={5*1e-9}
```

.PARAM Statement

Parameters are named values that you can use throughout your netlist. They make your circuits more readable and easier to modify by allowing you to define values in one place and reference them everywhere.

Recommended

```
.PARAM p1=val1 <p2=val2 <...>>
```

Where:

- `p1, p2, ...` - Parameter names
- `val1, val2, ...` - Parameter values (“Constants” (p. 429) or “Expressions” (p. 430) in curly braces)

Parameters and Functions (p. 470) should not depend on themselves in infinite loops. For clarity, we recommend to name nodes, parameters and functions differently. The names of “Predefined Parameters and Functions” (p. 430) must not be used as function name.

Parameters defined after `PARAMS:` in the current Subcircuit or the X Statement (p. 469) creating the instance of the current subcircuit are treated like parameters defined via `.PARAM`. In particular, they can not be redefined.

> **Note**
>
> Defining a parameter, e.g. `.PARAM IS=1e-9`, does not automatically set the device-specific parameter `IS` for .MODEL statements to `1e-9`. The parameter needs to be passed explicitly after the desired `.MODEL` via `IS={IS}`.

Examples

```
.PARAM a=1 b={2*3}
.PARAM c={a+b}
.FUNC f(x)={2*x}
.PARAM d={f(5)}
R1 n 0 {c} ; 1+2*3
R2 n 0 {d} ; 2*5
```

.SUBCKT Statement

Subcircuits allow you to define reusable circuit blocks with their own local nodes and parameters. They can be instantiated via "(X) Subcircuit Instance Statement" (p. 469) and are subject to "Scopes and Shadowing" (p. 433).

Recommended

```
.SUBCKT name <n1 <n2 ...>> <PARAMS: <p1=v1 <p2=v2...>>>
*...
.ENDS
```

Where:

- `name` - Unique name for the subcircuit
- `n1, n2, ...` - Distinct node names (ground `0` is not allowed)
- `p1=v1, p2=v2, ...` - Optional parameters with default values ("Constants" (p. 429) or "Expressions" (p. 430) in curly braces)

Examples

```
.SUBCKT sub1 n1 n2
R n1 n2 1
.ENDS

.SUBCKT sub2 n1 n2 n3 PARAMS: a=1 b={2+3}
R1 n1 n2 1
R2 n2 n3 1
.ENDS
```

15

Components by Category

This chapter lists the blocks of the Component library by category.

System

Configurable Subsystem	Provide subsystem with exchangeable implementations
Display	Display signal values in the schematic
Dynamic Signal Selector	Select or reorder elements from vectorized signal depending on control signal
Enable	Control execution of an atomic subsystem
From File	Read time and signal values from file
Model Reference	Reference a subsystem or netlist from the same or another model
Pause / Stop	Pause or stop the simulation
Scalar Expander	Extend the input signal to the width of the reference
Scope	Display simulation results versus time
Signal Demultiplexer	Split vectorized signal
Signal From	Reference signal from Signal Goto block by name
Signal Goto	Make signal available by name
Signal Inport	Add signal input connector to subsystem
Signal Multiplexer	Combine several signals into vectorized signal
Signal Outport	Add signal output connector to subsystem
Signal Selector	Select or reorder elements from vectorized signal
Subsystem	Create functional entity in hierarchical simulation model
Switch Loss Calculator	Calculate the average sum of the switch losses of all probed components over the specified averaging period
Task Frame	Associate the enclosed components with a task in a multi-tasking environment

continues on next page

Table 15.1 – continued from previous page

Task Transition	Transfer data between tasks using a double buffer
To File	Write time and signal values to file
Trigger	Control execution of an atomic subsystem
XY Plot	Display correlation between two signals

Assertions

Assert Dynamic Lower Limit	Check whether a signal stays above another signal
Assert Dynamic Range	Check whether a signal stays between two other signals
Assert Dynamic Upper Limit	Check whether a signal stays below another signal
Assertion	Check whether a condition is true
Assert Lower Limit	Check whether a signal stays above a constant
Assert Range	Check whether a signal stays within a constant range
Assert Upper Limit	Check whether a signal stays below a constant

Control

Sources

Clock	Provide current simulation time
Constant	Generate constant signal
Initial Condition	Output specified initial value in the first simulation step
Pulse Generator	Generate periodic rectangular pulses
Ramp	Generate constantly rising or falling signal
Random Numbers	Generate uniformly distributed random numbers
Sine Wave	Generate time-based sine wave with optional bias
Step	Generate constant signal with instantaneous step change
Triangular Wave Generator	Generate periodic triangular or sawtooth waveform
White Noise	Generate normally distributed random numbers

Math

Abs	Calculate absolute value of input signal
Algebraic Constraint	Enforce an algebraic constraint
Data Type	Cast the input signal to the specified data type
Gain	Multiply input signal by constant
Math Function	Apply specified mathematical function
Minimum / Maximum	Output input signal with highest resp. lowest value
Offset	Add constant to input signal

continues on next page

Table 15.4 – continued from previous page

Product	Multiply and divide scalar or vectorized input signals
Rounding	Round floating point signal to integer values
Signum	Provide sign of input signal
Sum	Add and subtract input signals
Trigonometric Function	Apply specified trigonometric function

Continuous

Continuous PID Controller	Implementation of a continuous-time controller (P, I, PI, PD or PID)
Integrator	Integrate input signal with respect to time
PLL (Single-Phase)	Implementation of a single phase PLL
PLL (Three-Phase)	Implementation of a three phase PLL
State Space	Implement linear time-invariant system as state-space model
Transfer Function	Model linear time-invariant system as transfer function

Delays

Memory	Provide input signal from previous major time step
Pulse Delay	Delay discrete-value input signal by fixed time
Transport Delay	Delay continuous input signal by fixed time
Turn-on Delay	Delay rising flank of input pulses by fixed dead time

Discontinuous

Comparator	Compare two input signals with minimal hysteresis
Dead Zone	Output zero while input signal is within dead zone limits
Hit Crossing	Detect when signal reaches or crosses given value
Manual Signal Switch	Manually select one of two input signals
Multiport Signal Switch	Select one of multiple input signals depending on control signal
Quantizer	Apply uniform quantization to input signal
Rate Limiter	Limit rising and falling rate of change
Relay	Toggle between on- and off-state with configurable threshold
Saturation	Limit input signal to upper and/or lower value
Signal Switch	Select one of two input signals depending on control signal

Discrete

Delay	Delay input signal by given number of samples
Discrete Integrator	Calculate discrete integral of input signal
Discrete Mean Value	Calculate running mean value of input signal
Discrete PID Controller	Implementation of a discrete-time controller (P, I, PI, PD or PID)
Discrete State Space	Implement discrete time-invariant system as state-space model
Discrete Transfer Function	Model discrete system as transfer function
Zero Order Hold	Sample and hold input signal periodically

Filters

Fourier Transform	Perform Fourier transform on input signal
Moving Average	Continuously average input signal over specified time period
Periodic Average	Periodically average input signal over specified time
Periodic Impulse Average	Periodically average Dirac impulses over specified time
RMS Value	Calculate root mean square (RMS) value of input signal
Total Harmonic Distortion	Calculate total harmonic distortion (THD) of input signal

Functions & Tables

1D Look-Up Table	Compute piece-wise linear function of one input signal
2D Look-Up Table	Compute piece-wise linear function of two input signals
3D Look-Up Table	Compute piece-wise linear function of three input signals
C-Script	Execute custom C code
DLL	Interface with externally generated dynamic-link library
FMU	Use a model stored in an FMU model
Fourier Series	Synthesize periodic output signal from Fourier coefficients
Function	Apply arbitrary arithmetic expression to scalar or vectorized input signal

Logical

Combinatorial Logic	Use binary input signals to select one row from truth table
Compare to Constant	Compare input signal to constant threshold
D Flip-flop	Implement edge-triggered flip-flop
Edge Detection	Detect edges of pulse signal in given direction
JK Flip-flop	Implement edge-triggered JK flip-flop
Logical Operator	Combine input signals logically
Monoflop	Generate pulse of specified width when triggered
Relational Operator	Compare two input signals
SR Flip-flop	Implement set-reset flip-flop

Modulators

2-Pulse Generator	Generate firing pulses for H-bridge thyristor rectifier
3-Phase Index-Based Modulation	Generate the modulation index for a three-phase reference voltage
3-Phase Overmodulation	Extend linear range of modulation index for 3-phase inverters
6-Pulse Generator	Generate firing pulses for 3-phase thyristor rectifier
Blanking Time	Generate commutation delay for 2-level inverter bridges
Blanking Time (3-Level)	Generate commutation delay for 3-level inverter bridges
Peak Current Controller	Implement peak current mode control
Sawtooth PWM	Generate PWM signal using sawtooth carrier
Sawtooth PWM (3-Level)	Generate 3-level PWM signal using sawtooth carriers

continues on next page

Table 15.12 – continued from previous page

Space Vector PWM	Generate PWM signals for 3-phase inverter using space-vector modulation
Space Vector PWM (3-Level)	Generate PWM signals for 3-phase NPC inverter using space-vector modulation
Symmetrical PWM	Generate PWM signal using symmetrical triangular carrier
Symmetrical PWM (3-Level)	Generate 3-level PWM signal using symmetrical triangular carriers
Variable Frequency PWM	Generate PWM signals with variable frequency
Variable Phase PWM	Generate PWM signals with variable phase shift

Transformations

Polar to Rectangular	Convert polar coordinates to Cartesian coordinates
Rectangular to Polar	Convert Cartesian coordinates to polar coordinates
Transformation 3ph->RRF	Transform 3-phase signal to rotating reference frame
Transformation 3ph->SRF	Transform 3-phase signal to stationary reference frame
Transformation RRF->3ph	Transform vector in rotating reference frame into 3-phase signal
Transformation RRF->SRF	Transform vector from rotating to stationary reference frame
Transformation SRF->3ph	Transform vector in stationary reference frame into 3-phase signal
Transformation SRF->RRF	Transform vector from stationary to rotating reference frame

State Machine

State Machine	Model a state machine

Small Signal Analysis

(PLECS Standalone only)

Small Signal Gain	Measure loop gain of closed control loop using small-signal analysis
Small Signal Perturbation	Generate perturbation signal for small-signal analysis
Small Signal Response	Measure system response for small-signal analysis

Electrical

Connectivity

Electrical Ground	Connect to common electrical ground
Electrical Label	Connect electrical potentials by name
Electrical Port	Add electrical connector to subsystem
Wire Multiplexer	Bundle several wires into bus
Wire Selector	Select or reorder elements from wire bus

Sources

Current Source (Controlled)	Generate variable current
Current Source AC	Generate sinusoidal current
Current Source DC	Generate constant current
Voltage Source (Controlled)	Generate variable voltage
Voltage Source AC	Generate sinusoidal voltage
Voltage Source AC (3-Phase)	Generate 3-phase sinusoidal voltage
Voltage Source DC	Generate constant voltage

Meters

Ammeter	Output measured current as signal
Meter (3-Phase)	Measure voltages and currents of 3-phase system
Voltmeter	Output measured voltage as signal

Passive Components

Capacitor	Ideal capacitor
Electrical Algebraic Component	Enforce an algebraic constraint in terms of voltage and current
Inductor	Ideal inductor
Mutual Inductor	Ideal mutual inductor
Pi-Section Line	Single-phase pi-section transmission line
Piece-wise Linear Resistor	Resistance defined by voltage-current pairs
Resistor	Ideal resistor
Saturable Capacitor	Capacitor with piece-wise linear saturation
Saturable Inductor	Inductor with piece-wise linear saturation
Transmission Line (3ph)	3-phase transmission line
Variable Capacitor	Capacitance controlled by signal
Variable Inductor	Inductance controlled by signal
Variable Resistor	Resistance controlled by signal
Variable Resistor with Constant Capacitor	Controlled resistance in parallel with constant capacitance
Variable Resistor with Constant Inductor	Controlled resistance in series with constant inductance
Variable Resistor with Variable Capacitor	Controlled resistance in parallel with controlled capacitance
Variable Resistor with Variable Inductor	Controlled resistance in series with controlled inductance

Power Semiconductors

Diode	Ideal diode with optional forward voltage and on-resistance

continues on next page

Table 15.20 – continued from previous page

Diode with Reverse Recovery	Dynamic diode model with reverse recovery
GTO	Ideal GTO with optional forward voltage and on-resistance
GTO (Reverse Conducting)	Ideal GTO with ideal anti-parallel diode
IGBT	Ideal IGBT with optional forward voltage and on-resistance
IGBT with Diode	Ideal IGBT with ideal anti-parallel diode
IGBT with Limited di/dt	Dynamic IGBT model with finite current slopes during turn-on and turn-off
IGCT (Reverse Blocking)	Ideal IGCT with optional forward voltage and on-resistance
IGCT (Reverse Conducting)	Ideal IGCT with ideal anti-parallel diode
MOSFET	Ideal MOSFET with optional on-resistance
MOSFET with Diode	Ideal MOSFET with ideal anti-parallel diode
MOSFET with Limited di/dt	Dynamic MOSFET model with finite current slopes during turn-on and turn-off
Thyristor	Ideal thyristor (SCR) with optional forward voltage and on-resistance
Thyristor with Reverse Recovery	Dynamic thyristor (SCR) model with reverse recovery
TRIAC	Ideal TRIAC with optional forward voltage and on-resistance
Zener Diode	Zener diode with controlled reverse breakdown voltage

Power Modules

Power Modules provide compact building blocks for switched-mode power converters and feature a “Sub-cycle Averaging” implementation.

3-Level Half Bridge (ANPC)	Single leg of a 3-level active neutral-point clamped half-bridge converter
3-Level Half Bridge (NPC)	Single leg of a 3-level neutral-point clamped voltage source inverter
3-Level Half Bridge (T-Type)	Single leg of a 3-level T-type half-bridge converter
3-Phase Current Source Inverter	3-phase 2-level current source inverter
3-Phase Voltage Source Inverter	3-phase 2-level voltage source inverter
5-Level Half Bridge (ANPC)	Single leg of a 5-level active neutral-point clamped half-bridge converter
Chopper (High-Side Switch)	Chopper used in buck converters
Chopper (High-Side Switch with Reverse Diode)	Chopper used in buck converters
Chopper (Low-Side Switch)	Chopper used in boost converters
Chopper (Low-Side Switch with Reverse Diode)	Chopper used in boost converters
Diode Pair	Diode rectifier module
Flying Capacitor Half Bridge	Multi-level inverter half bridge with flying capacitors
Full Bridges (Series Connected)	Series-connected inverter cells for modular multilevel converters
Half Bridge	Single leg of a 2-level voltage source inverter
Half Bridges (Low-/High-Side Connected)	Series-connected inverter cells for modular multilevel converters
Thyristor Pair	Thyristor rectifier/inverter module

Sub-cycle Averaging

> **Note**
>
> Sub-cycle averaging is available for all components in the “Power Modules” (p. 488) library section and for selected components in the “Nanostep” (p. 493) library section.

Sub-cycle averaging is an established modeling approach for power electronic converters that enables an accurate representation of switching behavior while maintaining computational efficiency. This is particularly relevant for C code generation (p. 341) and real-time simulations (p. 360). Sub-cycle averaging consists of two main ingredients:

- An averaged duty-cycle signal with values between 0 and 1, obtained by averaging the input switching signal over an interval equal to the model discretization step size. In real-time simulations, these average values can be obtained by sampling PWM signals at each FPGA clock cycle of the real-time simulator, which is typically in the range of a few nanoseconds. Even with high switching frequencies and simulation steps much larger than the FPGA clock cycle, the resulting phase current remains accurate because the applied volt-seconds of the averaged signals correspond to the volt-seconds of the original PWM signals.
- A converter Power Module consisting of controlled current and voltage sources and a diode pair. The controlled sources are driven by the averaged duty-cycle signal, and the diode pair is required to simulate natural commutation and discontinuous conduction mode. Depending on the direction of the phase current, either the lower or the upper diode conducts, applying the corresponding phase voltage and DC current.

The main limitation of sub-cycle averaging is the inaccuracy caused by zero crossings of the phase current. As the direction of the phase current is determined only once per simulation step, slightly incorrect volt-seconds may be applied during the step in which the current crosses zero or enters DCM.

References

J. Allmeling and N. Felderer, “Sub-Cycle Average Models with Integrated Diodes for Real-Time Simulation of Power Converters”, IEEE Southern Power Electronics Conference (SPEC), 2017

J. Allmeling, N. Felderer and M. Luo, “High Fidelity Real-Time Simulation of Multi-Level Converters”, International Power Electronics Conference (IPEC-Niigata 2018 - ECCE Asia), 2018

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
Mutual Inductance (2 Windings)	Magnetic coupling between two lossy windings
Mutual Inductance (3 Windings)	Magnetic coupling between three lossy windings
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 (Open Stator Windings)	Non-saturable induction machine with squirrel-cage rotor and open stator windings
Induction Machine (Slip Ring)	Non-saturable induction machine with slip-ring rotor
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
Permanent Magnet Synchronous Machine (Open Winding)	Synchronous machine excited by permanent magnets with open stator windings
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

Diode Rectifier (3ph)	3-phase diode rectifier

continues on next page

Table 15.25 – continued from previous page

Ideal 3-Level Converter (3ph)	Switch-based 3-phase 3-level converter
Ideal Converter (3ph)	Switch-based 3-phase converter
IGBT 3-Level Converter (3ph)	3-phase 3-level neutral-point clamped IGBT converter
IGBT Converter (3ph)	3-phase IGBT converter
MOSFET Converter (3ph)	3-phase MOSFET converter
Thyristor Rectifier/Inverter	3-phase thyristor rectifier/inverter

Nanostep

↪ *Added in PLECS 4.9.*

For general information on Nanostep components, see “Nanostep Description” (p. 495).

3-Phase Dual Active Bridge	Simple 3-phase dual active bridge converter without magnetizing inductance
3-Phase NPC Voltage Source Inverter	3-phase neutral-point clamped voltage source inverter with filter inductors
3-Phase T-Type Voltage Source Inverter	3-phase T-type voltage source inverter with filter inductors
3-Phase Voltage Source Inverter	3-phase voltage source inverter with filter inductors
4-Phase Voltage Source Inverter	4-phase voltage source inverter with filter inductors
Bidirectional Phase-Shifted Full-Bridge Converter	Bidirectional phase-shifted full-bridge converter with optional resonant capacitor
Boost Converter	Boost converter with filter inductance
Boost PFC Converter	Rectifier with boost converter for power factor correction
Boost PFC Converter with LC Filter	Rectifier with boost converter and LC input filter for power factor correction

continues on next page

Table 15.26 – continued from previous page

Buck Converter	Buck converter with filter inductance
CLLLC Converter	Dual active bridge or resonant converter with magnetizing inductance
CLLLC Converter with NPC Half-Bridge	Dual active bridge or resonant converter with a primary-side NPC half-bridge
Flyback Converter with Snubber	Flyback converter with leakage inductance and snubber diode
Frequency-doubling LLC Converter	Frequency-doubling LLC converter
Full-Bridge Inverter	Full-bridge voltage source inverter with filter inductance
Full-Bridge Inverter with LCL Filter	Full-bridge voltage source inverter with LCL filter on the AC side
Full-Bridge LLC Converter	Full-bridge LLC resonant converter
Full-Bridge LLC Converter with Synchronous Rectification	Full-bridge LLC resonant converter with synchronous rectification
Half-Bridge Inverter	Half-bridge voltage source inverter with filter inductance
Half-Bridge LLC Converter	Half-bridge LLC resonant converter
Half-Bridge LLC Converter with Synchronous Rectification	Half-bridge LLC resonant converter with synchronous rectification
Matrix Bidirectional Phase-Shifted Full-Bridge Converter	Bidirectional phase-shifted full-bridge converter with 3-phase matrix converter frontend
Matrix CLLLC Converter	CLLLC converter with 1/2/3/4-phase matrix converter frontend
Matrix Phase-Shifted Full-Bridge Converter	Phase-shifted full-bridge converter with 3-phase matrix converter frontend
Phase-Shifted Full-Bridge Converter	Phase-shifted full-bridge converter with optional resonant capacitor

continues on next page

Table 15.26 – continued from previous page

Simple Dual Active Bridge	Dual active bridge converter without magnetizing inductance
Triple Active Bridge	Simple triple active bridge converter without magnetizing inductance
Vienna Rectifier	3-phase Vienna rectifier with filter inductors

Nanostep Description

Nanostep converter modules are primarily intended for real-time simulations (p. 360) on the RT Box (p. 361). The Nanostep solver (p. 363) incorporates the switching network and energy storage elements of the converter. Gate signal sampling, inductor current integration, and current zero-crossing detection occur at the FPGA clock cycle, typically in the range of a few nanoseconds. This makes the Nanostep solver suitable for converters in which the current direction changes frequently, especially resonant DC/DC converters and high-frequency converter topologies.

Some Nanostep components also provide a `Sub-cycle average` configuration, as described in "Sub-cycle Averaging" (p. 489). In this configuration, generated code for real-time simulation on the RT Box runs on the CPU (p. 362) or FlexArray (p. 363) solver.

Electronics

Op-Amp	Ideal operational amplifier with finite gain
Op-Amp with Limited Output	Ideal operational amplifier with limited output voltage

Model Settings

Electrical Model Settings	Configure settings for an individual electrical circuit

SPICE Components

↪ *Added in PLECS 5.0.*

Netlist	Netlist reader for importing a SPICE netlist defining a subcircuit
Netlist File Reference	Variant of the netlist reader for importing a SPICE netlist at runtime from a file

Thermal

Connectivity

Ambient Temperature	Connect to Heat Sink on which subsystem is placed
Thermal Ground	Connect to common reference temperature
Thermal Multiplexer	Combine several connections into one vector
Thermal Port	Add thermal connector to subsystem
Thermal Selector	Select or reorder elements from vector connection

Sources

Constant Heat Flow	Generate constant heat flow
Constant Temperature	Provide constant temperature
Controlled Heat Flow	Generate variable heat flow
Controlled Temperature	Provide variable temperature

Meters

Heat Flow Meter	Output measured heat flow as signal
Thermometer	Output measured temperature as signal

Components

Heat Sink	Isotherm environment for placing components
Thermal Capacitor	Thermal capacitance of piece of material
Thermal Chain	Thermal impedance implemented as RC chain

continues on next page

Table 15.33 – continued from previous page

Thermal Package Impedance	Model thermal coupling in a semiconductor package
Thermal Resistor	Thermal resistance of piece of material

Model Settings

Thermal Model Settings	Configure settings for an individual thermal system

Magnetic

Connectivity

Magnetic Multiplexer	Combine several connections into one vector
Magnetic Port	Add magnetic connector to subsystem
Magnetic Selector	Select or reorder elements from vector connection

Sources

MMF Source (Constant)	Generate a constant magneto-motive force
MMF Source (Controlled)	Generate a variable magneto-motive force

Meters

Flux Rate Meter	Output the measured rate-of-change of magnetic flux
MMF Meter	Output the measured magneto-motive force

Components

Air Gap	Air gap in a magnetic core
Hysteretic Core	Magnetic core element with static hysteresis
Leakage Flux Path	Permeance of linear leakage flux path
Linear Core	Linear magnetic core element
Magnetic Permeance	Linear magnetic permeance
Magnetic Resistance	Effective magnetic resistance for modeling losses
Saturable Core	Magnetic core element with saturation

continues on next page

Table 15.38 – continued from previous page

Variable Magnetic Permeance	Variable permeance controlled by external signal
Winding	Ideal winding defining an electro-magnetic interface

Mechanical

Translational

Connectivity

Translational Multiplexer	Combine several connections into one vector
Translational Port	Add translational connector to subsystem
Translational Reference	Connect to translational reference frame
Translational Selector	Select or reorder elements from vector connection

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
Position Sensor	Output measured absolute or relative position as signal
Translational Speed Sensor	Output measured linear speed as signal

Components

Mass	Model sliding mass with inertia

continues on next page

Table 15.42 – continued from previous page

Rack and Pinion	Ideal conversion between translational and rotational motion
Translational Algebraic Component	Define an algebraic constraint in terms of force and speed
Translational Backlash	Ideal translational backlash
Translational Clutch	Ideal translational clutch
Translational Damper	Ideal viscous translational damper
Translational Friction	Ideal translational stick/slip friction
Translational Friction (Controlled)	Generate variable translational friction
Translational Hard Stop	Ideal translational single- or double-sided hard stop
Translational Spring	Ideal translational spring

Model Settings

Translational Model Settings	Configure settings for an individual mechanical system

Rotational

Connectivity

Rotational Multiplexer	Combine several connections into one vector
Rotational Port	Add rotational connector to subsystem
Rotational Reference	Connect to rotational reference frame
Rotational Selector	Select or reorder elements from vector connection

Sources

Rotational Speed (Constant)	Maintain constant angular speed
Rotational Speed (Controlled)	Maintain variable angular speed
Torque (Constant)	Generate constant torque
Torque (Controlled)	Generate variable torque

Sensors

Angle Sensor	Output measured absolute or relative angle as signal
Rotational Speed Sensor	Output measured angular speed as signal
Torque Sensor	Output measured torque as signal

Components

Gear	Ideal gear
Inertia	Model rotating body with inertia
Planetary Gear Set	Ideal planetary gear set
Rack and Pinion	Ideal conversion between translational and rotational motion
Rotational Algebraic Component	Define an algebraic constraint in terms of torque and angular speed
Rotational Backlash	Ideal rotational backlash
Rotational Clutch	Ideal rotational clutch
Rotational Damper	Ideal viscous rotational damper
Rotational Friction	Ideal rotational stick/slip friction
Rotational Friction (Controlled)	Generate variable rotational friction

continues on next page

Table 15.47 – continued from previous page

Rotational Hard Stop	Ideal rotational single- or double-sided hard stop
Torsion Spring	Ideal torsion spring

Model Settings

Rotational Model Settings	Configure settings for an individual mechanical system

Additional Simulink Blocks

(PLECS Blockset only)

AC Sweep	Perform AC sweep
Impulse Response Analysis	Perform impulse response analysis
Loop Gain Analysis (AC Sweep)	Determine loop gain of closed control loop
Loop Gain Analysis (Multi-tone)	Determine loop gain of closed control loop
Multitone Analysis	Perform a multitone analysis
Steady-State Analysis	Determine periodic steady-state operating point
Timer	Generate piece-wise constant signal

See also “Simulink Blocks” (p. 1135).

16

Component Reference

This chapter lists the contents of the Component library in alphabetical order.

1D Look-Up Table

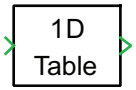
Purpose

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 (p. 511) to map two input signals to an output signal.

Example Model

- See the example model **"Lookup Table"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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.

2-Pulse Generator

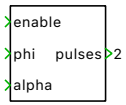
Purpose

Generate firing pulses for H-bridge thyristor rectifier

Library

Control / Modulators

Description



This block generates the pulses used to fire the thyristors of an H-bridge rectifier. The inputs of the block are a logical enable signal, a ramp signal φ (produced e.g. by a PLL), and the firing angle α.

2D Look-Up Table

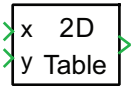
Purpose

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 (p. 508).

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.

3-Level Half Bridge (ANPC)

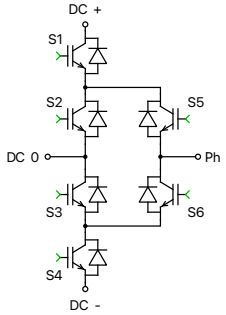
Purpose

3-level active neutral-point clamped half-bridge converter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a single leg of a 3-level active neutral-point clamped voltage source converter. It contains six switches S1 to S6 as indicated in the figure.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if any condition in the following table is met:

Gate signal condition	Shoot-through path	Conducting devices
$S_1 + S_2 > 1$	`DC+` to `DCO`	S_1, S_2
$S_3 + S_4 > 1$	`DCO` to `DC-`	S_3, S_4
$S_1 + S_6 > 1$ AND $S_5 + S_6 > 1$	`DC+` to `DCO`	S_1, S_5, S_6, D_3
$S_4 + S_5 > 1$ AND $S_5 + S_6 > 1$	`DCO` to `DC-`	D_2, S_5, S_6, S_4

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

↪ *Added in PLECS 4.4.*

3-Level Half Bridge (NPC)

Purpose

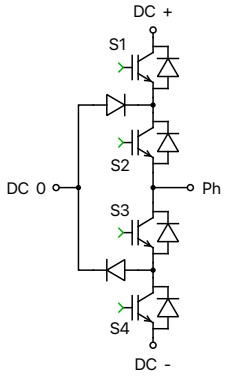
3-level neutral-point clamped half-bridge converter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description

This Power Module implements a single leg of a 3-level neutral-point clamped voltage source inverter.



The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if any condition in the following table is met:

Gate signal condition	Shoot-through path
$S_1 + S_3 > 1$ AND $S_2 + S_3 > 1$	`DC+` to `DCO`
$S_2 + S_3 > 1$ AND $S_2 + S_4 > 1$	`DCO` to `DC-`

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

↪ *Added in PLECS 4.9.*

3-Level Half Bridge (T-Type)

Purpose

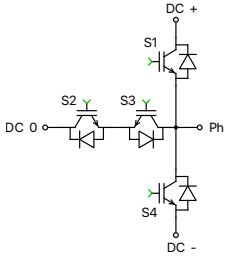
3-level T-type half-bridge converter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description

This Power Module implements a single leg of a 3-level T-Type voltage source converter. It contains 4 switches S1 to S4 as indicated in the figure.



The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Neutral-point connection

Specifies if the neutral-point switches are series connected with `Common collector` (`drain` for MOSFETs) or `Common emitter` (`source` for MOSFETs).

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if any condition in the following table is met:

Gate signal condition	Shoot-through path	Neutral-point connection
$S_1 + S_4 > 1$	`DC+` to `DC-`	Any
$S_1 + S_3 > 1$	`DC+` to `DC0`	`Common emitter`
$S_2 + S_4 > 1$	`DC0` to `DC-`	`Common emitter`
$S_1 + S_2 > 1$	`DC+` to `DC0`	`Common collector`
$S_3 + S_4 > 1$	`DC0` to `DC-`	`Common collector`

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

↪ *Added in PLECS 4.4.*

3-Phase Current Source Inverter

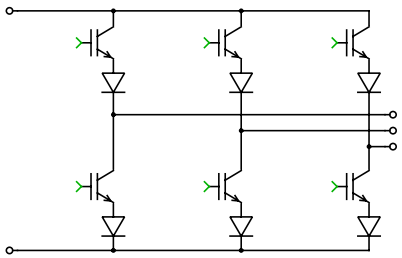
Purpose

3-phase 2-level current source inverter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a 3-phase 2-level current source inverter. It offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the inverter has voltage source behavior and must be connected to positively biased inductors or current sources. The AC side is typically connected to filter capacitors. Only delta-connected filter capacitors are allowed since no neutral point is considered in the model.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations

(p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

The current source inverter model correctly accounts for live times, i.e. intervals during commutation in which at least one upper switch and one lower switch are turned on to maintain a current path for the externally connected DC-side current source.

It also supports rectifier operation, where power flows from the AC side to the DC side. In rectifier mode, the DC side can be modeled with or without a free-wheeling diode, as specified by the **Free-wheeling diode on DC side** parameter.

> **Note**
>
> The sub-cycle average implementation cannot model an open circuit of the DC side. Therefore, in inverter mode, the sum of the control signals for the upper three switches and the lower three switches must be larger than 1 at any point in time. The block will flag an error if this condition is not met. For rectifier mode, please see the description of the **Free-wheeling diode on DC side** parameter below. Also, the applied DC currents must never become negative.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Free-wheeling diode on DC side

This parameter is only applicable for rectifier mode. The option `no` means no extra free-wheeling diode on the DC side is modeled and that the sum of the control signals for the upper three switches and the lower three switches must be larger than 1 at any point in time. Otherwise the block will flag an error. The option `yes` means an extra DC-side free-wheeling diode is included. In this case, no error is flagged when the sum of the control signals for the upper three switches and the lower three switches is smaller than 1, because the DC current can flow through this extra free-wheeling diode.

↪ *Added in PLECS 4.4.*

3-Phase Dual Active Bridge

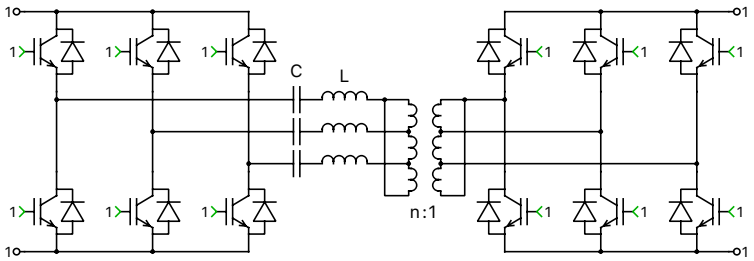
Purpose

3-phase dual active bridge converter without magnetizing inductance, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated three-phase dual active bridge (DAB) converter with optional resonant capacitors.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.1.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Primary inductance
A non-zero scalar or 3-element vector specifying the primary-side inductance L per phase, including the transformer leakage inductances and any external series-connected inductors, in henries (H).

Winding resistance
A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Include primary capacitors
Allows you to include (`yes`) or remove (`no`) resonant capacitors on the primary-side phases.

Resonant capacitance
If the **Include primary capacitors** option is set to `yes`, this parameter requires a non-zero scalar or 3-element vector for the capacitances C, in farads (F).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.1 for the probe signal positions.

Inductor currents
A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltages
A vector containing the voltages across the capacitors C, in volts (V), defined positive at the sides marked with (+).

Nanostep Probe Signals

Inductor currents
A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltages
A vector containing the voltages across the capacitors C, in volts (V), defined positive at the sides marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

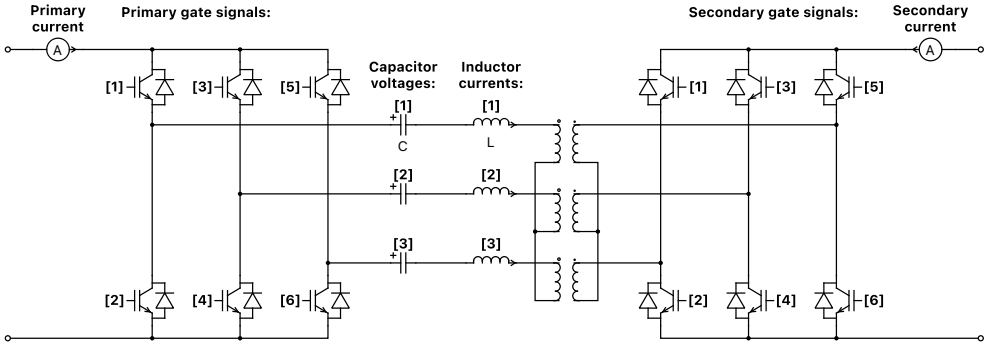


Fig. 16.1: Probes for the 3-Phase Dual Active Bridge

↪ *Added in PLECS 4.9.*

3-Phase Index-Based Modulation

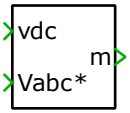
Purpose

Generate the modulation index for a three-phase reference voltage

Library

Control / Modulators

Description



This block generates the modulation index for a three-phase reference voltage defined by the input V^* with the chosen modulation strategy. This block may be used in series with a PWM generation block for the control of three-phase inverters without neutral point connection.

A universal representation of the three-phase modulation index output is as follows (phase $i = \{a, b, c\}$):

$$m_i(t) = m_i^*(t) + m_0(t)$$

where $m_0(t)$ represents the zero-sequence harmonic injected into all three phases, dependent on the selected modulation strategy. $m_i^*(t)$ represents three-phase fundamental sinusoidal modulation indices calculated using the block inputs, i.e. V^* divided by $V_{dc}/2$. It can be written as follows:

$$\begin{aligned} m_a^*(t) &= M \sin(\omega t) \\ m_b^*(t) &= M \sin(\omega t - 2\pi/3) \\ m_c^*(t) &= M \sin(\omega t + 2\pi/3) \end{aligned}$$

M stands for the modulation depth of the three-phase converter, where $M = 1$ defines the limit of the linear modulation range by the `Sinusoidal PWM` modulation strategy (i.e. the formed voltage vector modulus equals to $V_{dc}/2$).

The modulation strategies demonstrated below all showcase the same $M = 1$ for a three-phase reference voltage at $50\,\mathrm{Hz}$ fundamental frequency. The modulation index pattern of only phase A is depicted in each case, i.e. $m_a(t)$.

Sinusoidal PWM

Sinusoidal modulation without adding a zero sequence:

$$m_0(t) = 0$$

Space Vector PWM (Symmetrical)

Symmetrical SVPWM pattern where the injected zero sequence is:

$$m_0(t) = \frac{1}{2}(1 - m_{\max}^*(t)) + \frac{1}{2}(-1 - m_{\min}^*(t))$$

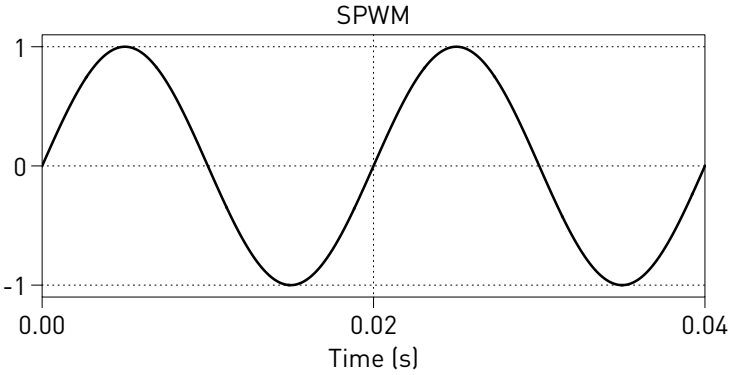


Fig. 16.2: Sinusoidal PWM

Note that $m^*_{\max}(t)$ and $m^*_{\min}(t)$ represent the instantaneous maximum and minimum of the three-phase $m^*_i(t)$ signals respectively.

Space Vector PWM (DPWM1)

For the Discontinuous PWM pattern 1, the one with the largest magnitude between $m^*_{\max}(t)$ and $m^*_{\min}(t)$ determines the injected zero sequence. $m_0(t)$ can be expressed by:

$$m_0(t) = \begin{cases} 1 - m^*_{\max}(t), & \text{if } |m^*_{\max}(t)| \geq |m^*_{\min}(t)| \\ -1 - m^*_{\min}(t), & \text{if } |m^*_{\max}(t)| < |m^*_{\min}(t)| \end{cases}$$

It represents a 60-degree max/min modulation index clamping interval, centered at each half fundamental period.

Space Vector PWM (DPWM0)

For the Discontinuous PWM pattern 0, the zero-sequence signal is generated based on the one in DPWM1. The only difference is that the three-phase $m^*_i(t)$ signals are phase-shifted by 30-degree leading.

Space Vector PWM (DPWM2)

For the Discontinuous PWM pattern 2, the zero-sequence signal is generated based on the one in DPWM1. The only difference is that the three-phase $m^*_i(t)$ signals are phase-shifted by 30-degree lagging.

Space Vector PWM (DPWM3)

For the Discontinuous PWM pattern 3, the one with smallest magnitude between $m^*_{\max}(t)$ and $m^*_{\min}(t)$ determines the injected zero sequence. It

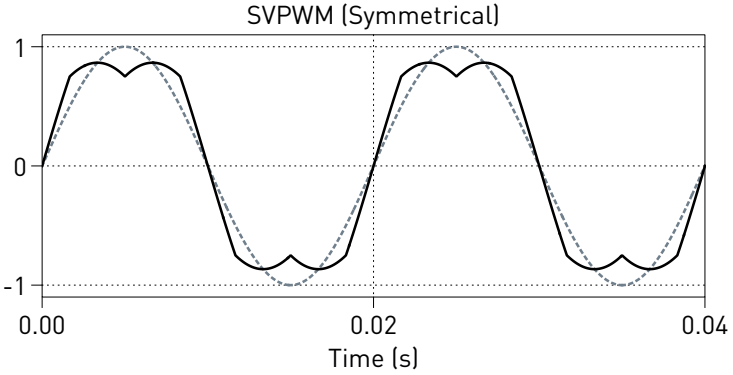


Fig. 16.3: Space Vector PWM (Symmetrical)

represents a 30-degree max/min modulation index clamping interval, centered at every quarter of the fundamental period.

Space Vector PWM (DPWMMIN)
For the Discontinuous PWM MIN pattern, $m^*_{\min}(t)$ determines the injected zero sequence. It clamps the modulation index to -1 for a 120-degree interval, centered inside each negative half fundamental period.

Space Vector PWM (DPWMMAX)
For the Discontinuous PWM MAX pattern, $m^*_{\max}(t)$ determines the injected zero sequence. It clamps the modulation index to 1 for a 120-degree interval, centered inside each positive half fundamental period.

Parameters

Modulation strategy
The modulation strategy can be set to `Sinusoidal PWM`, `Space Vector PWM (Symmetrical)`, `Space Vector PWM (DPWM0)`, `Space Vector PWM (DPWM1)`, `Space Vector PWM (DPWM2)`, `Space Vector PWM (DPWM3)`, `Space Vector PWM (DPWMMIN)`, `Space Vector PWM (DPWMMAX)` using the combo box.

Inputs and Outputs

DC voltage
The input signal $V_{\rm dc}$ is the voltage measured on the dc side of the inverter, specified as a scalar.

Reference voltage
This input, labeled $V^*_{\rm abc}$, is a three-dimensional vector signal comprising the three-phase voltage reference $[V^*_{\rm a}, V^*_{\rm b}, V^*_{\rm c}]$.

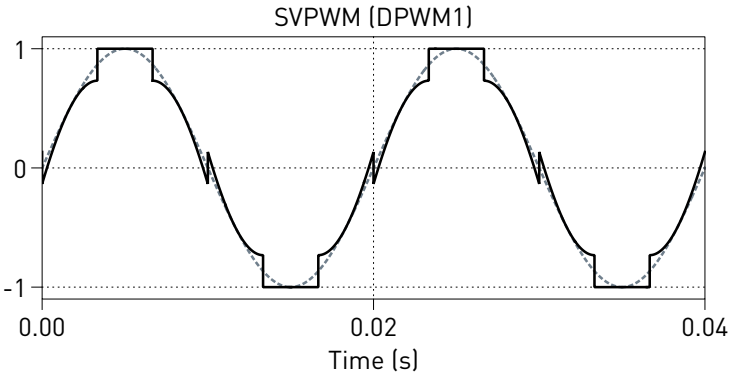


Fig. 16.4: Space Vector PWM (DPWM1)

Modulation index output

The output labeled m is formed from three-phase modulation indices $[m_\mathrm{a}, m_\mathrm{b}, m_\mathrm{c}]$ of the selected modulation strategy, which is also a three-dimensional vector signal. For the linear modulation range of each modulation strategy it varies within $[-1, 1]$. A PWM generator block can be connected in series with this block to generate control signals for the three-phase inverter legs A, B, and C.

Probe Signals

3-phase modulation index

A vector signal consisting of the three-phase modulation indices, $[m_\mathrm{a}, m_\mathrm{b}, m_\mathrm{c}]$ for the selected modulation strategy.

References

- K. Zhou and D. Wang, “Relationship between space-vector modulation and three-phase carrier-based PWM: a comprehensive analysis three-phase inverters,” in IEEE Transactions on Industrial Electronics, vol. 49, no. 1, pp. 186-196, Feb. 2002.
- A. M. Hava, R. J. Kerkman and T. A. Lipo, “Simple analytical and graphical methods for carrier-based PWM-VSI drives,” in IEEE Transactions on Power Electronics, vol. 14, no. 1, pp. 49-61, Jan. 1999.

↪ *Added in PLECS 4.7.*

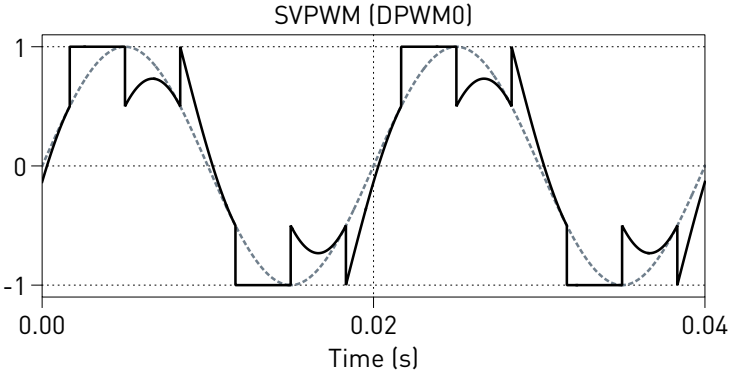


Fig. 16.5: Space Vector PWM (DPWM0)

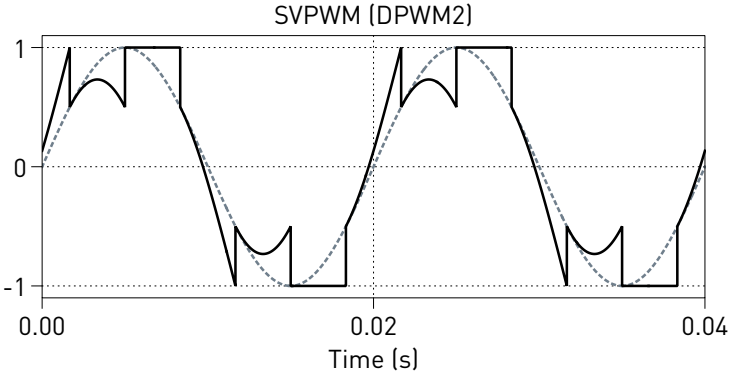


Fig. 16.6: Space Vector PWM (DPWM2)

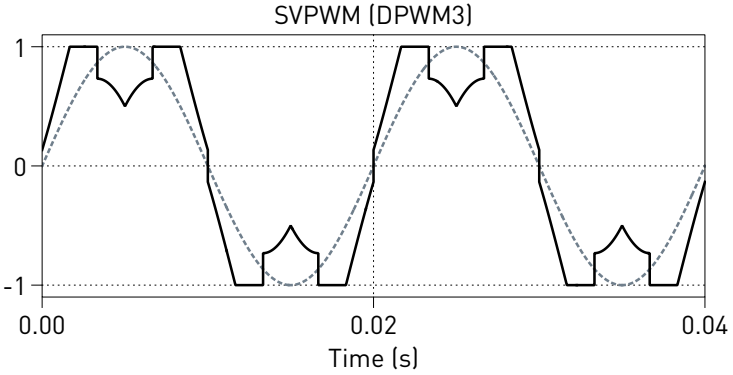


Fig. 16.7: Space Vector PWM (DPWM3)

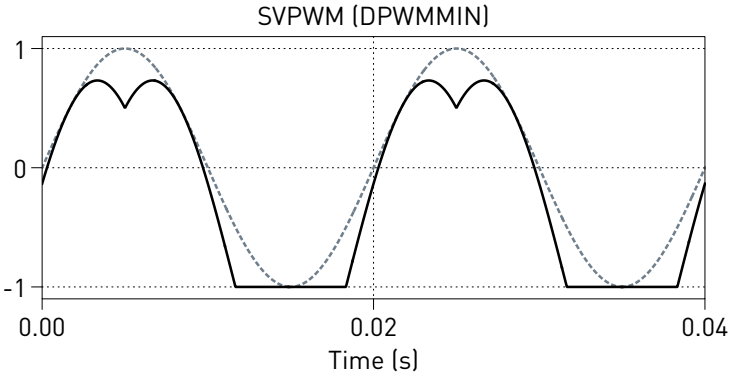


Fig. 16.8: Space Vector PWM (DPWMMIN)

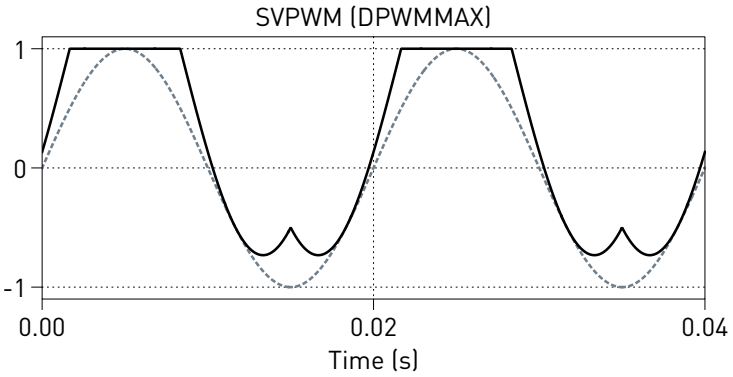


Fig. 16.9: Space Vector PWM (DPWMMAX)

3-Phase NPC Voltage Source Inverter

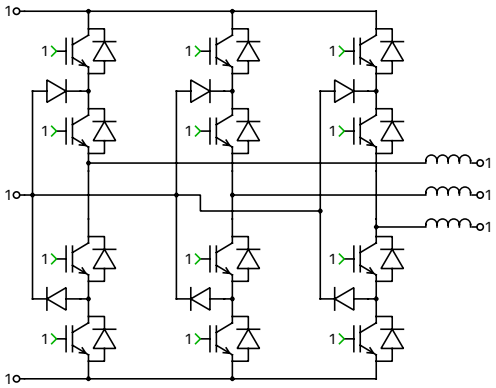
Purpose

3-phase neutral-point clamped voltage source inverter with filter inductors, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a 3-phase 3-level neutral-point clamped (NPC) voltage source inverter. It includes filter inductors on the AC side.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`
Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar or 3-element vector specifying the inductances L, in henries (H).

Winding resistance

A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Assertions

When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Nanostep Probe Signals

See Fig. 16.10 for the probe signal positions.

Inductor currents

A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

DC link currents

A vector containing the currents at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signals

A vector for each phase containing the gate signals of the semiconductor switches.

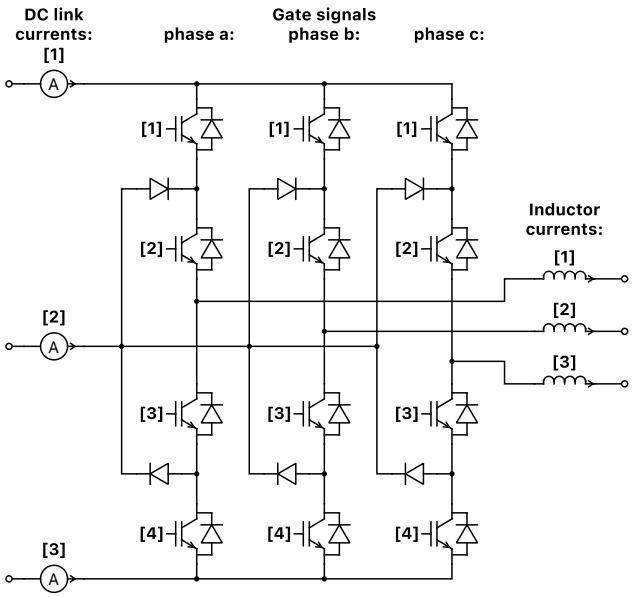


Fig. 16.10: Probes for the 3-Phase NPC Voltage Source Inverter

↪ *Added in PLECS 4.9.*

3-Phase Overmodulation

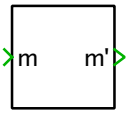
Purpose

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 (3ph) (p. 750).

Fig. 16.11 illustrates the working principle of the 3-Phase Overmodulation block in conjunction with the Symmetrical PWM (p. 988).

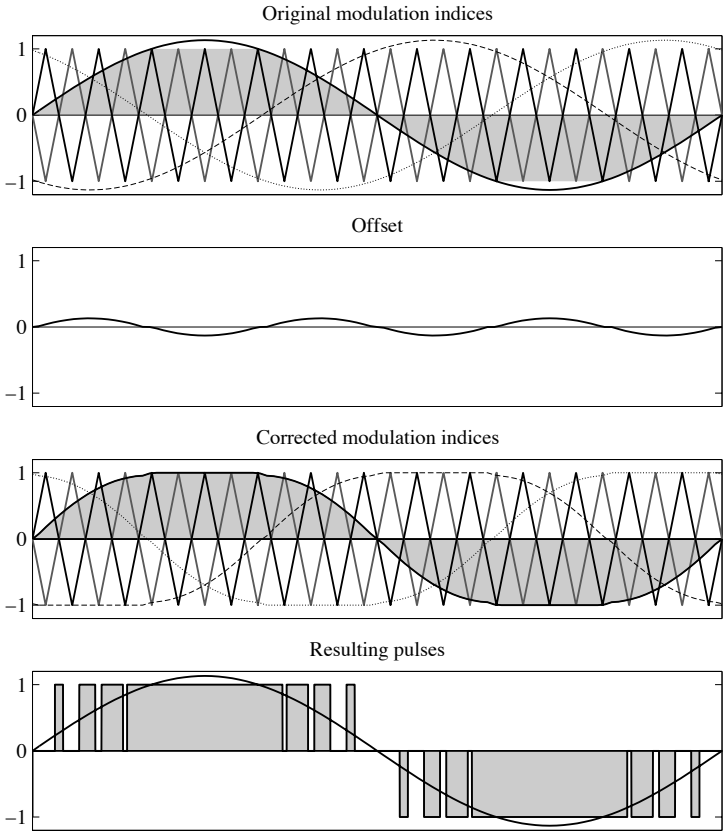


Fig. 16.11: Working principle of the 3-Phase Overmodulation block

3-Phase T-Type Voltage Source Inverter

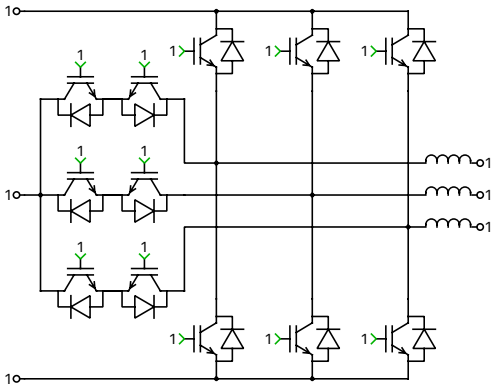
Purpose

3-phase T-type voltage source inverter with filter inductors, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a 3-phase 3-level T-type voltage source inverter. It includes filter inductors on the AC side.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`
Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Neutral-point connection
Specifies if the neutral-point switches are series connected with `Common collector` (`drain` for MOSFETs) or `Common emitter` (`source` for MOSFETs).

Inductance
A non-zero scalar or 3-element vector specifying the inductances L, in henries (H).

Winding resistance
A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Nanostep Probe Signals

See Fig. 16.12 for the probe signal positions.

Inductor currents
A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

DC link currents
A vector containing the currents at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signals
A vector for each phase containing the gate signals of the semiconductor switches.

↪ *Added in PLECS 4.9.*

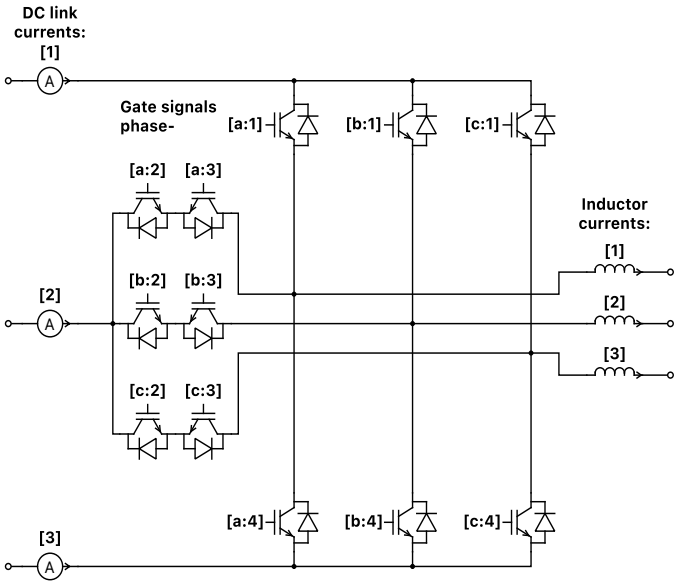


Fig. 16.12: Probes for the 3-Phase T-Type Voltage Source Inverter

3-Phase Voltage Source Inverter

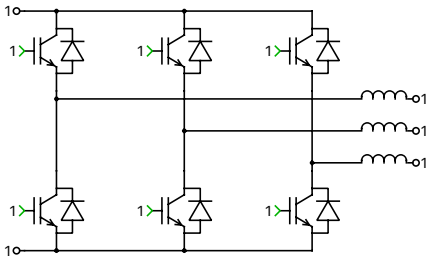
Purpose

3-phase voltage source inverter with filter inductors, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a 3-phase voltage source inverter (VSI). It includes filter inductors on the AC side.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar or 3-element vector specifying the inductances L, in henries (H).

Winding resistance
A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Nanostep Probe Signals

See Fig. 16.13 for the probe signal positions.

Inductor currents

A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

DC link current

The current at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signals

A vector for each phase containing the gate signals of the semiconductor switches.

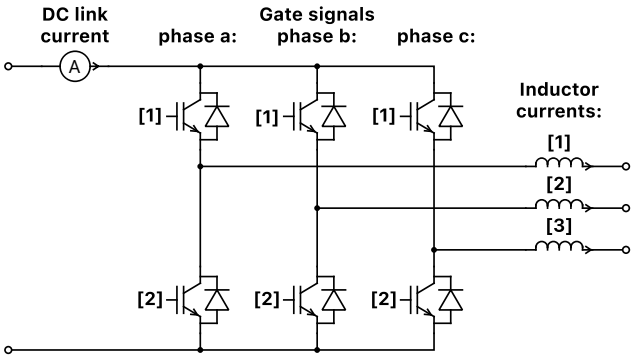


Fig. 16.13: Probes for the 3-Phase Voltage Source Inverter

↪ *Added in PLECS 4.9.*

3-Phase Voltage Source Inverter

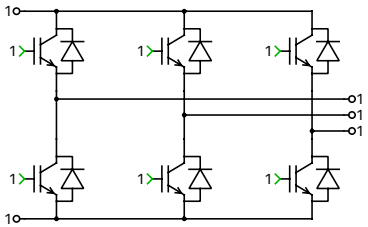
Purpose

3-phase 2-level voltage source inverter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a 3-phase 2-level voltage source inverter.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if the following condition is met:

- The sum of the gate signals for the upper and lower switches of the same leg exceeds 1.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

↪ *Added in PLECS 4.4.*

3D Look-Up Table

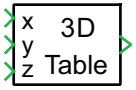
Purpose

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 (p. 508).

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.

4-Phase Voltage Source Inverter

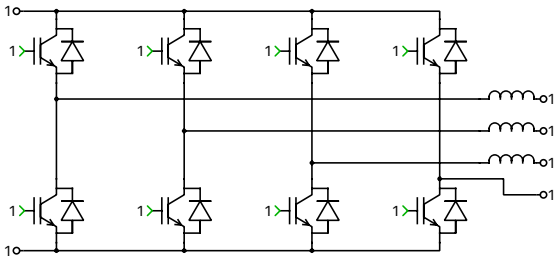
Purpose

4-phase voltage source inverter with filter inductors, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a 4-phase voltage source inverter (VSI). It includes three filter inductors on the AC side phases. The fourth phase is without filter inductor and is typically connected to the neutral point of the AC system.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding

semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

Note

This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

Note

The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance

A non-zero scalar or 3-element vector specifying the inductances L, in henries (H).

Winding resistance

A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Assertions

When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Nanostep Probe Signals

See Fig. 16.14 for the probe signal positions.

DC link current

The current at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Inductor currents

A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

Neutral-point current

The d-phase current, in amperes (A), defined positive in the direction of the arrow.

Gate signals

A vector for each phase containing the gate signals of the semiconductor switches.

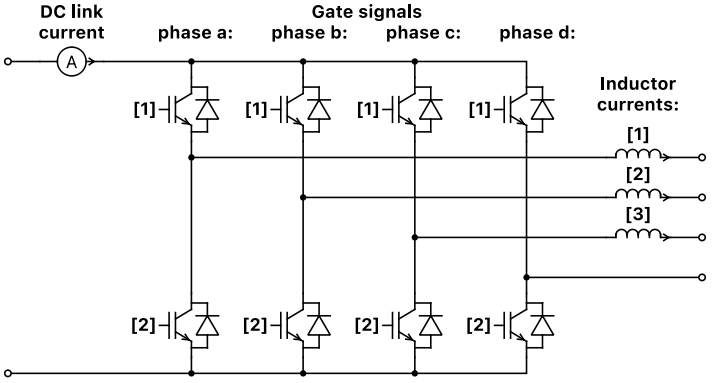


Fig. 16.14: Probes for the 4-Phase Voltage Source Inverter

↪ *Added in PLECS 5.0.*

5-Level Half Bridge (ANPC)

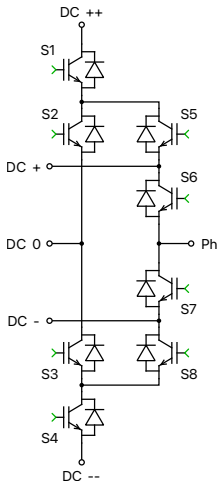
Purpose

5-level active neutral-point clamped half-bridge converter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a single leg of a 5-level active neutral-point clamped voltage source converter. It contains eight switches S1 to S8 as indicated in the figure.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if any condition in the following table is met:

Gate signal condition	Shoot-through path
$S_1 + S_2 > 1$	`DC++` to `DC0`
$S_3 + S_4 > 1$	`DC0` to `DC--`
$S_6 + S_7 > 1$	`DC+` to `DC-`
$S_5 + S_8 > 1$	Invalid flying-capacitor connection

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

↪ *Added in PLECS 4.4.*

6-Pulse Generator

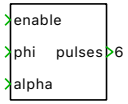
Purpose

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 φ (produced e.g. by a PLL), and the firing angle alpha.

If the `Double pulses` option is selected, each thyristor receives two pulses: one when the firing angle is reached, and a second, when the next thyristor is fired.

Parameters

Pulse width

The width of the firing pulses in radians with respect to one period of fundamental frequency.

Pulse type

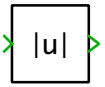
`Double pulses` enables a second firing pulse for each thyristor.

Abs

Purpose Calculate absolute value of input signal

Library Control / Math

Description The Abs block outputs the absolute value of the input signals, $y = |u|$.



Probe Signals

Input
The block input signal.

Output
The block output signal.

Air Gap

Purpose

Air gap in a magnetic core

Library

Magnetic / Components

Description



This component models an air gap in a magnetic core. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force $\mathcal{F}$

$$\frac{\Phi}{\mathcal{F}} = \frac{\mu_0 A}{l}$$

where $\mu_0 = 4\pi \times 10^{-7}\,\mathrm{N/A^2}$ 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 square meters $(\mathrm{m^2})$.

Length of flux path
Effective length l of the air gap, in meters (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 amperes per meter $(\mathrm{A/m})$.

Flux density
The magnetic flux density B in the air gap, in teslas (T).

Algebraic Constraint

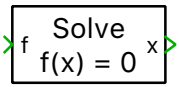
Purpose

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” (p. 36) for more information on algebraic loops.

> **Example Model**
>
> - See the example model **“Algebraic Constraint”**.
> - Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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.

↪ *Added in PLECS 4.1.*

Ambient Temperature

Purpose

Connect to Heat Sink on which component is placed

Library

Thermal / Connectivity

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” (p. 177).

> **Note**
>
> Ambient Temperature blocks may not be used in schematics that contain Thermal Port blocks (p. 1019).

Parameters

Allow vector connection to scalar heat sink

This parameter is only relevant if the block has a non-scalar inner connection. When the parameter is set to `on`, the enclosing heat sink must also be non-scalar and have a matching width. When the parameter is set to `off` (the default), the enclosing heat sink may also be scalar, and all elements of the inner connection will be connected to the single heat sink element.

Ammeter

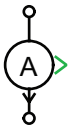
Purpose

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 Signal Outport block (p. 952) 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.

Probe Signals

Measured current

The measured current in amperes (A).

Angle Sensor

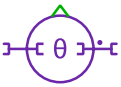
Purpose

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 Signals

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” (p. 135)). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” (p. 113)).

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, 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” (p. 135)). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” (p. 113)).

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, 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” (p. 135)). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” (p. 113)).

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, 0 otherwise.

Upper limit

The upper limit input signal.

Assertion

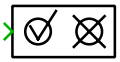
Purpose

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” (p. 135)). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” (p. 113)).

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. A highlight level of `1` means that the component containing the assertion block will be highlighted when the assertion fails, etc.

Assert Lower Limit

Purpose

Issue a warning or an error message when the input exceeds the specified lower limit

Library

Assertions

Description

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.



Parameters

Lower limit
A constant specifying the lower 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” (p. 135)). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” (p. 113)).

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, 0 otherwise.

Assert Range

Purpose

Issue a warning or an error message when the input leaves the specified range

Library

Assertions

Description



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.

Parameters

Upper limit
A constant specifying the upper limit for the input signal.

Lower limit
A constant specifying the lower limit for the input signal.

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” (p. 135)). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” (p. 113)).

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, 0 otherwise.

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” (p. 135)). Note that during analyses and simulation scripts, assertions may be partly disabled (see “Assertions” (p. 113)).

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, 0 otherwise.

Bidirectional Phase-Shifted Full-Bridge Converter

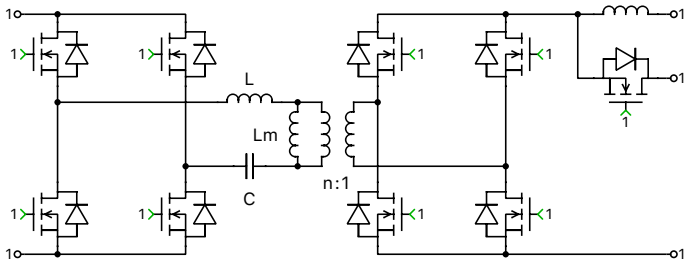
Purpose

Bidirectional phase-shifted full-bridge converter with optional series capacitor, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a bidirectional phase-shifted full-bridge converter with free-wheeling on the secondary side. A series DC blocking capacitor to prevent transformer saturation is optional.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.15 on p. 570.

Include primary capacitor
Allows you to include (yes) or remove (no) a DC blocking capacitor at the primary side, to prevent transformer saturation.

Primary capacitance
If the **Include primary capacitor** option is set to yes, this parameter requires a non-zero scalar for the capacitance C, in farads (F).

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Include magnetizing inductance
Allows you to include (yes) or remove (no) the magnetizing inductance L_m on the transformer primary side.

Magnetizing inductance
If the **Include magnetizing inductance** option is set to yes, this parameter requires a non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Filter inductance
A non-zero scalar specifying the filter inductance L_f, in henries (H).

Filter resistance
A scalar specifying the filter winding resistance R_{Lf}, in ohms (Ω).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.15 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Filter current
The current through the inductor L_f, in amperes (A), defined positive in the direction of the arrow.

Snubber current
The current through the snubber switch, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

Snubber gate signal
The gate signal of the snubber semiconductor switch.

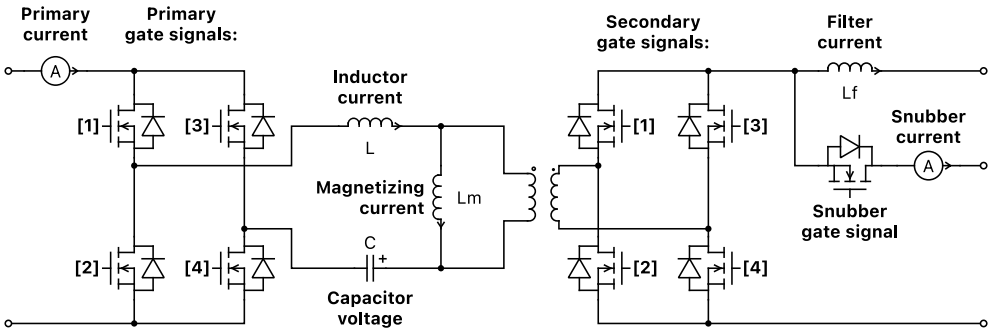


Fig. 16.15: Probe Signals for the Bidirectional Phase-Shifted Full-Bridge Converter

---

↪ *Added in PLECS 5.0.*

Blanking Time

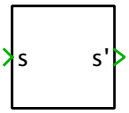
Purpose

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 (p. 988). 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.

Parameters

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)

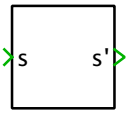
Purpose

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) (p. 992). 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.

Parameters

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.

Boost Converter

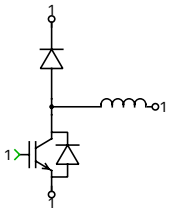
Purpose

Boost converter with filter inductance, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a boost converter including the filter inductor.

Both sides of the converter have current source behavior. The high-voltage side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 1, indicating that it occupies $\frac{1}{3}$ of a Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar specifying the inductance L, in henries (H).

Winding resistance
A scalar specifying the winding resistance $\mathrm{R_L}$, in ohms (Ω).

Nanostep Probe Signals

See Fig. 16.16 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

DC link current
The current at the high-voltage side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signal
The gate signal applied to the active semiconductor switch.

↪ *Added in PLECS 4.9.*

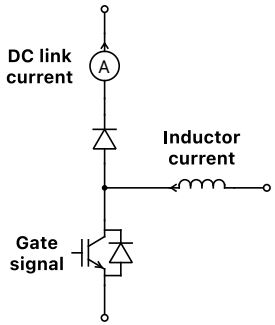


Fig. 16.16: Probes for the Boost Converter

Boost PFC Converter

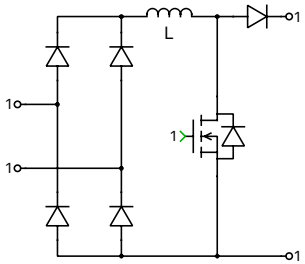
Purpose

Rectifier with boost converter for power factor correction, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a single-phase diode rectifier with a boost converter for power factor correction (PFC).

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 1, indicating that it occupies $\frac{1}{3}$ of a Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.17.

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar specifying the inductance L, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Probe Signals

See Fig. 16.17 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Input current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Output current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signal
The gate signal applied to the active semiconductor switch.

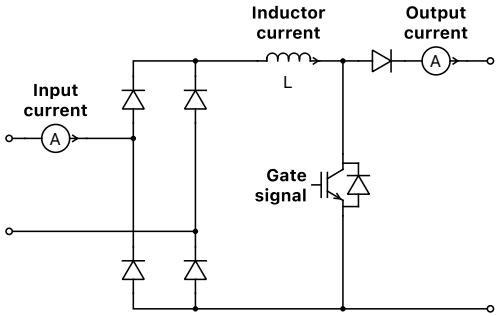


Fig. 16.17: Probes for the Boost PFC Converter

↪ *Added in PLECS 4.9.*

Boost PFC Converter with LC Filter

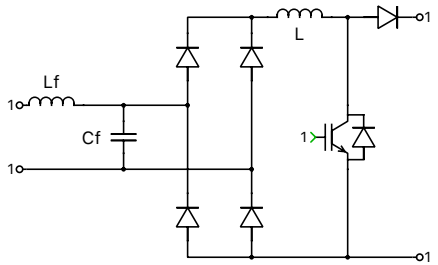
Purpose

Rectifier with boost converter and LC input filter for power factor correction, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a single-phase diode rectifier with a boost converter for power factor correction (PFC). It includes an LC filter on the AC side.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding

semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.18 on p. 582.

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar specifying the inductance L, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Filter inductance
A non-zero scalar specifying the filter inductance L_f, in henries (H).

Filter winding resistance
A scalar specifying the filter winding resistance $\mathrm{R_{Lf}}$, in ohms (Ω).

Filter capacitance
A scalar specifying the filter capacitance $\mathrm{C_f}$, in farads (F).

Filter capacitor series resistance
A scalar specifying the filter capacitor series resistance $\mathrm{R_{Cf}}$, in ohms (Ω).

Probe Signals

See Fig. 16.18 for the probe signal positions.

DC inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Filter capacitor voltage
The voltage across the filter capacitor $\mathrm{C_f}$, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

DC inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Filter inductor current
The current through the inductor $\mathrm{L_f}$, in amperes (A), defined positive in the direction of the arrow.

Output current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Filter capacitor voltage
The voltage across the filter capacitor $\mathrm{C_f}$, in volts (V), defined positive at the side marked with (+).

Gate signal
The gate signal applied to the active semiconductor switch.

↪ *Added in PLECS 5.0.*

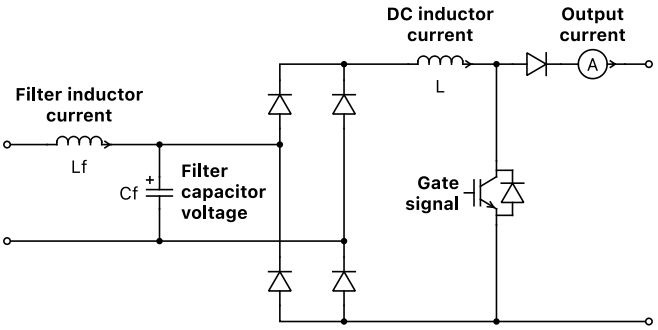


Fig. 16.18: Probes for the Boost PFC Converter with LC Filter

Breaker

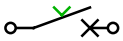
Purpose

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.

Parameters

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`.

Switch model (CPU code generation)

Select the switch model in the generated code.

Fundamental grid frequency

This parameter only takes effect when using the `non-ideal` switch model configuration. In this setup, the breaker can only interrupt the harmonic component of the current which is specified by this parameter, in hertz (Hz). The default 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

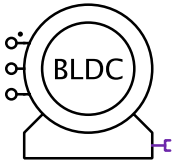
Purpose

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

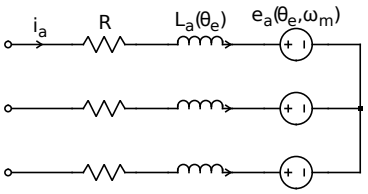


Fig. 16.19: Electrical system of a brushless DC machine

The back EMF voltages are determined by a shape function k_e and the mechanical rotor speed ω_m. The shape function in turn is expressed as a fourier series of the electrical rotor angle θ_e:

$$e_\mathrm{x}(\theta_\mathrm{e}, \omega_\mathrm{m}) = k_\mathrm{e,x}(\theta_\mathrm{e}) \cdot \omega_\mathrm{m}$$

$$k_\mathrm{e,a}(\theta_\mathrm{e}) = \sum_n K_{\mathrm{c},n} \cos(n\theta_\mathrm{e}) + K_{\mathrm{s},n} \sin(n\theta_\mathrm{e})$$

$$k_\mathrm{e,b}(\theta_\mathrm{e}) = \sum_n K_{\mathrm{c},n} \cos(n\theta_\mathrm{e} - \frac{2\pi n}{3}) + K_{\mathrm{s},n} \sin(n\theta_\mathrm{e} - \frac{2\pi n}{3})$$

$$k_\mathrm{e,c}(\theta_\mathrm{e}) = \sum_n K_{\mathrm{c},n} \cos(n\theta_\mathrm{e} + \frac{2\pi n}{3}) + K_{\mathrm{s},n} \sin(n\theta_\mathrm{e} + \frac{2\pi n}{3})$$

The stator self inductance is also expressed as a fourier series of the electrical rotor angle. The mutual inductance M between the stator phases is assumed to be constant. Since the stator windings are star connected, the mutual inductance can simply be subtracted from the self inductance:

$$L_{\mathrm{a}}(\theta_{\mathrm{e}}) = L_0 - M + \sum_n L_{\mathrm{c},n}\cos(n\theta_{\mathrm{e}}) + L_{\mathrm{s},n}\sin(n\theta_{\mathrm{e}})$$

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:

$$T_{\mathrm{e}} = \sum_{x=\mathrm{a,b,c}} k_{\mathrm{e},x} i_x + \frac{p}{2}\frac{dL_x}{d\theta_{\mathrm{e}}} i_x^2$$

The cogging torque is again expressed as a fourier series of the electrical rotor angle:

$$T_{\mathrm{cog}}(\theta_{\mathrm{e}}) = \sum_n T_{\mathrm{c},n}\cos(n\theta_{\mathrm{e}}) + T_{\mathrm{s},n}\sin(n\theta_{\mathrm{e}})$$

Mechanical System

Mechanical rotor speed:

$$\dot{\omega}_{\mathrm{m}} = \frac{1}{J}\left(T_{\mathrm{e}} + T_{\mathrm{cog}}(\theta_{\mathrm{e}}) - F\omega_{\mathrm{m}} - T_{\mathrm{m}}\right)$$

Mechanical and electrical rotor angle:

$$\dot{\theta}_{\mathrm{m}} = \omega_{\mathrm{m}}$$

$$\theta_{\mathrm{e}} = p \cdot \theta_{\mathrm{m}}$$

Parameters

Back EMF shape coefficients

Fourier coefficients $K_{\mathrm{c},n}$ and $K_{\mathrm{s},n}$ of the back EMF shape function $k_{\mathrm{e,a}}(\theta_{\mathrm{e}})$ in (Vs).

Stator resistance

The stator resistance R in ohms (Ω).

Stator inductance
The constant inductance $L_0 - M$ and the fourier coefficients $L_{c,n}$, $L_{s,n}$ of the phase a inductance $L_a(\theta_e)$ in henries (H).

Cogging torque coefficients
Fourier coefficients $T_{c,n}$, $T_{s,n}$ of the cogging torque $T_{cog}(\theta_e)$ in (Nm).

Inertia
Combined rotor and load inertia J in (Nms^2).

Friction coefficient
Viscous friction F in (Nms).

Number of pole pairs
Number of pole pairs p.

Initial rotor speed
Initial mechanical speed $\omega_{m,0}$ in radians per second $\left(\frac{rad}{s}\right)$.

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 amperes (A). Currents flowing into the machine are considered positive.

Back EMF
The back EMF voltages e_a, e_b, e_c in volts (V).

Rotational speed
The rotational speed ω_m of the rotor in radians per second $\left(\frac{rad}{s}\right)$.

Rotor position
The mechanical rotor angle θ_m in radians.

Electrical torque
The electrical torque T_e of the machine in (Nm).

Cogging torque
The cogging torque T_{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)

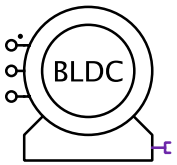
Purpose

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

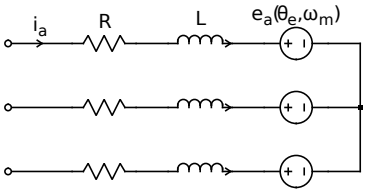


Fig. 16.20: Electrical system of a (simple model) brushless DC machine

The back EMF voltages are determined by a shape function k_e and the mechanical rotor speed ω_m. The shape function is a sinusoidal or an ideal trapezoidal function scaled with the back EMF constant K_E.

$$e_\mathrm{x}(\theta_\mathrm{e}, \omega_\mathrm{m}) = k_{\mathrm{e},x}(\theta_\mathrm{e}) \cdot \omega_\mathrm{m}$$

Electromechanical System

The electromagnetic torque is:

$$T_\mathrm{e} = \sum_{x=\mathrm{a,b,c}} k_{\mathrm{e},x} i_x$$

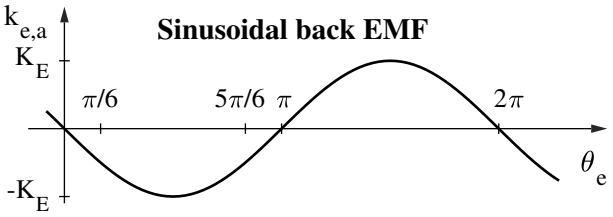


Fig. 16.21: Sinusoidal shape function

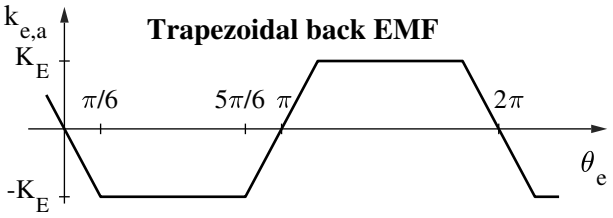


Fig. 16.22: Ideal trapezoidal shape function

Mechanical System

Mechanical rotor speed:

$$\dot{\omega}_{\mathrm{m}} \;=\; \frac{1}{J}\left(T_{\mathrm{e}} - F\omega_{\mathrm{m}} - T_{\mathrm{m}}\right)$$

Mechanical and electrical rotor angle:

$$\dot{\theta}_{\mathrm{m}} \;=\; \omega_{\mathrm{m}}$$

$$\theta_{\mathrm{e}} \;=\; p \cdot \theta_{\mathrm{m}}$$

Parameters

Back EMF shape
Choose between sinusoidal and trapezoidal back EMF.

Back EMF constant
The back EMF constant K_{E} in (Vs).

Stator resistance
The stator resistance R in ohms (Ω).

Stator inductance
The stator inductance $L - M$ in henries (H).

Inertia
Combined rotor and load inertia J in (Nms^2).

Friction coefficient
Viscous friction F in (Nms).

Number of pole pairs
Number of pole pairs p.

Initial rotor speed
Initial mechanical speed $\omega_{\mathrm{m},0}$ in radians per second $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Initial rotor angle
Initial mechanical rotor angle $\theta_{\mathrm{m},0}$ in radians.

Initial stator currents
A two-element vector containing the initial stator currents $i_{\mathrm{a},0}$ and $i_{\mathrm{b},0}$ 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 amperes (A). Currents flowing into the machine are considered positive.

Back EMF
The back EMF voltages e_{a}, e_{b}, e_{c} in volts (V).

Stator flux (dq)
The stator flux linkages Ψ_{d} and Ψ_{q} in the stationary reference frame in (Vs).

Rotational speed
The rotational speed ω_{m} of the rotor in radians per second $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Rotor position
The mechanical rotor angle θ_{m} in radians.

Electrical torque
The electrical torque T_{e} of the machine in (Nm).

References

- D. Hanselman, “Brushless permanent magnet motor design, 2nd ed.”, The Writers’ Collective, Mar. 2003.
- P. Pillay, R. Krishnan, “Modeling, simulation, and analysis of permanent-magnet motor drives, Part II: The brushless DC motor drive”, IEEE Trans. on Ind. App., Vol. 25, No. 2, Mar./Apr. 1989.

See also

For back EMF shapes other than sinusoidal or trapezoidal, and/or if the stator inductance L is angle dependent, please use the sophisticated model of the Brushless DC Machine (p. 584). The sophisticated BLDC machine can be configured as a simple BLDC machine with sinusoidal back EMF if the parameters are converted as follows:

$$K_{\mathrm{c},n} = [0]$$

$$K_{\mathrm{s},n} = [-K_{\mathrm{E}}]$$

$$L_0 - M = L - M$$

$$L_{\mathrm{c},n} = [0]$$

$$L_{\mathrm{s},n} = [0]$$

For machines with sinusoidal back EMF you may also consider to use the Permanent Magnet Synchronous Machine (p. 862). The parameters can be converted as follows provided that the stator inductance L is independent of the rotor angle:

$$[L_{\mathrm{d}} \;\; L_{\mathrm{q}}] = [L - M \;\; L - M]$$

$$\varphi'_{\mathrm{m}} = K_{\mathrm{E}}/p$$

Buck Converter

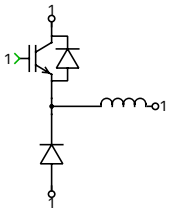
Purpose

Buck converter with filter inductance, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a buck converter including the filter inductor.

Both sides of the converter have current source behavior. The high-voltage side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

Note

This component has a Nanostep (p. 363) solver weight of 1, indicating that it occupies $\frac{1}{3}$ of a Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

Note

DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar specifying the inductance L, in henries (H).

Winding resistance
A scalar specifying the winding resistance $\mathrm{R_L}$, in ohms (Ω).

Nanostep Probe Signals

See Fig. 16.23 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

DC link current
The current at the high-voltage side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signal
The gate signal applied to the active semiconductor switch.

↪ *Added in PLECS 4.9.*

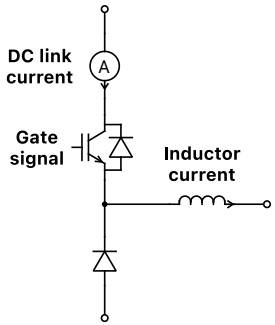


Fig. 16.23: Probes for the Buck Converter

C-Script

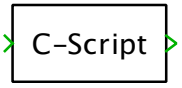
Purpose

Execute custom C code

Library

Control / Functions & Tables

Description



The C-Script block allows for custom functionality to be implemented in the C programming language. For a detailed description of C-Scripts see chapter "C-Scripts" (p. 269).

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. If you enter a scalar, the block will have one input or output terminal, and the scalar value determines the signal width. If you enter a vector, the number of elements determines the number of input or output terminals, and the elements in the vector determine the signal width of the corresponding terminal. The first input or output terminal is marked with a dot.

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.

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" (p. 36)).

If the C-Script block has one input terminal, a flag in the vector is applied to the corresponding element of the input signal vector; if the block has multiple input terminals, a flag is applied to the whole signal of the corresponding terminal. You can also specify a single scalar, which applies to all signals of all input terminals.

Sample time
A scalar specifying the sampling period or an $n \times 2$ matrix specifying the sampling period(s) and offset(s), in seconds (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" (p. 274).

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`	

Use terminal-based sample times
If this box is checked, the C-Script block uses different sample times for the individual input and output terminals. The **Sample time** parameter must be a matrix with one row per input terminal (in order) followed by one row per output terminal (in order) optionally followed by further block sample times.

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.

↪ *Changed in PLECS 4.1:* For older models,the extensions are enabled in C-Script blocks using the C99 language standard.

Highlight level
The number of levels the highlight is propagated upwards in the component hierarchy when the C-Script flags a diagnostic message. A highlight level of `0` means that the C-Script block itself will be highlighted. A highlight level of `1` means that the component containing the C-Script block will be highlighted when the assertion fails, etc.

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” (p. 270). 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” (p. 289).

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** (on macOS: **Cmd+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

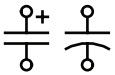
Purpose

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” (p. 60) 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 $[C_1 \;\; C_2 \ldots C_n]$. The length n of the vector determines the component's width:

$$\begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} C_1 & 0 & \cdots & 0 \\ 0 & C_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt}v_1 \\ \frac{d}{dt}v_2 \\ \vdots \\ \frac{d}{dt}v_n \end{bmatrix}$$

In order to model a coupling between the internal capacitors enter a square matrix. The size n of the matrix corresponds to the width of the

component:

$$\begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} C_1 & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_2 & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n,1} & C_{n,2} & \cdots & C_n \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt}v_1 \\ \frac{d}{dt}v_2 \\ \vdots \\ \frac{d}{dt}v_n \end{bmatrix}$$

The capacitance matrix must be invertible, i.e. it may not be singular.

Initial voltage

The initial voltage of the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the width of the component. The positive pole is marked with a "+". The initial voltage default is 0.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with "+" is greater than the potential at the unmarked terminal.

Capacitor current

The current flowing through the capacitor, in amperes (A).

Chopper (High-Side Switch)

Purpose

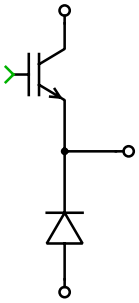
Buck converter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description

This Power Module implements a chopper used in buck converters.



The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The high-voltage side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The low-voltage side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value 1 for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Chopper (High-Side Switch with Reverse Diode)

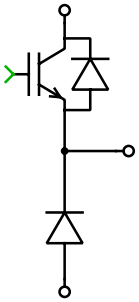
Purpose

Buck converter with reverse diode and a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a chopper with a reverse diode in the high-side switch. It is typically used in buck converters. If the reverse diode is not needed, use the alternative chopper module (p. 599) to minimize model complexity.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The high-voltage side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The low-voltage side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value 1 for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Chopper (Low-Side Switch)

Purpose

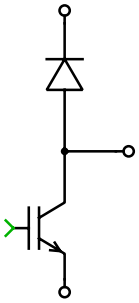
Boost converter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description

This Power Module implements a chopper used in boost converters.



The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The high-voltage side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The low-voltage side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value 1 for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Chopper (Low-Side Switch with Reverse Diode)

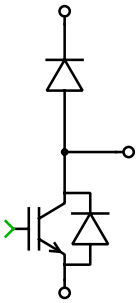
Purpose

Boost converter with reverse diode and a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a chopper with a reverse diode in the low-side switch. It is typically used in boost converters. If the reverse diode is not needed, use the alternative chopper module (p. 603) to minimize model complexity.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The high-voltage side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The low-voltage side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

CLLLC Converter

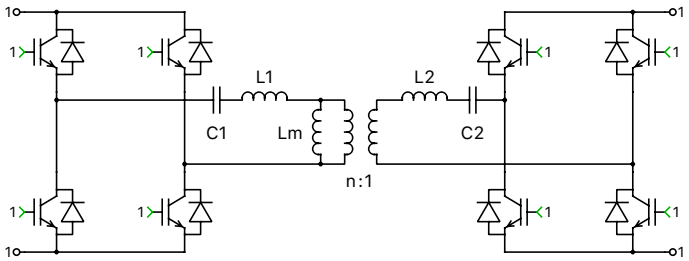
Purpose

Dual active bridge or resonant converter with magnetizing inductance, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated, resonant CLLLC converter. By omitting the capacitors, it can be configured as a dual active bridge (DAB) with magnetizing inductance. If only the primary side capacitor is present, the component can be used as a full-bridge LLC with synchronous rectification or an LLC with secondary leakage inductance.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.24 on p. 610.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Primary inductance
A non-zero scalar specifying the primary-side inductance L_1, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Primary winding resistance
A scalar specifying the primary-side winding resistance R_{L1}, in ohms (Ω).

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a resonant capacitor at the primary side.

Primary resonant capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C_1, in farads (F).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Include secondary inductance
Allows you to include (`yes`) or remove (`no`) a series leakage inductance on the secondary side.

Secondary inductance
If the **Include secondary inductance** option is set to `yes`, this parameter requires a non-zero scalar specifying the secondary-side series inductance L_2, including the transformer leakage inductance and any external inductors, in henries (H).

Secondary winding resistance
If the **Include secondary inductance** option is set to yes, this parameter requires a scalar specifying the secondary-side winding resistance R_{L2}, in ohms (Ω).

Include secondary capacitor
Allows you to include (yes) or remove (no) a resonant capacitor at the secondary side.

Secondary resonant capacitance
If the **Include secondary capacitor** option is set to yes, this parameter requires a non-zero scalar for the capacitance C_2, in farads (F).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.24 for the probe signal positions.

Inductor currents
A vector containing the currents through the inductors L_1 and L_2,in amperes (A),defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m,in amperes (A),defined positive in the direction of the arrow.

Capacitor voltages
A vector containing the voltages across the capacitors C_1 and C_2,in volts (V),defined positive at the sides marked with (+).

Nanostep Probe Signals

Inductor currents
A vector containing the currents through the inductors L_1 and L_2, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current

The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltages

A vector containing the voltages across the capacitors C_1 and C_2, in volts (V), defined positive at the sides marked with (+).

Primary gate signals

A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals

A vector containing the gate signals of the secondary-side semiconductor switches.

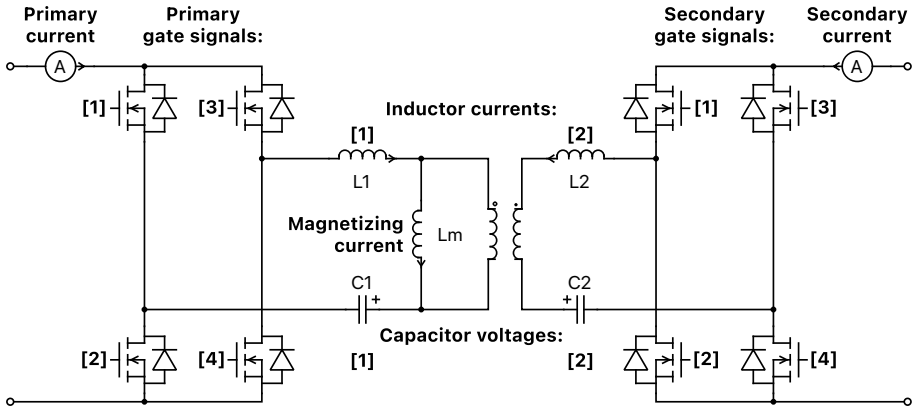


Fig. 16.24: Probes for the CLLLC Converter

↪ *Added in PLECS 4.9.*

CLLLC Converter with NPC Half-Bridge

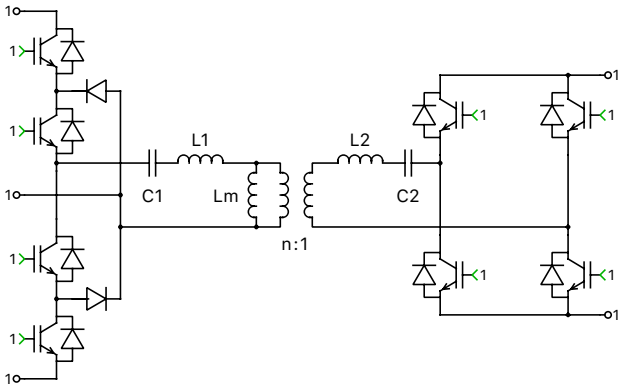
Purpose

Dual active bridge or resonant converter with a primary-side NPC half-bridge, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated, resonant CLLLC converter with a 3-level neutral-point clamped half bridge on the primary side. By omitting the capacitors, it can be configured as a dual active bridge (DAB) with magnetizing inductance. If only the primary side capacitor is present, the component can be used as a full-bridge LLC with synchronous rectification or an LLC with secondary leakage inductance.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.25 on p. 614.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Primary inductance
A non-zero scalar specifying the primary-side inductance L_1, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Primary winding resistance
A scalar specifying the primary-side winding resistance R_{L1}, in ohms (Ω).

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a resonant capacitor at the primary side.

Primary resonant capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C_1, in farads (F).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Include secondary inductance
Allows you to include (`yes`) or remove (`no`) a series leakage inductance on the secondary side.

Secondary inductance
If the **Include secondary inductance** option is set to `yes`, this parameter requires a non-zero scalar specifying the secondary-side series inductance L_2, including the transformer leakage inductance and any external inductors, in henries (H).

Secondary winding resistance
If the **Include secondary inductance** option is set to `yes`, this parameter requires a scalar specifying the secondary-side winding resistance R_{L2}, in ohms (Ω).

Include secondary capacitor
Allows you to include (`yes`) or remove (`no`) a resonant capacitor at the secondary side.

Secondary resonant capacitance
If the **Include secondary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C_2, in farads (F).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.25 for the probe signal positions.

Inductor currents
A vector containing the currents through the inductors L_1 and L_2, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltages
A vector containing the voltages across the capacitors C_1 and C_2, in volts (V), defined positive at the sides marked with (+).

Nanostep Probe Signals

Inductor currents
A vector containing the currents through the inductors L_1 and L_2, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary currents
A vector containing the currents at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltages
A vector containing the voltages across the capacitors C_1 and C_2, in volts (V), defined positive at the sides marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

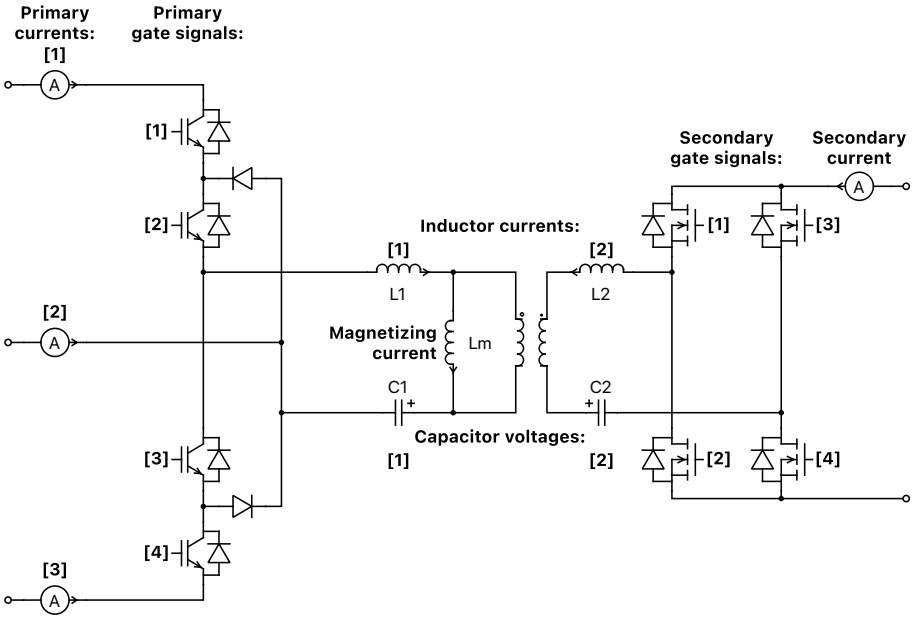


Fig. 16.25: Probes for the CLLLC Converter with NPC Half-Bridge

↪ *Added in PLECS 4.9.*

Clock

Purpose Provide current simulation time

Library Control / Sources

Description The Clock block outputs the current simulation time.



Probe Signals

Output
The time signal.

Combinatorial Logic

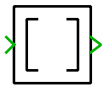
Purpose

Use binary input signals to select one row from truth table

Library

Control / Logical

Description



The Combinatorial Logic block interprets its input 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]

Parameters

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

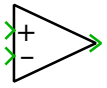
Purpose

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

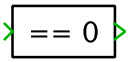
Purpose

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, the block 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 (\sim=)
- 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 signals.

Configurable Subsystem

Purpose

Provide subsystem with exchangeable implementations

Library

System

Description



A configurable subsystem is a subsystem that has multiple, exchangeable configurations. Each subsystem configuration has its own schematic diagram.

All subsystem configurations of the configurable subsystem share the same input, output and physical 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 **Edit > Subsystem** 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.

> **Note**
>
> A subsystem (see the Subsystem block (p. 977)) can be converted to a configurable subsystem by selecting **Convert to configurable subsystem** from the **Edit > Subsystem** menu or the block's context menu.

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 Dialog" (p. 98) for more details).

Probe Signals

Probe signals for the Configurable Subsystem can be created by masking the block (see "Mask Probe Signals" (p. 104) for more details).

Only the probed components from the current configuration are used during simulation. To ensure that a Probe signal behaves the same for all configurations, it should contain the same number of component signals from each subsystem in the same order. If a configuration does not provide a specific signal, a dummy component (e.g. a Constant block) can be used.

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.

Parameters

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`.

Output data type

The data type of the output signal. See “Data Types” (p. 48).

Probe Signals

Output

The constant signal.

Constant Heat Flow

Purpose Generate constant heat flow

Library Thermal / Sources

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.



Parameters

Heat flow
The magnitude of the heat flow, in watts (W). The default is 1.

Probe Signals

Heat flow
The heat flow in watts (W).

Constant Temperature

Purpose

Provide constant temperature

Library

Thermal / Sources

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.

Parameters

Temperature

The temperature difference generated by the component, in degrees Celsius (°C). The default is 0.

Probe Signals

Temperature

The temperature difference in degrees Celsius (°C).

Continuous PID Controller

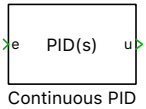
Purpose

Implementation of a continuous-time controller (P, I, PI, PD or PID)

Library

Control / Continuous

Description



This block implements a highly configurable continuous-time controller with two different anti-windup mechanisms, see subsection **Anti-windup methods**. The output signal is a weighted sum of at maximum three types of control actions: proportional action, integral action and derivative action. A practical implementation of the derivative action needs an additional first-order low-pass filter. The selection of the filter time constant K_f is a trade-off between filtering noise and avoiding interactions with the dominant PID controller dynamics. This leads to the transfer function below:

$$C_{\mathrm{PID}}(s) = \frac{U(s)}{E(s)} = K_{\mathrm{p}} + \frac{K_{\mathrm{i}}}{s} + \frac{K_{\mathrm{d}} s}{\frac{1}{K_{\mathrm{f}}} s + 1}.$$

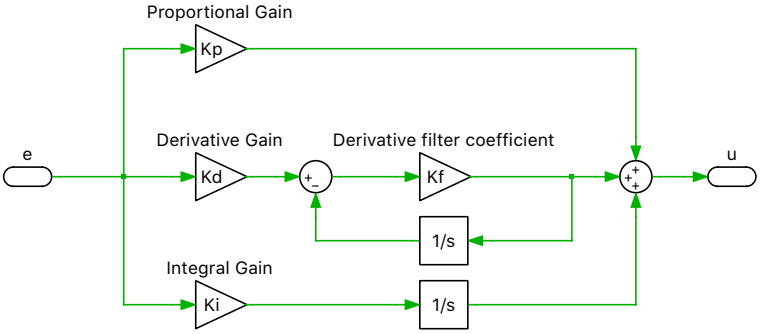


Fig. 16.26: Implementation of a PID controller in parallel form

Anti-windup Methods

Set-point changes together with actuator saturation limits can lead to the integrator windup effect and degraded controller performance. To avoid this phenomenon, this component implements the following two mechanisms.

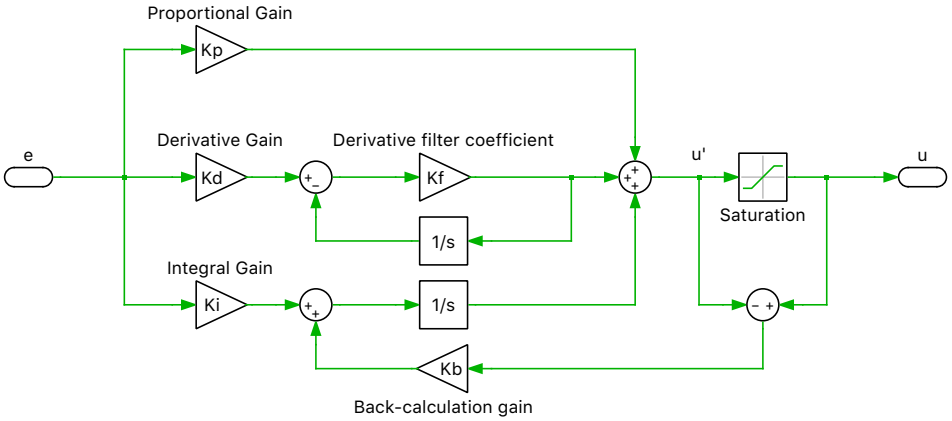


Fig. 16.27: Anti-windup scheme with back-calculation

Back-Calculation

The **Back-calculation** method changes the integral action when the controller output is in saturation. The integral term is reduced/increased when the controller output is higher/lower than the upper/lower saturation limit. This is done by feeding back the difference between the saturated and the unsaturated controller output. The value of the back-calculation gain K_{bc} determines the rate at which the integrator is reset and is therefore crucial for performance of the anti-windup mechanism. A common choice for this back-calculation gain is:

$$K_{\mathrm{bc}} = \sqrt{\frac{K_{\mathrm{i}}}{K_{\mathrm{d}}}}.$$

However, this rule can only be applied to full PID controllers. In the case of a PI controller, where $K_{\mathrm{d}} = 0$, it is suggested that

$$K_{\mathrm{bc}} = \frac{K_{\mathrm{i}}}{K_{\mathrm{p}}}.$$

Conditional Integration

The **Conditional integration** method stops the integration when the controller output saturates and the control error e and the control variable u have the same sign. This means, the integral action is still applied if it helps to push the controller output out of saturation.

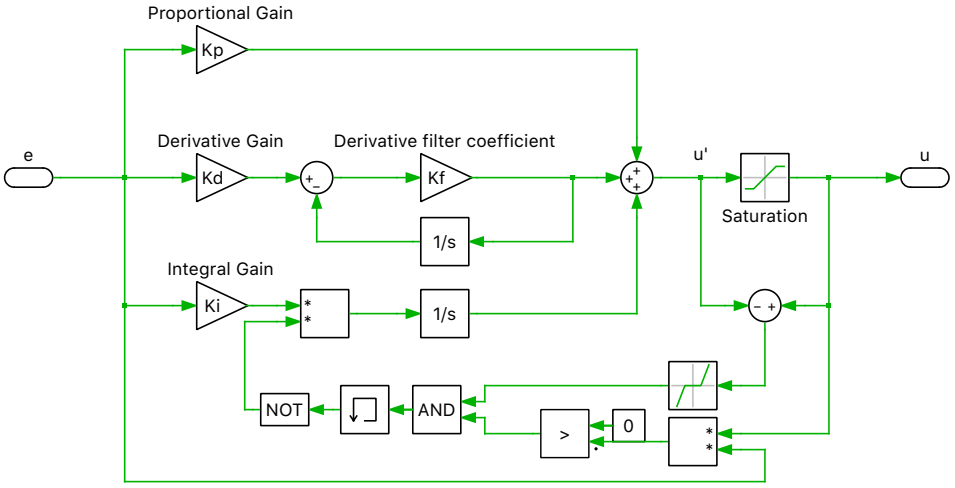


Fig. 16.28: Anti-windup scheme with conditional integration

Anti-Windup with External Saturation

If the saturation is placed externally to the PID controller block, i.e. **Saturation** is set to `external`, one can still use the above described anti-windup methods. To feedback the saturated output of the external Saturation block (p. 936) an additional subsystem port u^* is made visible if the **Saturation** parameter is set to `external` and the **Anti-windup method** parameters is set to an option other than `none`, see the figure below.

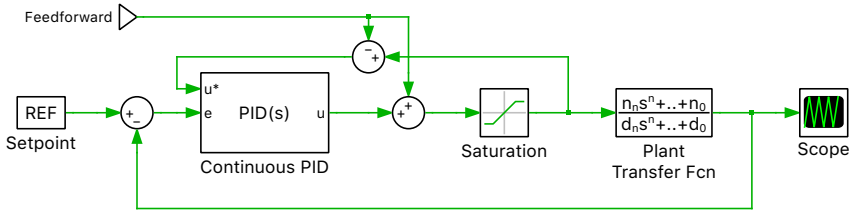


Fig. 16.29: Anti-windup scheme with external saturation

Parameters

Basic

Controller type
Specifies the controller type. The controller can be of type `P`, `I`, `PI`, `PD` or `PID`.

Parameter source
Specifies whether the controller parameters are provided via the mask parameters (`internal`) or via input signals (`external`).

Proportional gain Kp
The proportional gain of the controller. This parameter is shown only if the **Controller type** parameter is set to `P`, `PI`, `PD` or `PID` and the **Parameter source** parameter is set to `internal`.

Integral gain Ki
The integral gain of the controller. This parameter is shown only if the **Controller type** parameter is set to `I`, `PI` or `PID` and the **Parameter source** parameter is set to `internal`.

Derivative gain Kd
The derivative gain of the controller. This parameter is shown only if the **Controller type** parameter is set to `PD` or `PID` and the **Parameter source** parameter is set to `internal`.

Derivative filter coefficient Kf
The filter coefficient which specifies the pole location of the first-order filter in the derivative term. This parameter is shown only if the **Controller type** parameter is set to `PD` or `PID` and the **Parameter source** parameter is set to `internal`.

External reset
The behavior 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 `level` is chosen, the output signal keeps the initial value while the reset input is not 0. Only the integrator in the integral action is reset.

Initial condition source
Specifies wheter 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 in the integral action. 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`.

Anti-Windup

Saturation
Specifies if the internally placed saturation (`internal`) is used or if the user wants to place the saturation externally (`external`) to the Continuous PID Controller block (p. 624). If `external` is selected, the internal Saturation block (p. 936) is not active.

Saturation limits
Specifies whether the saturation limits are provided via the mask parameters (`constant`) or via input signals (`variable`).

Upper saturation limit
An upper limit for the output signal. If the value is `inf`, the output signal is unlimited. If input and output are vectorized, signals a vector can be used. The number of elements in the vector must match the number of input signals. This parameter is shown only if the **Saturation** parameter is set to `internal` and the **Saturation limits** parameter is set to `constant`.

Lower saturation limit
A lower limit for the output signal. If the value is `-inf`, the output signal is unlimited. If input and output are vectorized, signals a vector can be used. The number of elements in the vector must match the number of input signals. This parameter is shown only if the **Saturation** parameter is set to `internal` and the **Saturation limits** parameter is set to `constant`.

Anti-Windup method
Specifies the method to avoid windup of the integral action. See **Anti-windup methods** above.

Back-calculation gain
The gain of the back-calculation anti-windup method. This parameter is shown only of the **Anti-windup method** parameter is set to `Back-calculation`.

Probe Signals

Proportional action
Proportion of the proportional action of the controller output signal.

Integral action
Proportion of the integral action of the controller output signal.

Derivative action
Proportion of the derivative action of the controller output signal.

Controller output before saturation
The input signal of the saturation block.

Controller output after saturation
The output signal of the saturation block.

References

- A. Visioli, “Practical PID Control - Advances in industrial control”, Springer-Verlag, 2006.

↪ *Added in PLECS 4.4.*

Controlled Heat Flow

Purpose

Generate variable heat flow

Library

Thermal / Sources

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.

Parameters

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 Signals

Heat flow

The heat flow in watts (W).

Controlled Temperature

Purpose

Provide variable temperature

Library

Thermal / Sources

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.

Parameters

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 Signals

Temperature

The temperature difference in degrees Celsius (°C).

Current Source (Controlled)

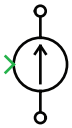
Purpose

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” (p. 39).

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 ω, in radians per second $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$. The default is `2*pi*50` which corresponds to $50\,\mathrm{Hz}$.

Phase
The phase shift φ, in radians. The default is `0`.

Probe Signals

Source current
The source current in amperes (A).

Source voltage
The voltage measured across the source, in volts (V).

Source power
The instantaneous output power of the source, in watts (W).

Current Source DC

Purpose

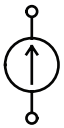
Generate constant current

Library

Electrical / Sources

Description

The DC Current Source generates a constant current between its two electrical terminals. The direction of a positive current through the component is marked with an arrow.



> **Note**
>
> A current source may not be open-circuited or connected in series to an inductor or any other current source.

Parameters

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).

Data Type

Purpose

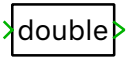
Cast the input signal to the specified data type.

Library

Control / Math

Description

This block casts the input signal to the specified data type. If the input signal already has the specified data type, this block does nothing.



Parameters

Data type

The data type of the output signal. See “Data Types” (p. 48).

Data type overflow handling

Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if **Data type** is not set to a floating-point data type.

↪ *Added in PLECS 4.2.*

DC Machine

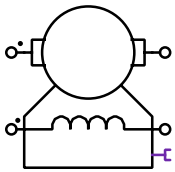
Purpose

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

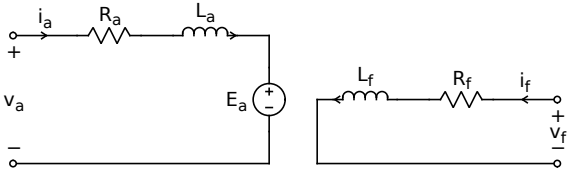


Fig. 16.30: Electrical system of a (simple model) DC machine

Electromagnetic torque:

$$T_{\mathrm{e}} = L_{\mathrm{af}} \cdot i_{\mathrm{f}} \cdot i_{\mathrm{a}}$$

Induced voltage of the armature winding:

$$E_{\mathrm{a}} = L_{\mathrm{af}} \cdot i_{\mathrm{f}} \cdot \omega_{\mathrm{m}}$$

Mechanical System

$$\dot{\omega}_{\mathrm{m}} = \frac{1}{J}\left(T_{\mathrm{e}} - F\omega_{\mathrm{m}} - T_{\mathrm{m}}\right)$$

Parameters

Armature resistance
Armature winding resistance R_{a} in ohms (Ω).

Armature inductance
Armature winding inductance L_{a} in henries (H).

Field resistance
Field winding resistance R_{f} in ohms (Ω).

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 (Nms^2).

Friction coefficient
Viscous friction F in (Nms).

Initial rotor speed
Initial mechanical speed $\omega_{\mathrm{m},0}$ in radians per second $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Initial rotor position
Initial mechanical rotor angle $\theta_{\mathrm{m},0}$ in radians.

Initial armature current
Initial current $i_{\mathrm{a},0}$ in the armature winding in amperes (A).

Initial field current
Initial current $i_{\mathrm{f},0}$ 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 ω_{m} of the rotor in radians per second $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Rotor position
The mechanical rotor angle θ_{m} in radians.

Electrical torque
The electrical torque T_{e} of the machine in (Nm).

Dead Zone

Purpose

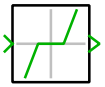
Output zero while input signal is within dead zone limits

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

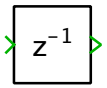
Delay input signal by given number of samples

Library

Control / Discrete

Description

The Delay block delays the input signal by N sample periods.



Example Model

- See the example model **"Delay"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

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
A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section "Sample Times" (p. 43).

Probe Signals

Input
The input signal.

Output
The delayed output signal.

D Flip-flop

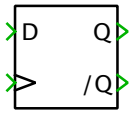
Purpose

Implement edge-triggered flip-flop

Library

Control / Logical

Description



The D flip-flop sets its output Q to the value of its input D when an edge on the clock input is detected. The behavior is shown in the following truth table:

D	Clk	Q	/Q
0	0	No change	No change
0	1	No change	No change
1	0	No change	No change
1	1	No change	No change
0	Triggering edge	0	1
1	Triggering edge	1	0

The input D is latched, i.e. when a triggering edge in the clock signal is detected the value of D from the previous simulation step is used to set the output. In other words, D must be stable for at least one simulation step before the flip-flop is triggered by the clock signal.

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 $/Q$.

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 (Ω). The default is `0`.

Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter "Thermal Modeling" (p. 171). If no thermal description is given, the losses are calculated based on the voltage drop $v_\mathrm{on} = V_\mathrm{f} + R_\mathrm{on} \cdot i$.

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in $(\mathrm{K/W})$. The default is `0`.

↪ *Added in PLECS 4.6.*

Number of parallel devices

This parameter is used to simulate the effect of connecting multiple *identical* devices in parallel while adding only one single switch element to the electrical system equations. Other component parameters such as the **Thermal description**, the **Thermal interface resistance** and also the electrical **On-resistance** refer to the characteristics of an individual device, while the calculated losses and other probe signals refer to the sum over all devices.

If N_p is the number of parallel devices, each device will conduct $1/N_\mathrm{p}$ of the total current through the component, and the total component loss will be N_p times the loss of an individual device. The effective electrical on-resistance and the effective thermal interface resistance of the component are also $1/N_\mathrm{p}$ times the respective parameter values. The default is `1`.

↪ *Added in PLECS 5.0.*

Note

If **Number of parallel devices** is greater than 1, the component itself cannot be vectorized, i.e. all other component parameters must have scalar values, and the component terminals must be connected to a scalar connection (see also "Vectorizing Physical Components" (p. 78)).

Example Model

- See the example model **"Thermal Modeling with Parallel Devices"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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" (p. 177).

Switch model (CPU code generation)

Select the switch model in the generated code.

Fundamental grid frequency

Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is `0`.

> **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” (p. 205) 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 Pair

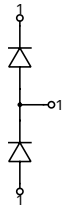
Purpose

Diode rectifier with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a sub-cycle averaged (p. 489) model of a series-connected diode pair with a single internal switching element and controlled current and voltage sources. One diode at most may conduct at a given time. The DC side of the rectifier has current source behavior. The AC side is typically connected to an inductor.

The single-switch implementation has fewer conduction states compared to two Diode components (p. 641). This reduces the number of switching combinations when generating code from a simulation model, as described in "Code Generation for Physical Systems" (p. 342).

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver.

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Switch model (CPU code generation)
: This parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Fundamental grid frequency
: Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is `0`.

Diode Rectifier (3ph)

Purpose

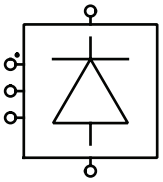
3-phase diode rectifier

Library

Electrical / Converters

Description

Implements a three-phase rectifier based on the Diode model (p. 641). The electrical circuit for the rectifier is given in Fig. 16.31.



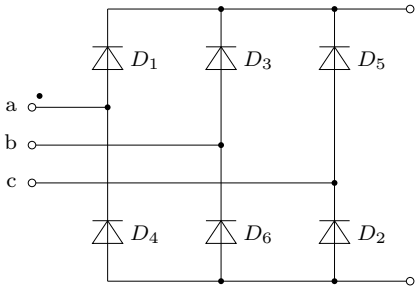


Fig. 16.31: Electrical circuit for three-phase rectifier

Parameters

For a description of the parameters see the documentation of the Diode (p. 641).

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.

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.

Fig. 16.32 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 in Fig. 16.33 on p. 648. It is composed of a resistance, and inductance, and a controlled current source which is linearly dependent on the inductor voltage. The values of these internal elements are automatically calculated from the diode parameters.

Parameters

Forward voltage

Additional dc voltage V_{f} in volts (V) between anode and cathode when the diode is conducting. The default is `0`.

On-resistance

The resistance R_{on} of the conducting device, in ohms (Ω). The default is `0`.

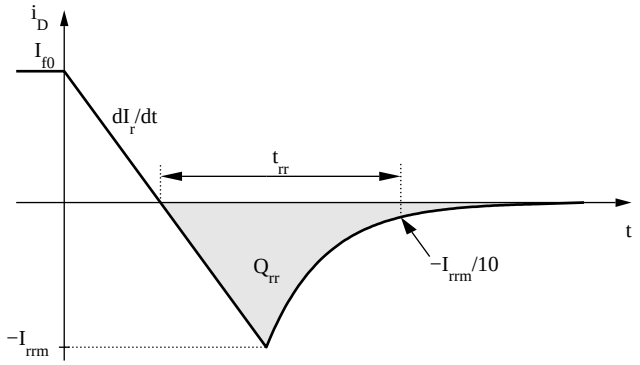


Fig. 16.32: Reverse recovery current waveform

Off-resistance
The resistance R_{off} of the blocking device, in ohms (Ω). 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} in amperes (A) under test conditions.

Current slope at turn-off
The turn-off current slope dI_{r}/dt in $\left(\frac{\mathrm{A}}{\mathrm{s}}\right)$ under test conditions.

Reverse recovery time
The turn-off time t_{rr} in seconds (s) under test conditions.

Peak recovery current
The absolute peak value of the reverse current I_{rrm} in amperes (A) under test conditions.

Reverse recovery charge
The reverse recovery charge Q_{rr} in coulombs (C) 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, in henries (H). The default is `10e-10`.

Probe Signals

Diode voltage
The voltage measured between anode and cathode.

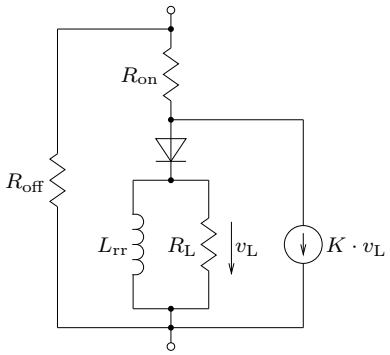


Fig. 16.33: Reverse recovery diode equivalent circuit

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.

Discrete Integrator

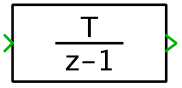
Purpose

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 integrator state may saturate at an upper and lower limit, or it can be wrapped between two limits. It can be reset to its initial value by an external trigger signal.

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 behavior 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.

Enable wrapping
When set to `on`, the integrator state is wrapped between the **Upper wrapping limit** and **Lower wrapping limit** parameters. The default is `off`. Note that wrapping is mutually exclusive with saturation.

Upper saturation limit
An upper limit for the integrator state. If the value is `inf`, the state is unlimited. This parameter is visible only if **Enable wrapping** is set to `off`.

Lower saturation limit
A lower limit for the integrator state. If the value is `-inf`, the state is unlimited. This parameter is visible only if **Enable wrapping** is set to `off`.

Upper wrapping limit
The upper wrapping limit for the integrator state. When the state exceeds the upper limit, it is wrapped to the lower limit. This parameter is visible only if **Enable wrapping** is set to `on`.

Lower wrapping limit
The lower wrapping limit for the integrator state. When the state exceeds the lower limit, it is wrapped to the upper limit. This parameter is visible only if **Enable wrapping** is set to `on`.

Integration method
The method used to integrate the input signal. See **Integration Methods** above.

Sample time
A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section "Sample Times" (p. 43).

Probe Signals

State
The internal state of the integrator.

Discrete Mean Value

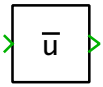
Purpose

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** minus 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

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section “Sample Times” (p. 43).

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 PID Controller

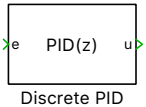
Purpose

Implementation of a discrete-time controller (P, I, PI, PD or PID)

Library

Control / Discrete

Description



This block implements a highly configurable discrete-time controller with two different anti-windup mechanisms. The output signal is a weighted sum of at maximum three types of control actions: proportional action, integral action and derivative action.

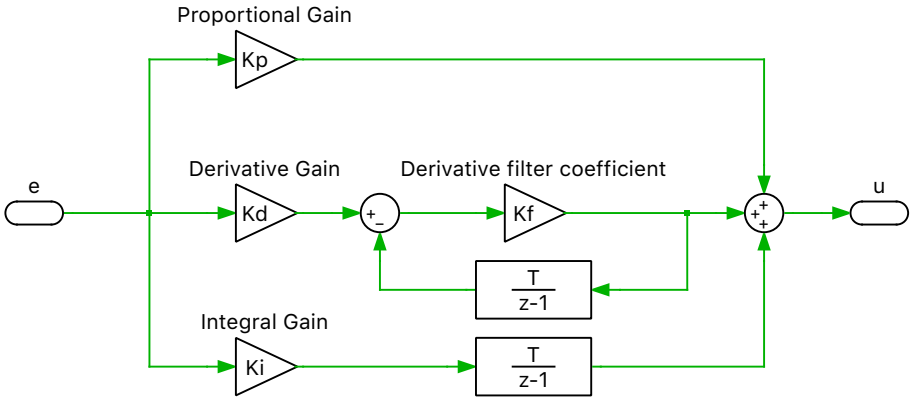


Fig. 16.34: Implementation of a discrete PID controller in parallel form

The derivative action is filtered with a first-order low-pass filter. This filter is always integrated with the Forward Euler method. The selection of the filter time constant K_f is a trade-off between filtering noise and avoid interactions with the dominant PID controller dynamics. This leads to the discrete transfer function below:

$$C_\mathrm{PID}(z) = \frac{U(z)}{E(z)} = K_\mathrm{p} + K_\mathrm{i} q(z) + K_\mathrm{d} \frac{K_\mathrm{f}}{1 + K_\mathrm{f} \frac{T}{z-1}}.$$

If `Forward Euler` is selected in the **Integration Method** parameter, then

$$q(z) = \frac{T}{z-1},$$

for `Backward Euler`

$$q(z) = \frac{Tz}{z-1},$$

and for `Trapezoidal`

$$q(z) = \frac{T}{2}\frac{z+1}{z-1}.$$

All other block parameters are explained in the continuous version of the PID controller (p. 624).

Parameters

Basic

Controller type
Specifies the controller type. The controller can be of type `P`, `I`, `PI`, `PD` or `PID`.

Parameter source
Specifies whether the controller parameters are provided via the mask parameters (`internal`) or via input signals (`external`).

Proportional gain Kp
The proportional gain of the controller. This parameter is shown only if the **Controller type** parameter is set to `P`, `I`, `PI`, `PD` or `PID` and the **Parameter source** parameter is set to `internal`.

Integral gain Ki
The integral gain of the controller. This parameter is shown only if the **Controller type** parameter is set to `I`, `PI` or `PID` and the **Parameter source** parameter is set to `internal`.

Derivative gain Kd
The derivative gain of the controller. This parameter is shown only if the **Controller type** parameter is set to `PD` or `PID` and the **Parameter source** parameter is set to `internal`.

Derivative filter coefficient Kf
The filter coefficient which specifies the pole location of the first-order filter in the derivative term.

External reset
The behavior 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 `level` is chosen, the output signal keeps the initial value while the reset input is not 0. Only the integrator in the integral action is reset.

Initial condition source
Specifies wheter 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 in the integral action. 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`.

Anti-Windup

Saturation
Specifies if the internally placed saturation (`internal`) is used or if the user wants to place the saturation externally (`external`). If `external` is selected, the internal Saturation block (p. 936) is not active.

Saturation limits
Specifies whether the saturation limits are provided via the mask parameters (`constant`) or via input signals (`variable`).

Upper saturation limit
An upper limit for the output signal. If the value is `inf`, the output signal is unlimited. If input and output are vectorized signals, a vector must be used. The number of elements in the vector must match the number of input signals. This parameter is shown only if the **Saturation** parameter is set to `internal` and the **Saturation limits** parameter is set to `constant`.

Lower saturation limit
A lower limit for the output signal. If the value is `-inf`, the output signal is unlimited. If input and output are vectorized signals, a vector must be used. The number of elements in the vector must match the number of input signals. This parameter is shown only if the **Saturation** parameter is set to `internal` and the **Saturation limits** parameter is set to `constant`.

Anti-Windup method
Specifies the method to avoid windup of the integral action. See also the **Anti-windup Methods** in the Continuous PID Controller block (p. 624).

Back-calculation gain
The gain of the back-calculation anti-windup method. This parameter is shown only of the **Anti-windup method** parameter is set to `Back-calculation`.

Discrete-time settings

Integration method
The method used to integrate the input signal. This parameter affects the integrator of the integral action only. Please note, that the first-order low-pass filter in the derivative action is always integrated using the Forward Euler method. See also the **Integration Methods** in the Discrete Integrator block (p. 649).

Sample time
A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section "Sample Times" (p. 43).

Probe Signals

Proportional action
Proportion of the proportional action of the controller output signal.

Integral action
Proportion of the integral action of the controller output signal.

Derivative action
Proportion of the derivative action of the controller output signal.

Controller output before saturation
The input signal of the saturation block.

Controller output after saturation
The output signal of the saturation block.

References

- A. Visioli, "Practical PID Control - Advances in industrial control", Springer-Verlag, 2006.

↪ *Added in PLECS 4.4.*

Discrete State Space

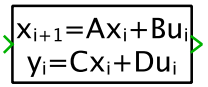
Purpose

Model discrete linear time-invariant system as state-space model

Library

Control / Discrete

Description



The Discrete State Space block models a discrete state-space system of the form

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{A}\mathbf{x}_i + \mathbf{B}\mathbf{u}_i \\ \mathbf{y}_i &= \mathbf{C}\mathbf{x}_i + \mathbf{D}\mathbf{u}_i\end{aligned}$$

where $\mathbf{x}_i$ is the state vector at sample time step i, $\mathbf{u}$ is the input vector, and $\mathbf{y}$ is the output vector. The widths of the input and output signals are determined by the number of columns of the $\mathbf{B}$ matrix and the number of rows of the $\mathbf{C}$ matrix, respectively.

For a single-input system (i.e. if the $\mathbf{B}$ matrix has only one column), the State Space block performs scalar expansion: If the input signal is a vector, the state-space model is applied to each element of the input vector individually, and the output signal is a concatenated vector of all system outputs.

↪ *Added in PLECS 4.9.*

Parameters

A, B, C, D

The coefficient matrices for the discrete state-space system are shown in Fig. 16.35. The dimensions for the coefficient matrices must conform to the dimensions shown in the diagram below:

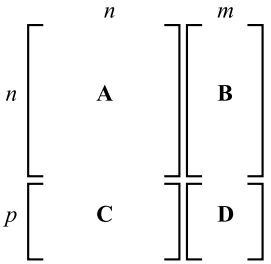


Fig. 16.35: Coefficient matrices for a discrete state-space system

where n is the number of states, m is the width of the input signal and p is the width of the output signal.

If the matrix $\mathbf{D}$ is empty, it is internally expanded to a zero matrix of the size $p \times m$.

Sample time
A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section "Sample Times" (p. 43).

Initial condition
A vector of initial values for the state vector, $\mathbf{x} = [x_0, x_1 \ldots x_n]$.

> **Note**
>
> A scalar initial condition will be applied to all internal states.

> **Note**
>
> In case of scalar expansion (the system is a single-input system with multiple input signals), the initial condition can also be a matrix, where each row defines the initial condition for the individual inputs.
> ↪ *Added in PLECS 4.9.*

Probe Signals

Input
The input vector, **u**.

Output
The output vector, **y**.

State
The state vector, **x**.

Discrete Transfer Function

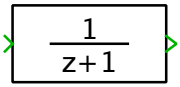
Purpose

Model discrete linear time-invariant system as transfer function

Library

Control / Discrete

Description



The Discrete Transfer Function models a discrete linear time-invariant system that is expressed in the z-domain:

$$\frac{Y(z)}{U(z)} = \frac{n_n z^n + \cdots + n_1 z + n_0}{d_n z^n + \cdots + d_1 z + d_0}$$

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.

The Transfer Function block performs scalar expansion: If the input signal is a vector, the transfer function is applied to each element of the input vector individually, and the output signal is a concatenated vector of all system outputs.

Parameters

Numerator coefficients

A vector of the z term coefficients $[n_n \ldots n_1, n_0]$ 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 Transfer Function supports multiple outputs for a single input by entering a matrix for the numerator. Each row of the matrix defines the numerator coefficients of an output.

Denominator coefficients

A vector of the z term coefficients $[d_n \ldots d_1, d_0]$ 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

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section “Sample Times” (p. 43).

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 condition must be specified for the controller normal form, depicted in Fig. 16.36 for the transfer function:

$$\frac{Y(z)}{U(z)} = \frac{n_2 z^2 + n_1 z + n_0}{d_2 z^2 + d_1 z + d_0} = b_2 \left(a_2 + \frac{a_1 z + a_0}{z^2 + b_1 z + b_0} \right)$$

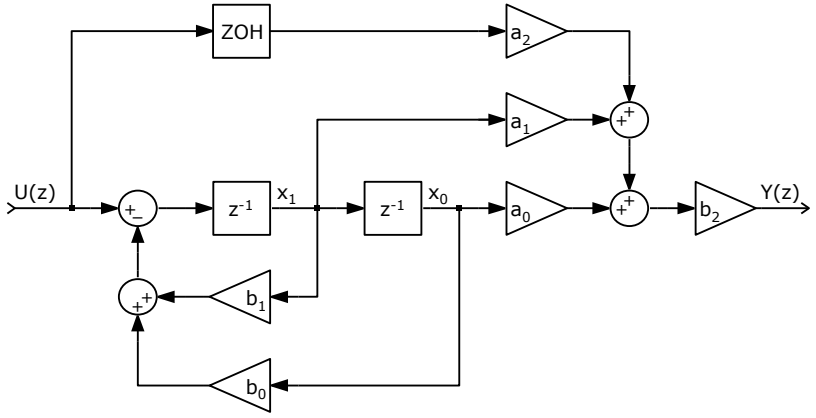


Fig. 16.36: Discrete controller normal form

where

$$\begin{aligned} b_i &= \frac{d_i}{d_n} && \text{for } i < n \\ b_n &= \frac{1}{d_n} \\ a_i &= n_i - \frac{n_n d_i}{d_n} && \text{for } i < n \\ a_n &= n_n \end{aligned}$$

For the normalized transfer function (with $n_n = 0$ and $d_n = 1$) this simplifies to $b_i = d_i$ and $a_i = n_i$.

Note

The number of internal states is defined by the highest power of z of the denominator.

> **Note**
>
> A scalar initial condition will be applied to all internal states.

> **Note**
>
> In case of scalar expansion (multiple input signals), the initial condition can also be a matrix, where each row defines the initial condition for the individual inputs.

Probe Signals

Input
The input signal.

Output
The output signal.

State
The internal states of the controller normal form.

Display

Purpose

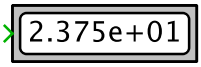
Display signal values in the schematic

Library

System

Description

The display block shows the numeric value of the input signal. The value is updated while the simulation is running.



For signals with a fixed-point data type the letters *fx* are displayed to the left of the signal's value. When clicking on *fx* the displayed value is toggled between the real representation and the raw underlying integer value.

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, in seconds (s).

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. For a 64-bit version of PLECS, a 64-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 64-bit	*filename*, *filename*.dll, *filename*_64.dll
macOS 64-bit	*filename*, *filename*.dylib, *filename*_64.dylib
Linux 64-bit	*filename*, *filename*.so, *filename*_64.so

Sample time
A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). This parameter 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" (p. 43).

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 `plecsSetSizes` method.

Probe Signals

Input
The input signal.

Output
The output signal.

Double Switch

Purpose

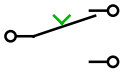
Changeover switch with two positions

Library

Electrical / Switches

Description

This changeover switch provides an ideal short or open circuit. If the input signal is zero, the switch is in the upper position. For all other values the switch is in the lower position.



Parameters

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`.

Switch model (CPU code generation)

Select the switch model in the generated code.

Fundamental grid frequency

Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is `0`.

Probe Signals

Switch position

State of the internal switches. The signal outputs 0 if the switch is in the upper position, and 1 if it is in the lower position.

Dynamic Signal Selector

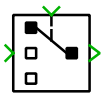
Purpose

Select or reorder elements from vectorized signal depending on control signal.

Library

System

Description



The block generates an output vector signal that consists of the specified elements of the input vector signal. The elements can be dynamically selected with the control input.

See also the Signal Selector block (p. 954).

Parameters

Default index
The default element if the control signal is out of range.

Indexing
Choose whether the elements from the input vector are selected using 0-based or 1-based indexing.

Probe Signals

Input
The block input signal.

Output
The block output signal.

Switch position
The state of the switch position. This is the control signal or, in case the control signal is out of range, the default index.

Out of range
A boolean value that is true if the control input is out of range.

↪ *Added in PLECS 4.5.*

Edge Detection

Purpose

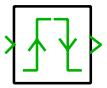
Detect edges of pulse signal in given direction

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.

Parameters

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

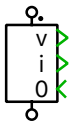
Purpose

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.

Fig. 16.37 shows a possible implementation of a variable resistor.

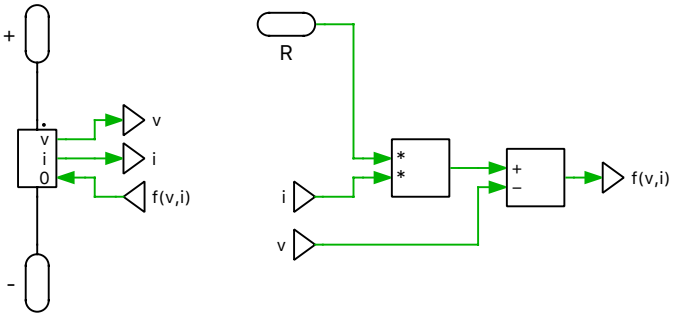


Fig. 16.37: Possible implementation of a variable resistor.

There is also a Variable Resistor component (p. 1109) in the library.

Note

The Electrical Algebraic Component creates an algebraic loop. See section "Block Sorting" (p. 36) for more information on algebraic loops.

Example Model

- See the example model **"Algebraic Component"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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.

↪ *Added in PLECS 4.1.*

Electrical Ground

Purpose Connect to common electrical ground

Library Electrical / Connectivity

Description The ground block implements an electrical connection to the ground.



> **Note**
>
> PLECS electrical circuits do not require to be grounded at one or more points. The ground block just provides a convenient means to connect distant points to a common potential. In SPICE electrical circuits, it is **mandatory** to put a ground.

> **Warning**
>
> Electrical grounds in PLECS electrical circuits are not connected to electrical grounds in SPICE electrical circuits.

Electrical Label

Purpose

Connect electrical potentials by name

Library

Electrical / Connectivity

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.

To quickly change the tag names of all matching Electrical Label blocks, choose **Rename all matching tags...** from the context menu.

↪ *Added in PLECS 4.9.*

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.

Action if there is no matching counterpart

An Electrical Label block that does not have any matching block typically indicates a modeling error. This parameter lets you specify whether PLECS should report an error or a warning or ignore this condition when a simulation is started.

Electrical Model Settings

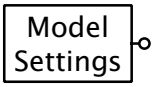
Purpose

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" (p. 342).

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" (p. 135).

Parameters

Target

This setting controls whether a physical model will be executed on the `CPU` or the `FlexArray` solver of a target. The default is `CPU`. The option `FlexArray` should be chosen only if the target has a FlexArray solver. Currently, this applies to the RT Box 2, 3 and 4 using version 3.0 or newer of the RT Box (p. 361) Target Support Package.

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. This parameter applies only to code generation for the CPU target.

↪ *Added in PLECS 4.0.*

Electrical Port

Purpose

Add electrical connector to subsystem

Library

Electrical / Connectivity

Description



Electrical ports are used to establish electrical connections between a schematic and the subschematic of a Subsystem (p. 977). 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.

A terminal can be moved around the edges of the subsystem by dragging it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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.

Parameters

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” (p. 978)) 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, drag the terminal with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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* enable 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
The data type of the output signal. See “Data Types” (p. 48).

Data type overflow handling
Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if **Output data type** is not set to a floating-point data type.

Probe Signals

Output
The output signal of the Enable block.

Flux Rate Meter

Purpose

Output the measured rate-of-change of magnetic flux

Library

Magnetic / Meters

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 Signal Outport block (p. 952) or by dragging the component into the dialog box of a Probe block.

The magnetic flux Φ cannot be measured directly in the circuit. However, most permeance components provide the magnetic flux as a probe signal.

> **Note**
>
> The Flux Rate Meter is ideal, i.e. it has infinite internal permeance. Hence, if multiple Flux Rate Meters are connected in parallel the flux through an individual meter is undefined. This produces a run-time error.

Probe Signals

Flux rate

The rate-of-change $\dot{\Phi}$ of magnetic flux flowing through the component, in (Wb/s) or V.

Flyback Converter with Snubber

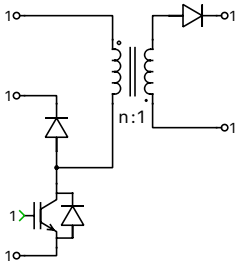
Purpose

Flyback converter with leakage inductance and snubber diode, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a flyback converter with a single output. In addition to the magnetization inductance, the transformer model contains leakage inductances on both the primary and secondary sides. To avoid excessive overvoltages when the switch is turned off, the primary side is equipped with a snubber diode. The diode can be connected to the positive input voltage rail via an RC snubber (resistor in parallel with a capacitor). It is assumed that the snubber capacitance is large enough that its voltage does not change significantly during one switching cycle.

Both sides of the converter have current source behavior. The primary side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding

semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.38 on p. 683.

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Primary inductance
A non-zero scalar specifying the primary-side inductance L_1, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Primary winding resistance
A scalar specifying the primary-side winding resistance R_{L1}, in ohms (Ω).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer $\mathrm{L_m}$, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Secondary inductance
A non-zero scalar specifying the secondary-side series inductance $\mathrm{L_2}$, including the transformer leakage inductance and any external inductors, in henries (H).

Secondary winding resistance
A scalar specifying the secondary-side winding resistance $\mathrm{R_{L2}}$, in ohms (Ω).

Probe Signals

See Fig. 16.38 for the probe signal positions.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Nanostep Probe Signals

Input current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Output current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Switch current
The current flowing in the active semiconductor switch, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Gate signal
The gate signal applied to the active semiconductor switch.

↪ *Added in PLECS 4.9.*

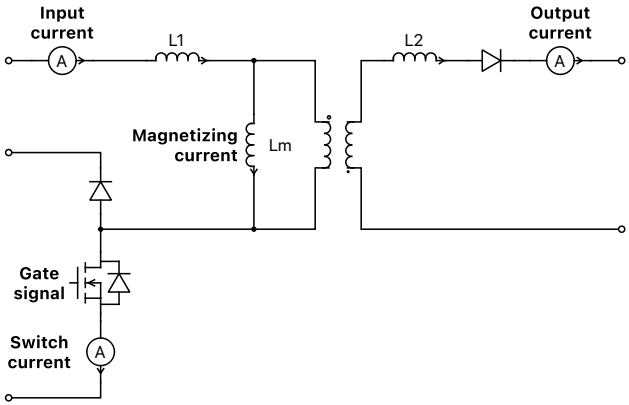


Fig. 16.38: Probes for the Flyback Converter with Snubber

Flying Capacitor Half Bridge

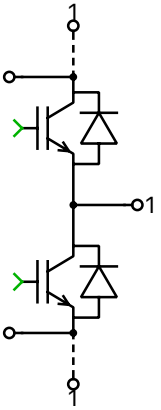
Purpose

Multi-level inverter half bridge with flying capacitors and a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a multi-level inverter half bridge with flying capacitors. The vectorized DC side terminals for the capacitors are ordered from outer to inner voltage levels. The vectorized control inputs are ordered from the top to the bottom switch.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Number of capacitors

Specifies how many capacitors and how many voltage levels the half bridge power module contains.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if the following condition is met:

- The sum of the gate signals for any checked switch pair exceeds 1. The checked pairs are (S_1, S_n), (S_2, S_{n-1}), and so on, where S_1 and S_n are connected to the outer DC bus terminals, S_2 and S_{n-1} are connected to the next inner capacitor voltage level, etc.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

↪ *Added in PLECS 4.4.*

FMU

Purpose

Include a Functional Mockup Unit in a model

Library

Control / Functions & Tables

Description



The FMU block imports existing Functional Mockup Units (FMUs) into a PLECS model. Supported FMU types are FMI Model Exchange and Co-Simulation in version 2.0 and 3.0.

↪ *Changed in PLECS 5.0:* Added support for Co-Simulation FMUs and the FMI 3.0 standard.

To open the **Functional Mockup Unit** dialog, select the FMU block and choose **Edit FMU file...** from the **Edit** menu or from the block's context menu. If no FMU file has been specified yet, you can also just double-click on the block. Click on the **...** button next to the **FMU file** field to choose an existing FMU file. By default, the FMU file is specified with a path relative to the model file. You can change this to an absolute path using the **FMU file reference** selector.

When you click **OK**, PLECS creates a folder *`filename`*`.fmu.expanded` next to the FMU file and updates the input and output terminals of the block according to the FMU description.

Once the FMU file has been specified, a double-click on the FMU block will open a dialog that allows you to view and edit the parameters that are defined by the FMU file. If the FMU file contains documentation, you can view this by clicking on the **Help** button of the parameter dialog.

↪ *Added in PLECS 4.5.*

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 force, in newtons (N). The default value is 1.

Probe Signals

Force
The generated force, in newtons (N).

Speed
The speed of the flange that the arrow points to with respect to the other flange, in $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$.

Force (Controlled)

Purpose

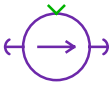
Generate variable force

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 newtons (N).

Speed
The speed of the flange that the arrow points to with respect to the other flange, in $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$.

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.

Parameters

Second flange

Controls whether the second flange is accessible or connected to the translational reference frame.

Probe Signals

Force

The measured force, in newtons (N).

Fourier Series

Purpose

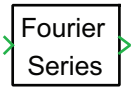
Synthesize periodic output signal from Fourier coefficients

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.

Fourier Transform

Purpose

Perform Fourier transform on input signal

Library

Control / Filters

Description



This block calculates the Fourier transform of a periodic input signal. The sample time, the fundamental frequency and the harmonic order(s) can be specified. 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

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section "Sample Times" (p. 43). If a sample time of 0 is specified, a continuous implementation based on the Moving Average block (p. 837) is active. The **Discrete-Periodic** implementation can potentially force a variable-step solver to take small integration steps. This may slow down the simulation. The default value of this parameter is 0.

Fundamental frequency

The fundamental frequency of the periodic input signal in hertz (Hz).

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.

↪ *Added in PLECS 4.4.*

Frequency-doubling LLC Converter

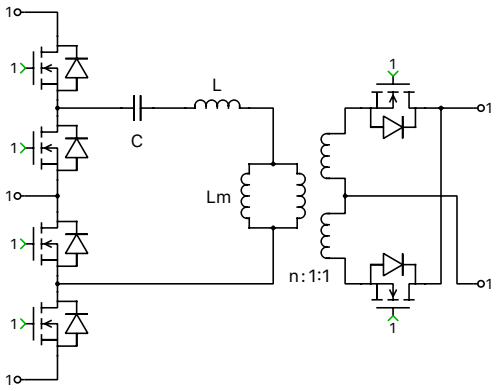
Purpose

Frequency-doubling resonant LLC converter with synchronous rectification, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated, resonant LLC converter. The input side is modeled as two series-connected half-bridges and the output side as a half-bridge rectifier with synchronous rectification.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

Note

This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.39.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Resonant capacitance
A non-zero scalar for the resonant capacitance C on the primary side, in farads (F).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.39 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary currents
A vector containing the currents at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

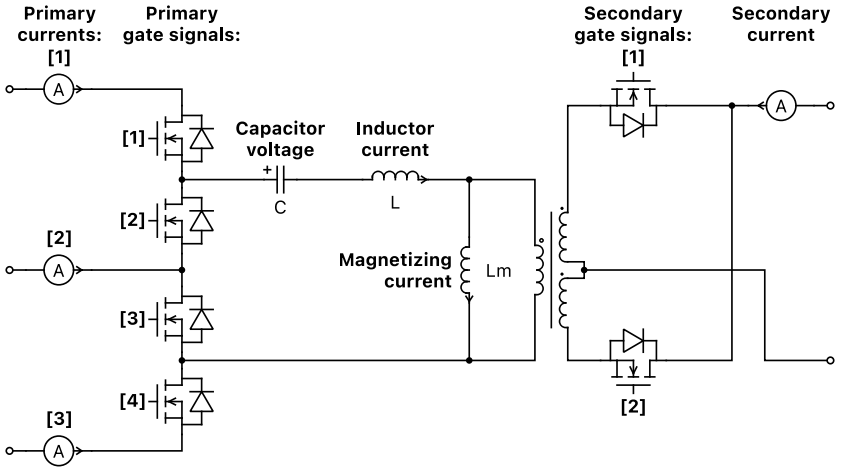


Fig. 16.39: Probe Signals for the Frequency-doubling LLC Converter

↪ *Added in PLECS 5.0.*

From File

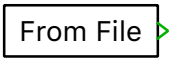
Purpose

Read time stamps and signal values from a file.

Library

System

Description



While a simulation is running, the From File block reads time stamps and signal values from a file. The file format can be either a text file with comma separated values (csv) or a MATLAB data file (mat).

During the simulation, the signal values for the current simulation time are read from the file and made available as a vectorized signal at the output of the From File block.

Data layout

↪ *Added in PLECS 4.6.*

The required input data layout differs for these two file types.

CSV file

A CSV file must contain one or more rows. The first element in each row contains a timestamp; the remaining elements contain data for the corresponding output values. Thus, the total number of columns must be equal to the parameter **Number of outputs** plus 1. The first row may contain column headers instead of numeric values. In this case, it is silently ignored.

MAT file

A MAT file must contain one matrix with real double values. Compared to the CSV file, the matrix layout is transposed. The total number of *rows* must be equal to the number of outputs plus 1. The first element in each column contains a timestamp, and the remaining elements in each column contain data for the corresponding output values.

Parameters

Filename

The name or path of the data file to read from. Relative paths are interpreted relative to the model directory.

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 the specified data file has to comply to. The block can read text files with comma separated values (csv) or MATLAB data files (mat).

Number of outputs
The number of signal values in each data record (row) of the file (which is the number of columns minus one). If this number does not correspond to the actual number of signal values provided by the file, an error message is shown.

Output before first time stamp
The method used to determine the output of the From File block for simulation times before the first time stamp in the data file. Possible values are `linear extrapolation`, `hold first value` and `specify value`. If the parameter is set to `specify value`, an additional parameter field **Initial output value** is shown to define the output value before the first time stamp.

Initial output value
The output value before the first signal value specified in the data file. This parameter is only visible if **Output before first time stamp** is set to `specify value`. The value can be given as a scalar or a vector. If it is a scalar, it is used for all signals, if it is a vector, its size has to match the **Number of outputs** parameter.

Output within time range
The method used to determine the output of the From File block for simulation times between time stamps specified in the data file. Possible values are `linear interpolation` and `zero order hold`.

Output after last time stamp
The method used to determine the output of the From File block for simulation times after the last time stamp in the data file. Possible values are `linear extrapolation`, `hold last value` and `specify value`. If the parameter is set to `specify value`, an additional parameter field **Final output value** is shown to define the output value after the last time stamp.

Final output value
The output value after the last signal value specified in the data file. This parameter is only visible if **Output after first time stamp** is set to `specify value`. The value can be given as a scalar or a vector. If it is

a scalar, it is used for all signals, if it is a vector, its size has to match the **Number of outputs** parameter.

Locate discontinuities
Possible values are `on` and `off`. If this parameter is set to `on`, the solver will ensure a time step at each time stamp specified in the data file. This setting requires a variable step solver.

↪ *Added in PLECS 4.4.*

Full-Bridge Inverter

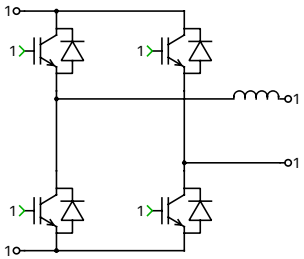
Purpose

Full-bridge voltage source inverter with filter inductance, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a full-bridge voltage source inverter including a filter inductor on the AC side.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

Note

This component has a Nanostep (p. 363) solver weight of 1, indicating that it occupies $\frac{1}{3}$ of a Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

Note

The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar specifying the inductance L, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Nanostep Probe Signals

See Fig. 16.40 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

DC link current
The current at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signals
A vector containing the gate signals of the active semiconductor switches.

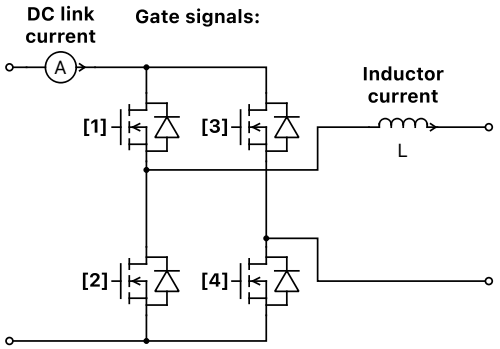


Fig. 16.40: Probes for the Full-Bridge Inverter

↪ *Added in PLECS 4.9.*

Full-Bridge Inverter with LCL Filter

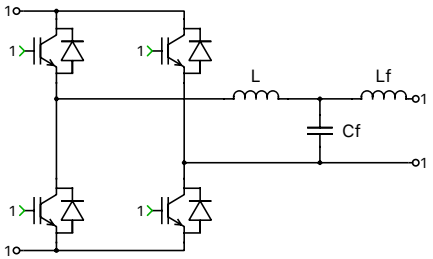
Purpose

Full-bridge voltage source inverter with LCL filter on the AC side, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a full-bridge voltage source inverter including an LCL filter on the AC side.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.41 on p. 705.

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar specifying the inductance L, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Filter inductance
A non-zero scalar specifying the filter inductance L_f, in henries (H).

Filter winding resistance
A scalar specifying the filter winding resistance $\mathrm{R_{Lf}}$, in ohms (Ω).

Filter capacitance
A scalar specifying the filter capacitance $\mathrm{C_f}$, in farads (F).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.41 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Filter capacitor voltage
The voltage across the filter capacitor $\mathrm{C_f}$, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Filter inductor current
The current through the inductor $\mathrm{L_f}$, in amperes (A), defined positive in the direction of the arrow.

DC link current
The current at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Filter capacitor voltage
The voltage across the filter capacitor $\mathrm{C_f}$, in volts (V), defined positive at the side marked with (+).

Gate signals
A vector containing the gate signals of the active semiconductor switches.

↪ *Added in PLECS 5.0.*

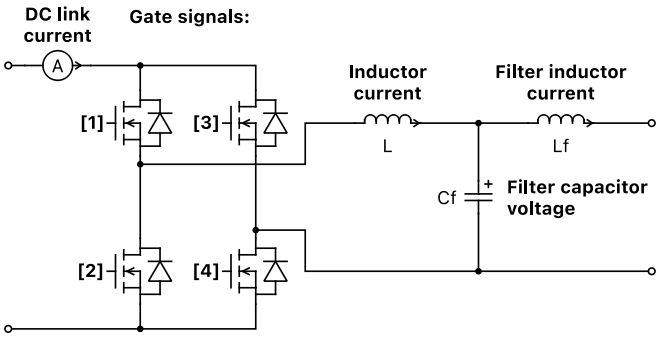


Fig. 16.41: Probes for the Full-Bridge Inverter with LCL filter

Full-Bridge LLC Converter

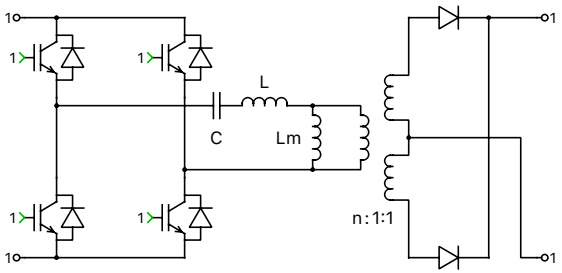
Purpose

Full-bridge LLC resonant converter, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated LLC resonant converter with a full bridge on the input side.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.42.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Output rectifier
Use this setting to choose between center-tapped `Half bridge` and `Full bridge` output rectifier topology.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Resonant capacitance
A non-zero scalar for the resonant capacitance C on the primary side, in farads (F).

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.42 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Gate signals
A vector containing the gate signals of the active semiconductor switches.

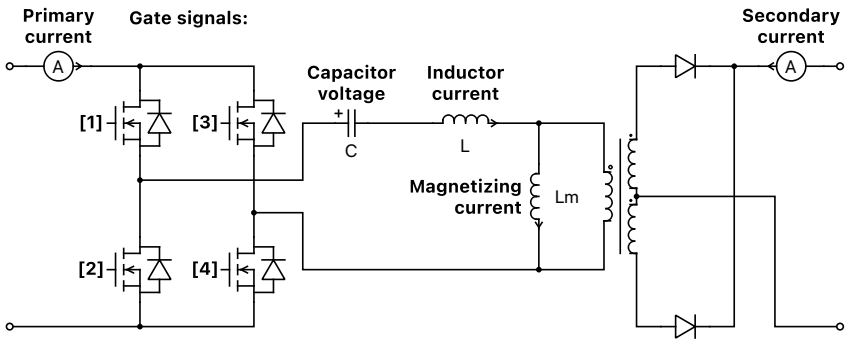


Fig. 16.42: Probes for the Full-Bridge LLC Converter

↪ *Added in PLECS 4.9.*

Full-Bridge LLC Converter with Synchronous Rectification

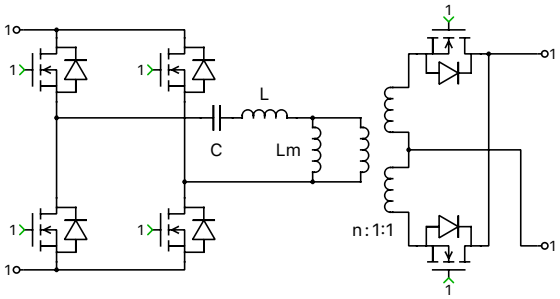
Purpose

Full-bridge LLC resonant converter with synchronous rectification, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated LLC resonant converter with a full bridge on the primary side and a half-bridge synchronous rectifier on the secondary side. To realize a full-bridge synchronous rectifier, use the CLLLC Converter (p. 607) component configured without secondary-side capacitance.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.43.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Resonant capacitance
A non-zero scalar for the resonant capacitance C on the primary side, in farads (F).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.43 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

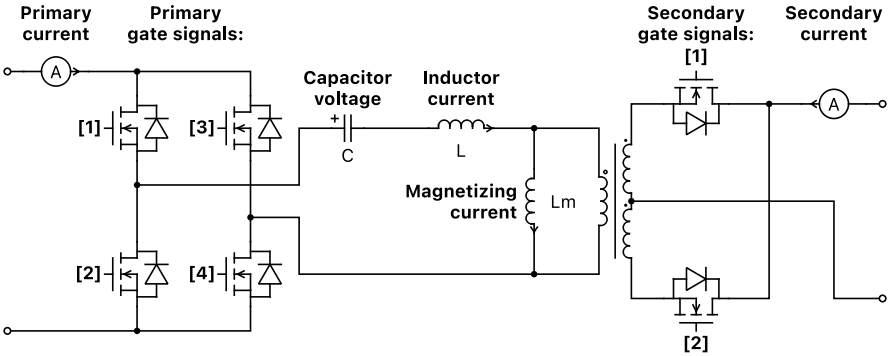


Fig. 16.43: Probe Signals for the Full-Bridge LLC Converter with Synchronous Rectification

---

↪ *Added in PLECS 5.0.*

Full Bridges (Series Connected)

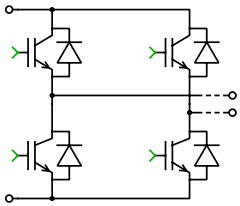
Purpose

Series-connected full-bridge inverter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



These Power Modules implement multiple series-connected inverter cells for use in modular multilevel converters. All control signals as well as the DC side terminals are vectorized with their width matching the number of cells.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> In `Sub-cycle average` configuration, the primary (DC) sides of the inverter cells must remain galvanically isolated from each other.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Number of cells

Number of series-connected full-bridge inverters.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if the following condition is met:

- The sum of the gate signals for the upper and lower switches of the same leg exceeds 1.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Function

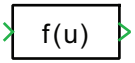
Purpose

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 element `i` 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

Parameters

Expression

The expression applied to the input signal, in C language syntax.

Probe Signals

Input

The input signal.

Output

The output signal.

Gain

Purpose

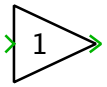
Multiply input signal by constant

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
The data type of the output signal. See “Data Types” (p. 48). If you choose `inherited`, the minimum data type is `int8_t`.

Data type overflow handling
Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if **Output data type** is not set to a floating-point data type.

Gain data type
The data type of the gain parameter. By choosing a suitable data type, the computational effort for fixed-point multiplications can be minimized. If you choose `inherited`, it is set to the output data type. This parameter only appears if **Multiplication** is set to element-wise and **Output data type** is set to a fixed-point data type.

↪ *Added in PLECS 4.9.*

Probe Signals

Input
The input signal.

Output
The output signal.

Gear

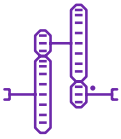
Purpose

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 ω_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.

Parameters

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 (Ω). The default is `0`.

Initial conductivity

Initial conduction state of the GTO. The GTO is initially blocking if the parameter evaluates to zero, otherwise it is conducting.

Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter "Thermal Modeling" (p. 171). If no thermal description is given, the losses are calculated based on the voltage drop $v_\mathrm{on} = V_\mathrm{f} + R_\mathrm{on} \cdot i$.

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in $(\mathrm{K/W})$. The default is `0`.

↪ *Added in PLECS 4.6.*

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" (p. 177).

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” (p. 171) and “Losses of Semiconductor Switch with Diode” (p. 206).

Thermal interface resistance
The thermal resistance of the interface material between case and heat sink, in (K/W). The default is `0`.

↪ *Added in PLECS 4.6.*

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” (p. 177).

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.

Half-Bridge Inverter

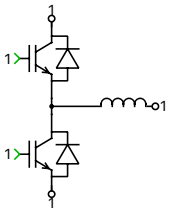
Purpose

Half-bridge voltage source inverter with filter inductance, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a half-bridge voltage source inverter including the filter inductor.

Both sides of the converter have current source behavior. The DC-link side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 1, indicating that it occupies $\frac{1}{3}$ of a Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar specifying the inductance L, in henries (H).

Winding resistance
A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Nanostep Probe Signals

See Fig. 16.44 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

DC link current

The current at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signals

A vector containing the gate signals of the active semiconductor switches.

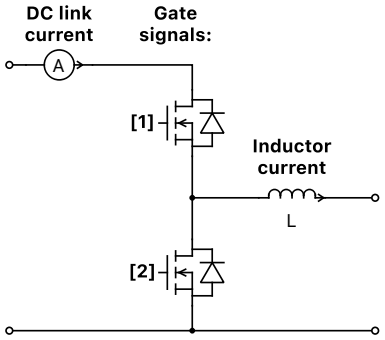


Fig. 16.44: Probes for the Half-Bridge Inverter

↪ *Added in PLECS 4.9.*

Half-Bridge LLC Converter

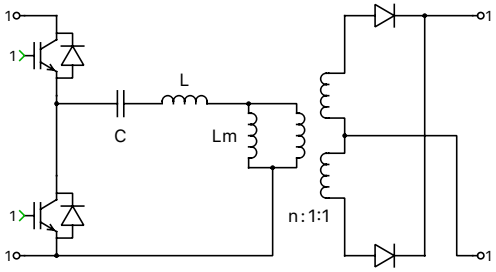
Purpose

Half-bridge LLC resonant converter, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated LLC resonant converter with a half bridge on the input side. For an LLC with a half-bridge synchronous output rectifier, refer to the Half-Bridge LLC Converter with Synchronous Rectification (p. 729) component.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.45 on p. 728.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Output rectifier
Use this setting to choose between center-tapped `Half bridge` and `Full bridge` output rectifier topology.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Resonant capacitance
A non-zero scalar for the resonant capacitance C on the primary side, in farads (F).

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.45 on p. 728 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Gate signals
A vector containing the gate signals of the active semiconductor switches.

↪ *Added in PLECS 4.9.*

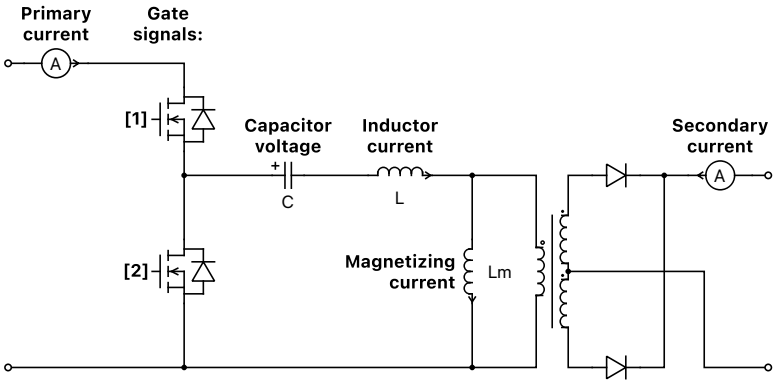


Fig. 16.45: Probes for the Half-Bridge LLC Converter

Half-Bridge LLC Converter with Synchronous Rectification

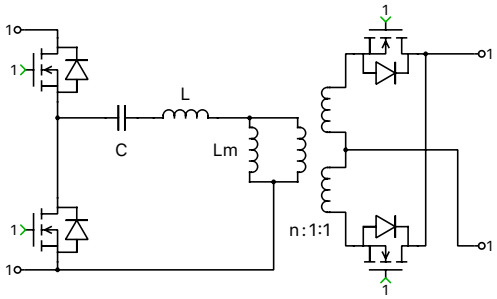
Purpose

Half-bridge LLC resonant converter with synchronous rectification, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated LLC resonant converter with a half bridge on the input side and a half bridge synchronous rectifier on the output side. To realize a full-bridge synchronous rectifier, use the CLLLC Converter (p. 607) component configured without secondary-side capacitance.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.46.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Resonant capacitance
A non-zero scalar for the resonant capacitance C on the primary side, in farads (F).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.46 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

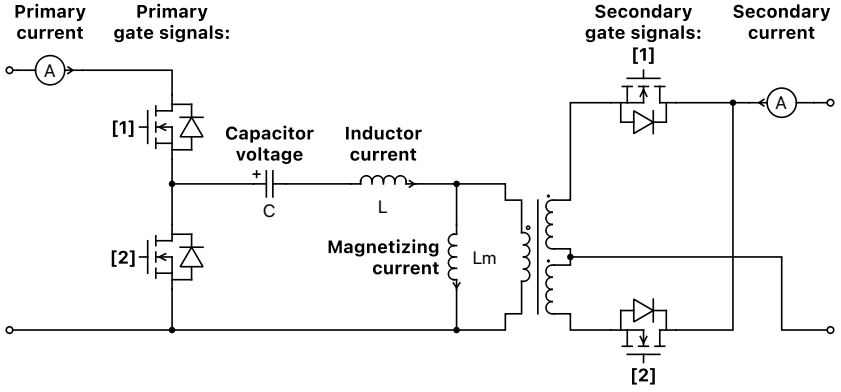


Fig. 16.46: Probe Signals for the Half-Bridge LLC Converter with Synchronous Rectifier

↪ *Added in PLECS 5.0.*

Half Bridge

Purpose

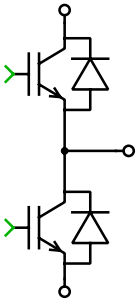
2-level half-bridge with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description

This Power Module implements a single leg of a 2-level voltage source inverter.



The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value 1 for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

Note

The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if the following condition is met:

- The sum of the gate signals for the upper and lower switches exceeds `1`.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Half Bridges (Low-/High-Side Connected)

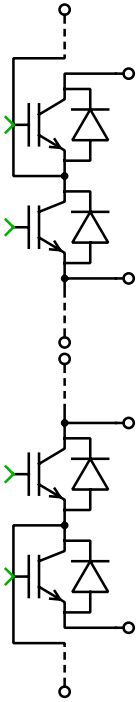
Purpose

Series-connected half-bridge inverter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



These Power Modules implement multiple series-connected inverter cells for use in modular multilevel converters. All control signals as well as the DC side terminals are vectorized with their width matching the number of cells.

The component offers two configurations:

`Sub-cycle average`

Each gate input is controlled by a floating-point number with values between `0` and `1`, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489). The DC side of the converter has current source behavior and must be connected to a positively biased capacitor or voltage source. The AC side is typically connected to an inductor.

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

`Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `non-zero`. For compatibility with the `Sub-cycle average` configuration it is recommended to use the value `1` for non-zero gate signals.

The `Switched` configuration is not recommended for real-time simulations (p. 360) on the RT Box (p. 361). It is intended for offline simulations (p. 361) that resolve each individual switching event and can serve as a reference for results obtained with the `Sub-cycle average` configuration.

> **Note**
>
> In `Sub-cycle average` configuration, the primary (DC) sides of the inverter cells must remain galvanically isolated from each other.

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. The corresponding restrictions are described under the **Assertions** parameter. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Configuration

Selects the component configuration. See the descriptions of `Sub-cycle average` and `Switched` above.

Semiconductor symbol

Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Number of cells

Number of series-connected half-bridge inverters.

Assertions

If the **Configuration** is set to `Sub-cycle average` and this parameter is set to `on`, the block checks the duty-cycle gate input signals for overlapping relative on-times. It issues a `Half-bridge shoot-through` error if the following condition is met:

- The sum of the gate signals for the upper and lower switches of the same half-bridge exceeds 1.

Switch model (CPU code generation)

If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Heat Flow Meter

Purpose

Output measured heat flow as signal

Library

Thermal / Meters

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 Signal Outport block (p. 952) or by dragging the component into the dialog box of a Probe block.

Probe Signals

Measured heat flow

The measured heat flow in watts (W).

Heat Sink

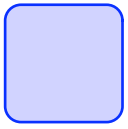
Purpose

Isotherm environment for placing components

Library

Thermal / Components

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 middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key. 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” (p. 171).

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 `0`.

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 degrees Celsius (°C). If left blank or if the value is `nan`, PLECS will initialize the value based on a thermal “DC” analysis, see “Temperature Initialization” (p. 177).

Width

This parameter allows you to set the width of the internal thermal capacitance and the external terminals. The default is `1`. A non-scalar heat sink can enclose only components that have the same width.

Probe Signals

Temperature

The temperature difference between the heat sink and the thermal reference, in degrees Celsius (°C).

Hit Crossing

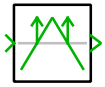
Purpose

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 / Components

Description



This component models a segment of a magnetic core. It establishes a nonlinear 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.

Fig. 16.47 shows a fully excited major hysteresis curve with some minor reversal loops. The major curve is defined by the saturation point (H_{sat}, B_{sat}), the coercitive field strength H_{c}, the remanence flux density B_{r} and the saturated permeability μ_{sat}.

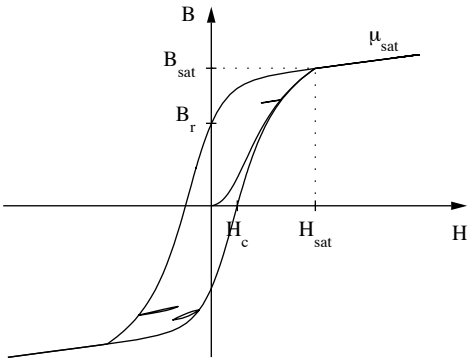


Fig. 16.47: Fully excited major hysteresis curve with some minor reversal loops

Parameters

Cross-sectional area
Cross-sectional area A of the flux path, in square meters (m^2).

Length of flux path
Length l of the flux path, in meters (m).

Coercitive field strength
Coercitive field strength H_{c} for $B = 0$, in amperes per meter $(\mathrm{A/m})$.

Remanence flux density
Remanence flux density B_{r} for $H = 0$, in teslas (T).

Saturation field strength
Field strength H_{sat} at the saturation point, in amperes per meter $(\mathrm{A/m})$.

Saturation flux density
Flux density B_{sat} at the saturation point, in teslas (T).

Saturated rel. permeability
Relative permeability $\mu_{\mathrm{r,sat}} = \mu_{\mathrm{sat}}/\mu_0$ of the core material for $H > H_{\mathrm{sat}}$.

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 amperes per meter $(\mathrm{A/m})$.

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)

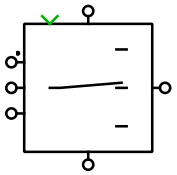
Purpose

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 (p. 1090). 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 in Fig. 16.48.

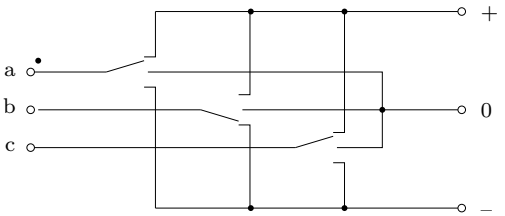


Fig. 16.48: Three-level converter circuit diagram

Ideal Converter (3ph)

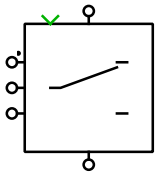
Purpose

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 (p. 668). 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 in Fig. 16.49.

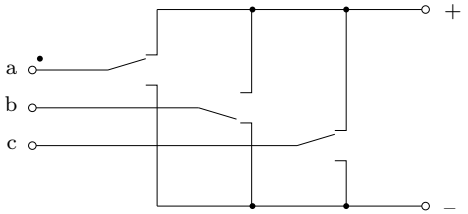


Fig. 16.49: Two-level converter circuit diagram

Ideal Transformer

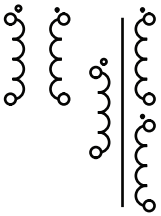
Purpose

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:

$$i_\mathrm{m} \cdot n_1 = i_1 \cdot n_1 + i_2 \cdot n_2 + i_3 \cdot n_3 + \dots$$

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

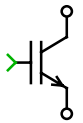
Ideal IGBT with optional forward voltage and on-resistance

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 (Ω). The default is `0`.

Initial conductivity

Initial conduction state of the IGBT. The IGBT is initially blocking if the parameter evaluates to zero, otherwise it is conducting.

Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (p. 171). If no thermal description is given, the losses are calculated based on the voltage drop $v_\mathrm{on} = V_\mathrm{f} + R_\mathrm{on} \cdot i$.

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in $(\mathrm{K/W})$. The default is `0`.

↪ *Added in PLECS 4.6.*

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” (p. 177).

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)

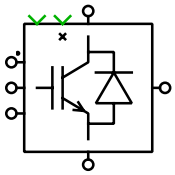
Purpose

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 topmost IGBT, connected to the positive dc level, is turned on if the corresponding gate signal is ≥ 1, and the second IGBT if the signal is ≥ 0. The third IGBT is turned on for signals ≤ 0 and the lowest one for signals ≤ -1. Gate signal values of 1, 0 and -1 connect the phase output to the positive, neutral and negative dc level. By applying a non-zero signal at the inhibit input marked with "x" you can turn off all IGBTs.

You can choose between two different converter models:

- The basic **IGBT 3-Level Converter** is modeled using the component IGBT with Diode (p. 752). No parameters can be entered.
- The **IGBT 3-Level Converter with Parasitics** is based on individual IGBT (p. 746) and Diode (p. 641) 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 (p. 752), 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 in Fig. 16.50.

Parameters

For a description of the parameters see the documentation of the IGBT with Diode (p. 752), the IGBT (p. 746) and the Diode (p. 641).

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.

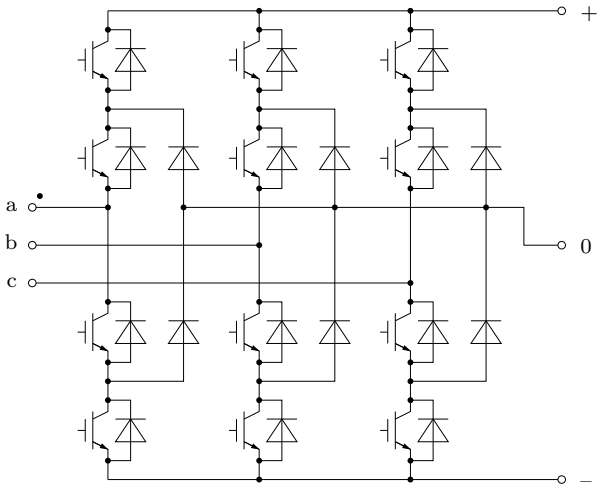


Fig. 16.50: Electrical circuit for IGBT 3-Level Converter

For the **IGBT 3-Level Converter with Parasitics** the diode probe signal vectors are in the order: anti-parallel diodes (top-to-bottom), clamping diodes (top-to-bottom).

IGBT Converter (3ph)

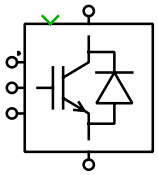
Purpose

3-phase IGBT converter

Library

Electrical / Converters

Description



Implements a three-phase two-level IGBT converter with reverse diodes. The gate input is a vector of three signals – one per leg. The upper IGBT, connected to the positive dc level, is on if the corresponding gate signal is positive. The lower IGBT is on if the gate signal is negative. If the gate signal is zero, both IGBTs in the leg are switched off.

You can choose between two different converter models:

- The basic **IGBT Converter** is modeled using the component IGBT with Diode (p. 752). 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 (p. 746) and Diode (p. 641) 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 (p. 752), 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 in Fig. 16.51.

Parameters

For a description of the parameters see the documentation of the IGBT with Diode (p. 752), the IGBT (p. 746) and the Diode (p. 641).

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-.

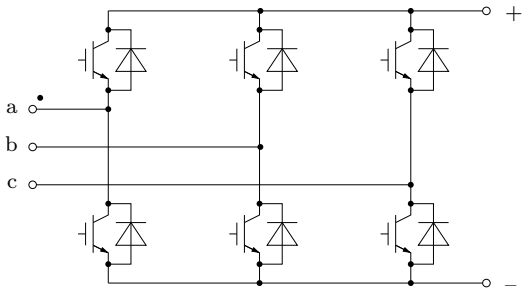


Fig. 16.51: Electrical circuit for IGBT Converter (3ph)

IGBT with Diode

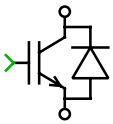
Purpose

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 (p. 746) with an individual anti-parallel Diode (p. 641), 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” (p. 171) and “Losses of Semiconductor Switch with Diode” (p. 206).

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in (K/W). The default is `0`.

↪ *Added in PLECS 4.6.*

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" (p. 177).

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

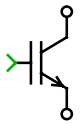
Purpose

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 (p. 746) 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 in Fig. 16.52. The free-wheeling diode for the inductive load is modeled with reverse recovery (p. 646).

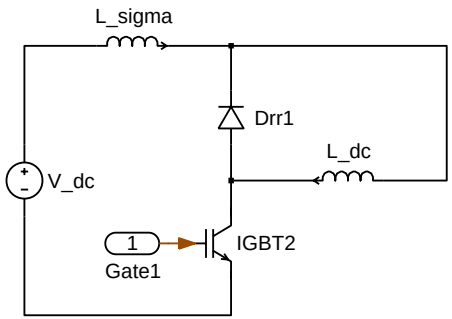


Fig. 16.52: Test circuit for IGBT with Limited di/dt

The diagram in Fig. 16.53 shows the collector current $i_{\mathrm{C}}(t)$ of the IGBT and the resulting collector-emitter voltage $v_{\mathrm{CE}}(t)$ during switching.

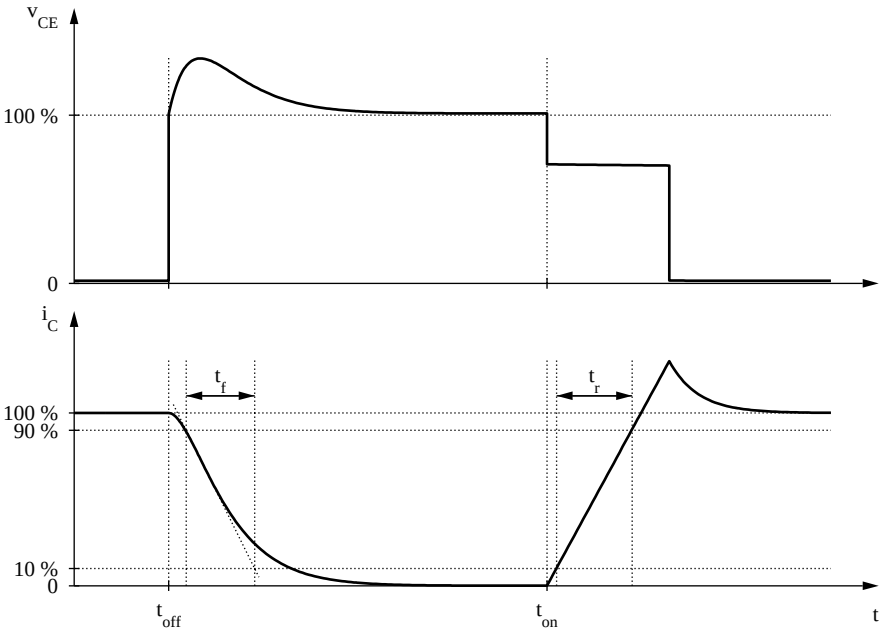


Fig. 16.53: Collector current and collector-emitter voltage

At $t = t_{\mathrm{off}}$ the gate signal becomes zero, and the device current i_{C} begins to fall. The current slope follows an aperiodic oscillation

$$i_{\mathrm{C}}(t) = i_{\mathrm{C}}(t_{\mathrm{off}}) \left[e^{-\frac{2.4\,(t - t_{\mathrm{off}})}{t_{\mathrm{f}}}} \left(1 + \frac{2.4\,(t - t_{\mathrm{off}})}{t_{\mathrm{f}}} \right) \right]$$

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 $t = t_{\mathrm{on}}$ 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:

$$\frac{\mathrm{d}i_\mathrm{max}}{\mathrm{d}t} = 0.8 \cdot \frac{I_\mathrm{C}}{t_\mathrm{r}}$$

The diagram in Fig. 16.54 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.

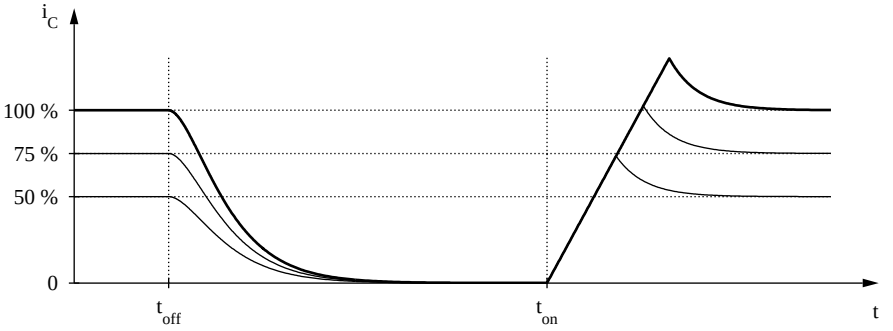


Fig. 16.54: Collector current transients for different on-state currents

Parameters

Blocking voltage
Maximum voltage V_CES in volts (V) that under any conditions should be applied between collector and emitter.

Continuous collector current
Maximum dc current I_C in amperes (A) that the IGBT can conduct.

Forward voltage
Additional dc voltage V_f in volts (V) between collector and emitter when the IGBT is conducting. The default is `0`.

On-resistance
The resistance R_on of the conducting device, in ohms (Ω). The default is `0`.

Off-resistance
The resistance R_off of the blocking device, in ohms (Ω). The default is `1e6`. This parameter may be set to `inf` unless multiple IGBTs are connected in series.

Rise time
Time t_r in seconds (s) 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 (s) between instants when the collector current has dropped from $90\,\%$ to $10\,\%$ of its initial value along an extrapolated straight line tangent to the maximum rate-of-change of the current (see figure above).

Stray inductance
Internal inductance L_σ in henries (H) measured between the collector and emitter terminals.

Initial current
The initial current through the component at simulation start, in amperes (A). The default is `0`.

Probe Signals

IGBT voltage
The voltage measured between collector and emitter.

IGBT current
The current through the IGBT flowing from collector to emitter.

IGBT conductivity
Conduction state of the internal switch. The signal outputs 0 when the IGBT is blocking, and 1 when it is conducting.

IGCT (Reverse Blocking)

Purpose

Ideal IGCT with optional forward voltage and on-resistance

Library

Electrical / Power Semiconductors

Description



The Integrated Gate Commutated Thyristor is a semiconductor switch that is controlled via the external gate. It conducts a current from anode to cathode only if the gate signal is not zero.

Parameters

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 (Ω). 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" (p. 171). If no thermal description is given, the losses are calculated based on the voltage drop $v_\mathrm{on} = V_\mathrm{f} + R_\mathrm{on} \cdot i$.

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in $(\mathrm{K/W})$. The default is `0`.

↪ *Added in PLECS 4.6.*

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" (p. 177).

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” (p. 171) and “Losses of Semiconductor Switch with Diode” (p. 206).

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in (K/W). The default is `0`.

↪ *Added in PLECS 4.6.*

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” (p. 177).

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 (Open Stator Windings)

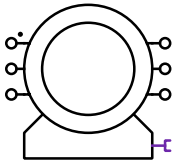
Purpose

Non-saturable induction machine with squirrel-cage rotor and open stator windings

Library

Electrical / Machines

Description



This model of a squirrel-cage induction machine can only be used with the continuous state-space method. The machine model is based on a stationary reference frame (Clarke transformation). A sophisticated implementation of the Clarke transformation facilitates the connection of external inductances in series with the stator windings.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, the positive terminal of phase a of the stator windings is marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit > Subsystem** menu or the block's context menu.

Electrical System

The equivalent electrical circuits for the d, q and 0 axes are shown in Fig. 16.55 on p. 764.

The rotor flux is computed as:

$$\Psi_{\mathrm{r,d}} = L'_{\mathrm{lr}}\, i'_{\mathrm{r,d}} + L_{\mathrm{m}}\,(i_{\mathrm{s,d}} + i'_{\mathrm{r,d}})$$

$$\Psi_{\mathrm{r,q}} = L'_{\mathrm{lr}}\, i'_{\mathrm{r,q}} + L_{\mathrm{m}}\,(i_{\mathrm{s,q}} + i'_{\mathrm{r,q}})$$

The three-phase voltages $v_{\mathrm{s,a}}$, $v_{\mathrm{s,b}}$ and $v_{\mathrm{s,c}}$ across the individual stator windings are transformed into dq0 quantities:

$$\begin{bmatrix} v_{\mathrm{s,d}} \\ v_{\mathrm{s,q}} \\ v_{\mathrm{s,0}} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} v_{\mathrm{s,a}} \\ v_{\mathrm{s,b}} \\ v_{\mathrm{s,c}} \end{bmatrix}$$

Likewise, the stator currents in the stationary reference frame are transformed back into three-phase currents:

$$\begin{bmatrix} i_{\mathrm{s,a}} \\ i_{\mathrm{s,b}} \\ i_{\mathrm{s,c}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 \end{bmatrix} \begin{bmatrix} i_{\mathrm{s,d}} \\ i_{\mathrm{s,q}} \\ i_{\mathrm{s,0}} \end{bmatrix}$$

Electro-Mechanical System

Electromagnetic torque:

$$T_{\mathrm{e}} = \frac{3}{2}\, p\, L_{\mathrm{m}} (i_{\mathrm{s,q}}\, i'_{\mathrm{r,d}} - i_{\mathrm{s,d}}\, i'_{\mathrm{r,q}})$$

Mechanical System

Mechanical rotor speed ω_{m}:

$$\dot{\omega}_{\mathrm{m}} = \frac{1}{J}(T_{\mathrm{e}} - F\omega_{\mathrm{m}} - T_{\mathrm{m}})$$

$$\omega_{\mathrm{e}} = p\, \omega_{\mathrm{m}}$$

Mechanical rotor angle θ_{m}:

$$\dot{\theta}_{\mathrm{m}} = \omega_{\mathrm{m}}$$

$$\theta_{\mathrm{e}} = p\, \theta_{\mathrm{m}}$$

Parameters

Most parameters for the Induction Machine (Slip Ring) (p. 765) are also applicable for this machine. Only the following parameter differs:

Initial stator currents

A three-element vector containing the initial stator currents $i_{\mathrm{s,a,0}}$, $i_{\mathrm{s,b,0}}$ and $i_{\mathrm{s,c,0}}$ of phase a, b and c in amperes (A).

Probe Signals

Most probe signals for the Induction Machine (Slip Ring) (p. 765) are also available with this machine. Only the following probe signal is different:

Rotor currents

The rotor currents $i'_{\mathrm{r,d}}$ and $i'_{\mathrm{r,q}}$ in the stationary reference frame in amperes (A), referred to the stator side.

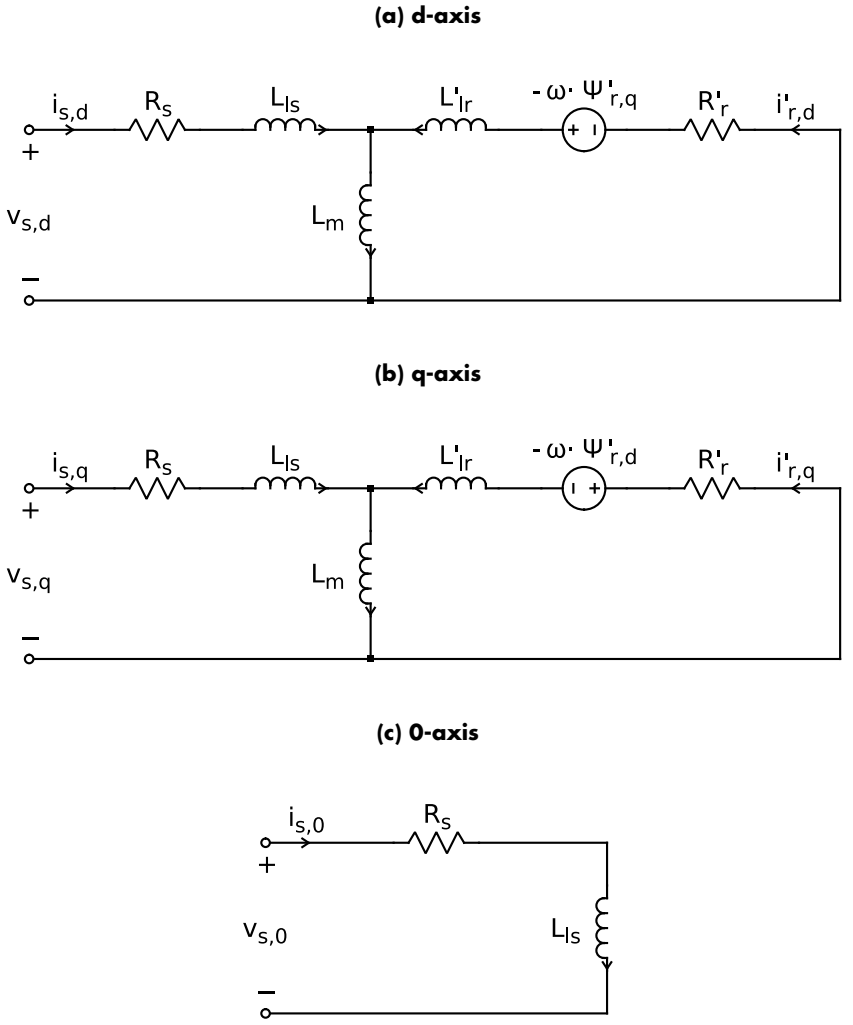


Fig. 16.55: Equivalent electrical circuits of the d, q and 0 axes of an open stator winding machine.

Induction Machine (Slip Ring)

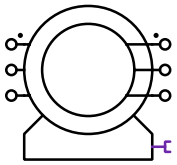
Purpose

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 (p. 773).

The machine model is based on a stationary reference frame (Clarke transformation). A sophisticated implementation of the Clarke transformation facilitates the connection of external inductances in series with the stator windings. However, external inductors cannot be connected to the rotor windings due to the current sources in the model. In this case, external inductors must be included in the leakage inductance of the rotor.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator and rotor windings is marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit > Subsystem** menu or the block's context 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 equivalent electrical circuits for the d and q axes are shown in Fig. 16.56.

The rotor flux is computed as:

$$\Psi_{\mathrm{r,d}} = L'_{\mathrm{lr}}\, i'_{\mathrm{r,d}} + L_{\mathrm{m}}\,(i_{\mathrm{s,d}} + i'_{\mathrm{r,d}})$$

$$\Psi_{\mathrm{r,q}} = L'_{\mathrm{lr}}\, i'_{\mathrm{r,q}} + L_{\mathrm{m}}\,(i_{\mathrm{s,q}} + i'_{\mathrm{r,q}})$$

The three-phase voltages $v_{\mathrm{s,ab}}$ and $v_{\mathrm{s,bc}}$ at the stator terminals are transformed into dq quantities:

$$\begin{bmatrix} v_{\mathrm{s,d}} \\ v_{\mathrm{s,q}} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} v_{\mathrm{s,ab}} \\ v_{\mathrm{s,bc}} \end{bmatrix}$$

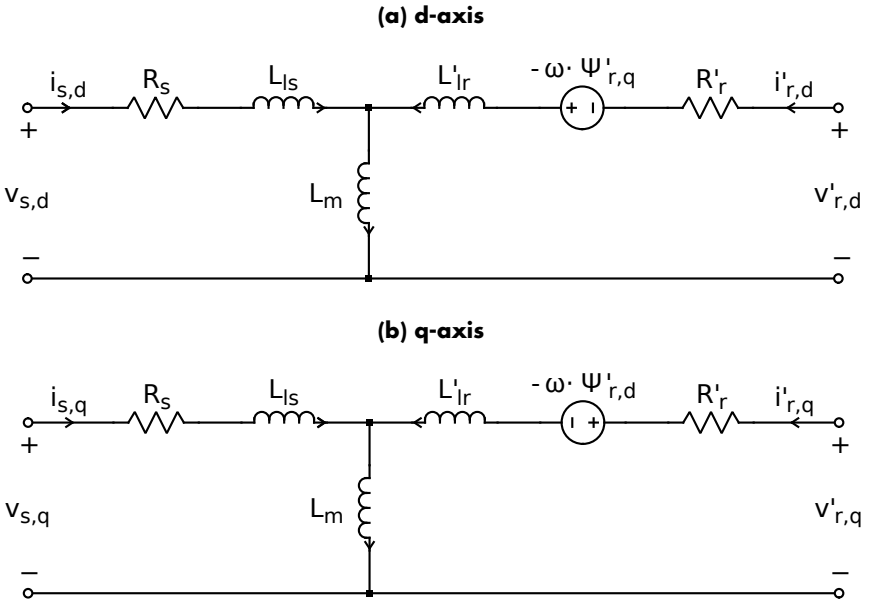


Fig. 16.56: Equivalent electrical circuits of the d and q axes of a slip ring induction machine.

Likewise, the stator currents in the stationary reference frame are transformed back into three-phase currents:

$$\begin{bmatrix} i_{\mathrm{s,a}} \\ i_{\mathrm{s,b}} \\ i_{\mathrm{s,c}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_{\mathrm{s,d}} \\ i_{\mathrm{s,q}} \end{bmatrix}$$

Similar equations apply to the voltages and currents at the rotor terminals with θ being the electrical rotor position:

$$\begin{bmatrix} v'_{\mathrm{r,d}} \\ v'_{\mathrm{r,q}} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & -\cos(\theta - \frac{2\pi}{3}) \\ \sin(\theta) & -\sin(\theta - \frac{2\pi}{3}) \end{bmatrix} \begin{bmatrix} v'_{\mathrm{r,ab}} \\ v'_{\mathrm{r,bc}} \end{bmatrix}$$

$$\begin{bmatrix} i'_{\mathrm{r,a}} \\ i'_{\mathrm{r,b}} \\ i'_{\mathrm{r,c}} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \cos(\theta + \frac{2\pi}{3}) & \sin(\theta + \frac{2\pi}{3}) \\ \cos(\theta - \frac{2\pi}{3}) & \sin(\theta - \frac{2\pi}{3}) \end{bmatrix} \begin{bmatrix} i'_{\mathrm{r,d}} \\ i'_{\mathrm{r,q}} \end{bmatrix}$$

Electro-Mechanical System

Electromagnetic torque:

$$T_{\mathrm{e}} = \frac{3}{2}\, p\, L_{\mathrm{m}} (i_{\mathrm{s,d}}\, i'_{\mathrm{r,q}} - i_{\mathrm{s,q}}\, i'_{\mathrm{r,d}})$$

Mechanical System

Mechanical rotor speed ω_{m}:

$$\dot{\omega}_{\mathrm{m}} = \frac{1}{J}(T_{\mathrm{e}} - F\omega_{\mathrm{m}} - T_{\mathrm{m}})$$

$$\omega_{\mathrm{e}} = p\, \omega_{\mathrm{m}}$$

Mechanical rotor angle θ_{m}:

$$\dot{\theta}_{\mathrm{m}} = \omega_{\mathrm{m}}$$

$$\theta_{\mathrm{e}} = p\, \theta_{\mathrm{m}}$$

Parameters

Stator resistance
Stator winding resistance R_{s} in ohms (Ω).

Stator leakage inductance
Stator leakage inductance L_{ls} in henries (H).

Rotor resistance
Rotor winding resistance R'_{r} in ohms (Ω), referred to the stator side.

Rotor leakage inductance
Rotor leakage inductance L'_{lr} in henries (H), referred to the stator side.

Magnetizing inductance
Magnetizing inductance L_{m} in henries (H), referred to the stator side.

Inertia
Combined rotor and load inertia J in (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 $\omega_{\mathrm{m},0}$ in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Initial rotor position
Initial mechanical rotor angle $\theta_{\mathrm{m},0}$ in radians. If $\theta_{\mathrm{m},0}$ 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 $i_{\mathrm{s,a},0}$ and $i_{\mathrm{s,b},0}$ of phases a and b in amperes (A).

Initial stator flux
A two-element vector containing the initial stator flux $\Psi'_{\mathrm{s,d},0}$ and $\Psi'_{\mathrm{s,q},0}$ in the stationary reference frame in (Vs).

Probe Signals

Stator phase currents
The three-phase stator winding currents $i_{\mathrm{s,a}}$, $i_{\mathrm{s,b}}$ and $i_{\mathrm{s,c}}$, in amperes (A). Currents flowing into the machine are considered positive.

Rotor phase currents
The three-phase rotor winding currents $i'_{\mathrm{r,a}}$, $i'_{\mathrm{r,b}}$ and $i'_{\mathrm{r,c}}$ in amperes (A), referred to the stator side. Currents flowing into the machine are considered positive.

Stator flux (dq)
The stator flux linkages $\Psi_{\mathrm{s,d}}$ and $\Psi_{\mathrm{s,q}}$ in the stationary reference frame in weber (Wb):

$$\Psi_{\mathrm{s,d}} = L_{\mathrm{ls}}\, i_{\mathrm{s,d}} + L_{\mathrm{m}}\,(i_{\mathrm{s,d}} + i'_{\mathrm{r,d}})$$

$$\Psi_{\mathrm{s,q}} = L_{\mathrm{ls}}\, i_{\mathrm{s,q}} + L_{\mathrm{m}}\,(i_{\mathrm{s,q}} + i'_{\mathrm{r,q}})$$

Magnetizing flux (dq)
The magnetizing flux linkages $\Psi_{\mathrm{m,d}}$ and $\Psi_{\mathrm{m,q}}$ in the stationary reference frame in (Vs):

$$\Psi_{\mathrm{m,d}} = L_{\mathrm{m}}\,(i_{\mathrm{s,d}} + i'_{\mathrm{r,d}})$$

$$\Psi_{\mathrm{m,q}} = L_{\mathrm{m}}\,(i_{\mathrm{s,q}} + i'_{\mathrm{r,q}})$$

Rotor flux (dq)

The rotor flux linkages $\Psi'_{\mathrm{r,d}}$ and $\Psi'_{\mathrm{r,q}}$ in the stationary reference frame in (Vs).

Rotational speed

The rotational speed ω_{m} of the rotor in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Rotor position

The mechanical rotor angle θ_{m} in radians.

Electrical torque

The electrical torque T_{e} of the machine in (Nm).

Induction Machine (Squirrel Cage)

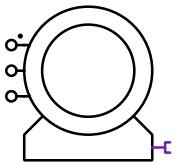
Purpose

Non-saturable induction machine with squirrel-cage rotor

Library

Electrical / Machines

Description



The machine model is based on a stationary reference frame (Clarke transformation). A sophisticated implementation of the Clarke transformation facilitates the connection of external inductances in series with the stator windings.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator winding is marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit > Subsystem** menu or the block's context menu.

Electrical System

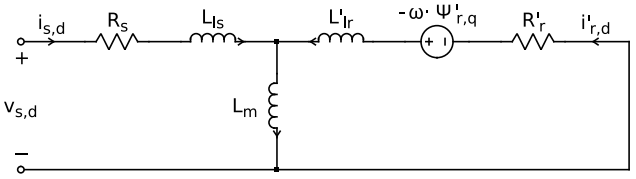


Fig. 16.57: d-axis

The rotor flux is computed as:

$$\Psi_{\mathrm{r,d}} = L'_{\mathrm{lr}}\, i'_{\mathrm{r,d}} + L_{\mathrm{m}}\,(i_{\mathrm{s,d}} + i'_{\mathrm{r,d}})$$

$$\Psi_{\mathrm{r,q}} = L'_{\mathrm{lr}}\, i'_{\mathrm{r,q}} + L_{\mathrm{m}}\,(i_{\mathrm{s,q}} + i'_{\mathrm{r,q}})$$

The three-phase voltages $v_{\mathrm{s,ab}}$ and $v_{\mathrm{s,bc}}$ at the stator terminals are transformed into dq quantities:

$$\begin{bmatrix} v_{\mathrm{s,d}} \\ v_{\mathrm{s,q}} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} v_{\mathrm{s,ab}} \\ v_{\mathrm{s,bc}} \end{bmatrix}$$

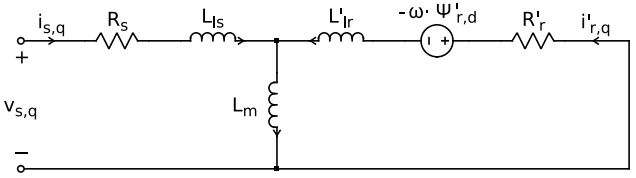


Fig. 16.58: q-axis

Likewise, the stator currents in the stationary reference frame are transformed back into three-phase currents:

$$\begin{bmatrix} i_{\mathrm{s,a}} \\ i_{\mathrm{s,b}} \\ i_{\mathrm{s,c}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_{\mathrm{s,d}} \\ i_{\mathrm{s,q}} \end{bmatrix}$$

Electro-Mechanical System

Electromagnetic torque:

$$T_{\mathrm{e}} = \frac{3}{2}\, p\, L_{\mathrm{m}} (i_{\mathrm{s,q}}\, i'_{\mathrm{r,d}} - i_{\mathrm{s,d}}\, i'_{\mathrm{r,q}})$$

Mechanical System

Mechanical rotor speed ω_{m}:

$$\dot{\omega}_{\mathrm{m}} = \frac{1}{J}(T_{\mathrm{e}} - F\omega_{\mathrm{m}} - T_{\mathrm{m}})$$

$$\omega_{\mathrm{e}} = p\, \omega_{\mathrm{m}}$$

Mechanical rotor angle θ_{m}:

$$\dot{\theta}_{\mathrm{m}} = \omega_{\mathrm{m}}$$

$$\theta_{\mathrm{e}} = p\, \theta_{\mathrm{m}}$$

Parameters

Same as for the Induction Machine (Slip Ring) (p. 765).

Probe Signals

Most probe signals for the Induction Machine (Slip Ring) (p. 765) are also available with this squirrel-cage machine. Only the following probe signal is different:

Rotor currents

The rotor currents $i'_{\mathrm{r,d}}$ and $i'_{\mathrm{r,q}}$ in the stationary reference frame in amperes (A), referred to the stator side.

Induction Machine with Saturation

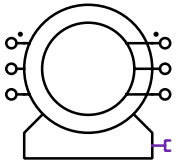
Purpose

Induction machine with slip-ring rotor and main-flux saturation

Library

Electrical / Machines

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

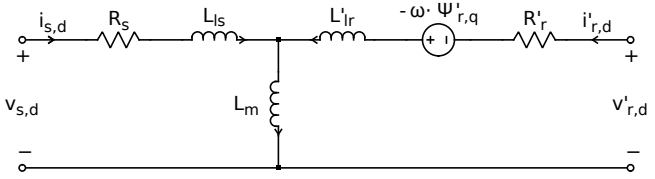


Fig. 16.59: d-axis

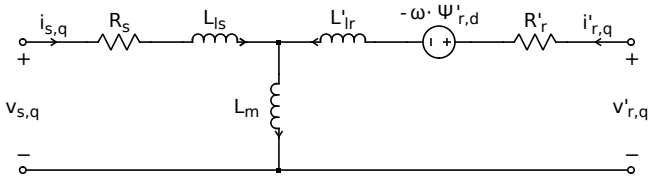


Fig. 16.60: q-axis

The rotor flux is defined as:

$$\Psi_{\mathrm{r,d}} = L'_{\mathrm{lr}}\, i'_{\mathrm{r,d}} + L_{\mathrm{m}}\,(i_{\mathrm{s,d}} + i'_{\mathrm{r,d}})$$

$$\Psi_{\mathrm{r,q}} = L'_{\mathrm{l,r}}\, i'_{\mathrm{r,q}} + L_{\mathrm{m}}\,(i_{\mathrm{s,q}} + i'_{\mathrm{r,q}})$$

The machine model offers two different implementations of the electrical system: a traditional stationary reference frame and a voltage behind reactance formulation.

Stationary Reference Frame

This implementation is based on machine equations in the stationary reference frame (Clarke transformation). Constant coefficients in the stator and rotor equations make the model numerically efficient. However, interfacing the reference frame with the external 3-phase network may be difficult. Since the coordinate transformations are based on voltage-controlled current sources inductors and naturally commutated devices such as diode rectifiers may not be directly connected to the stator terminals. In these cases, fictitious RC snubbers are required to create the necessary voltages across the terminals. The implementation can be used with both the continuous and the discrete state-space method.

Voltage Behind Reactance

This formulation allows for direct interfacing of arbitrary external networks with the 3-phase stator terminals. The rotor dynamics are expressed using explicit state-variable equations while the stator branch equations are described in circuit form. However, due to the resulting time-varying inductance matrices, this implementation is numerically less efficient than the traditional reference frame.

↪ *Added in PLECS 4.0.*

In both implementations, the value of the main flux inductances $L_{\mathrm{m,d}}$ and $L_{\mathrm{m,q}}$ are not constant but depend on the main flux linkage Ψ_{m} as illustrated in the $\Psi_{\mathrm{m}}/i_{\mathrm{m}}$ diagram.

For flux linkages far below the transition flux Ψ_{T}, the relationship between flux and current is almost linear and is determined by the unsaturated magnetizing inductance $L_{\mathrm{m,0}}$. For large flux linkages the relationship is governed by the saturated magnetizing inductance $L_{\mathrm{m,sat}}$. Ψ_{T} defines the knee of the transition between unsaturated and saturated main flux inductance. The tightness of the transition is defined with the form factor f_{T}. If you do not have detailed information about the saturation characteristic of your machine, $f_{\mathrm{T}} = 1$ 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

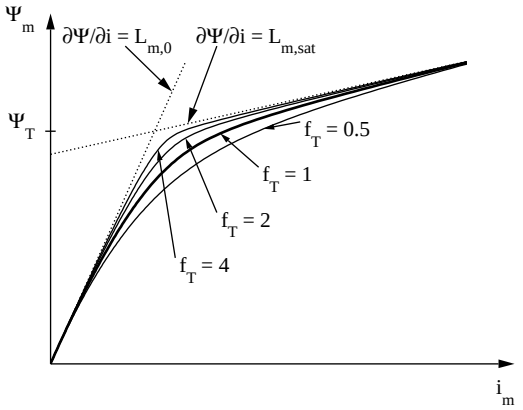


Fig. 16.61: Magnetic saturation characteristic

in both axes. In the implementation, the stator currents and the main flux linkage are chosen as state variables. With this type of model, the representation of dynamic cross-saturation can be neglected without affecting the machine's performance. The computation of the time derivative of the main flux inductance is not required.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit > Subsystem** menu or the block's context 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:

$$T_\mathrm{e} = \frac{3}{2}\, p(i_\mathrm{s,q}\, \Psi_\mathrm{s,d} - i_\mathrm{s,d}\, \Psi_\mathrm{s,q})$$

Mechanical System

Mechanical rotor speed ω_m:

$$\dot{\omega}_\mathrm{m} = \frac{1}{J}(T_\mathrm{e} - F\omega_\mathrm{m} - T_\mathrm{m})$$

$$\omega = p\,\omega_\mathrm{m}$$

Mechanical rotor angle θ_m:

$$\dot{\theta}_\mathrm{m} = \omega_\mathrm{m}$$

$$\theta_\mathrm{e} = p\,\theta_\mathrm{m}$$

Parameters

Model
Implementation in the stationary reference frame or as a voltage behind reactance.

Stator resistance
Stator winding resistance R_s in ohms (Ω).

Stator leakage inductance
Stator leakage inductance L_ls in henries (H).

Rotor resistance
Rotor winding resistance R'_r in ohms (Ω), referred to the stator side.

Rotor leakage inductance
Rotor leakage inductance L'_lr in henries (H), referred to the stator side.

Unsaturated magnetizing inductance
Unsaturated main flux inductance $L_\mathrm{m,0}$, in henries (H), referred to the stator side.

Saturated magnetizing inductance
Saturated main flux inductance $L_\mathrm{m,sat}$ in henries (H), referred to the stator side. If you do not want to model saturation, set $L_\mathrm{m,sat} = L_\mathrm{m,0}$.

Magnetizing flux at saturation transition
Transition flux linkage Ψ_T, in (Vs), defining the knee between unsaturated and saturated main flux inductance.

Tightness of saturation transition
Form factor f_T defining the tightness of the transition between unsaturated and saturated main flux inductance. The default is 1.

Inertia
Combined rotor and load inertia J in (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 $\omega_\mathrm{m,0}$ in radians per second $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Initial rotor position
Initial mechanical rotor angle $\theta_{\mathrm{m},0}$ in radians. If $\theta_{\mathrm{m},0}$ 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 $i_{\mathrm{s,a},0}$ and $i_{\mathrm{s,b},0}$ of phases a and b in amperes (A).

Initial stator flux
A two-element vector containing the initial stator flux $\Psi_{\mathrm{s,d},0}$ and $\Psi_{\mathrm{s,q},0}$ in the stationary reference frame in (Vs).

Probe Signals

Stator phase currents
The three-phase stator winding currents $i_{\mathrm{s,a}}$, $i_{\mathrm{s,b}}$ and $i_{\mathrm{s,c}}$, in amperes (A). Currents flowing into the machine are considered positive.

Rotor phase currents
The three-phase rotor winding currents $i'_{\mathrm{r,a}}$, $i'_{\mathrm{r,b}}$ and $i'_{\mathrm{r,c}}$ in amperes (A), referred to the stator side. Currents flowing into the machine are considered positive.

Stator flux (dq)
The stator flux linkages $\Psi_{\mathrm{s,d}}$ and $\Psi_{\mathrm{s,q}}$ in the stationary reference frame in (Vs).

Magnetizing flux (dq)
The magnetizing flux linkages $\Psi_{\mathrm{m,d}}$ and $\Psi_{\mathrm{m,q}}$ in the stationary reference frame in (Vs).

Rotor flux (dq)
The rotor flux linkages $\Psi'_{\mathrm{r,d}}$ and $\Psi'_{\mathrm{r,q}}$ in the stationary reference frame in (Vs), referred to the stator side.

Rotational speed
The rotational speed ω_{m} of the rotor in radians per second $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Rotor position
The mechanical rotor angle θ_{m} in radians.

Electrical torque
The electrical torque T_{e} of the machine in (Nm).

References

- D. C. Aliprantis, O. 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:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} L_1 & 0 & \cdots & 0 \\ 0 & L_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L_n \end{bmatrix} \begin{bmatrix} \frac{d}{dt} i_1 \\ \frac{d}{dt} i_2 \\ \vdots \\ \frac{d}{dt} i_n \end{bmatrix}$$

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:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} L_1 & M_{1,2} & \cdots & M_{1,n} \\ M_{2,1} & L_2 & \cdots & M_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & L_n \end{bmatrix} \begin{bmatrix} \frac{d}{dt} i_1 \\ \frac{d}{dt} i_2 \\ \vdots \\ \frac{d}{dt} i_n \end{bmatrix}$$

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 (p. 744).

The relationship between the coupling factor $k_{i,j}$ and the mutual inductance $M_{i,j}$ is

$$M_{i,j} = M_{j,i} = k_{i,j}\sqrt{L_i \cdot L_j}$$

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

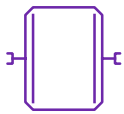
Purpose

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 τ_1 and τ_2 are the torques acting at the two flanges *towards* the body.

Parameters

Moment of inertia

The moment of inertia J, in $\left(\frac{\mathrm{kg\cdot m^2}}{\mathrm{rad^2}}\right)$.

> **Note**
>
> `Radian` is a dimensionless unit, therefore, moment of inertia values in $(\mathrm{kg\cdot m^2})$ and $(\mathrm{N\cdot m\cdot s^2})$ are also consistent with this definition.

Initial speed

The initial angular speed ω_0, in $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Initial angle

The initial angle θ_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 π).

Initial Condition

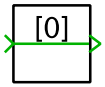
Purpose

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 “Block Sorting” (p. 36) for more information on algebraic loops.

> **Note**
>
> If a simulation is run from a saved simulation state (see “Scripted Simulation and Analysis Options” (p. 329)), the Initial Condition block immediately outputs the input value. The **Initial value** parameter is not used in this case.
>
> ↪ *Added in PLECS 4.7.*

Parameters

Initial value

The initial output value in the first simulation step.

Integrator

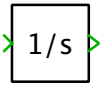
Purpose

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 integrator state may saturate at an upper and lower limit, or it can be wrapped between two limits. 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 behavior 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.

Enable wrapping
When set to `on`, the integrator state is wrapped between the **Upper wrapping limit** and **Lower wrapping limit** parameters. The default is `off`. Note that wrapping is mutually exclusive with saturation.

↪ *Added in PLECS 4.5.*

Upper saturation limit
An upper limit for the integrator state. If the value is `inf`, the state is unlimited. This parameter is visible only if **Enable wrapping** is set to `off`.

Lower saturation limit
A lower limit for the integrator state. If the value is `-inf`, the state is unlimited. This parameter is visible only if **Enable wrapping** is set to `off`.

Upper wrapping limit
The upper wrapping limit for the integrator state. When the state exceeds the upper limit, it is wrapped to the lower limit. This parameter is visible only if **Enable wrapping** is set to `on`.

Lower wrapping limit
The lower wrapping limit for the integrator state. When the state exceeds the lower limit, it is wrapped to the upper limit. This parameter is visible only if **Enable wrapping** is set to `on`.

Probe Signals

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

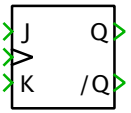
Purpose

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	$/Q_{\text{prev}}$	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. they 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.

/Q

The output signals $/Q$.

Leakage Flux Path

Purpose

Permeance of linear leakage flux path

Library

Magnetic / Components

Description



This component models a magnetic leakage flux path. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force $\mathcal{F}$. Magnetic permeance $\mathcal{P}$ is the reciprocal of magnetic reluctance $\mathcal{R}$:

$$\mathcal{P} = \frac{1}{\mathcal{R}} = \frac{\Phi}{\mathcal{F}}$$

This component is equivalent to the Magnetic Permeance (p. 795). 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 / Components

Description

This component models a segment of a magnetic core. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force $\mathcal{F}$



$$\frac{\Phi}{\mathcal{F}} = \frac{\mu_0 \mu_\mathrm{r} A}{l}$$

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

Parameters

Cross-sectional area
Cross-sectional area A of the flux path, in square meters $(\mathrm{m^2})$.

Length of flux path
Length l of the flux path, in meters (m).

Rel. permeability
Relative permeability μ_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 amperes per meter $(\mathrm{A/m})$.

Flux density
The magnetic flux density B in the core element, in teslas (T).

Linear Transformer (2 Windings)

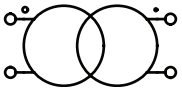
Purpose

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 magnetizing 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 in Fig. 16.62.

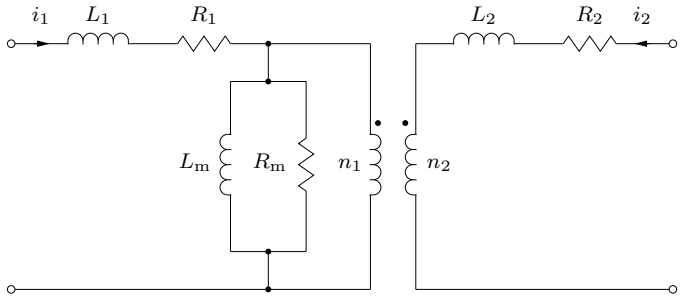


Fig. 16.62: Electrical circuit of a linear transformer (2 windings)

In the transformer symbol, the primary side winding is marked with a little circle. The secondary side winding is marked with a dot.

Parameters

Leakage inductance
A two-element vector containing the leakage inductance of the primary side L_1 and the secondary side L_2. The inductivity is given in henries (H).

Winding resistance
A two-element vector containing the resistance of the primary winding R_1 and the secondary winding R_2, in ohms (Ω).

Winding ratio
The ratio n_1/n_2 between the number of turns of the primary and secondary winding.

Magnetizing inductance
The magnetizing inductance L_m, in henries (H). The value is referred to the primary side.

Core loss resistance

An equivalent resistance R_{m} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

Initial current

A two-element vector containing the initial currents on the primary side i_1 and the secondary side i_2, in amperes (A). The currents are considered positive if flowing into the transformer at the marked terminals. The default is `[0 0]`.

Linear Transformer (3 Windings)

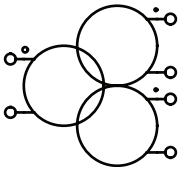
Purpose

Single-phase transformer with winding resistance and optional core loss

Library

Electrical / Transformers

Description



This transformer models three coupled windings on the same core. The magnetizing 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 in Fig. 16.63.

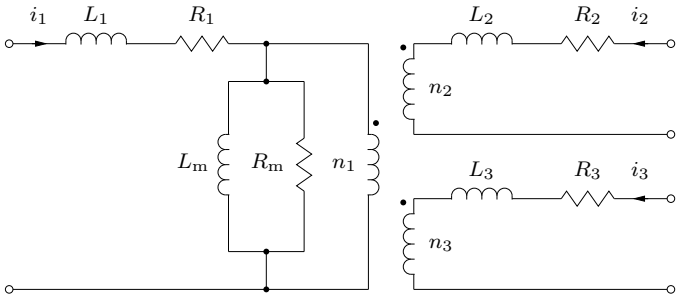


Fig. 16.63: Electrical circuit of a linear transformer (3 windings)

In the transformer symbol, the primary side winding is marked with a little circle. The secondary winding is marked with a dot at the outside terminal, the tertiary winding with a dot at the inside terminal.

Parameters

Leakage inductance

A three-element vector containing the leakage inductance of the primary side L_1, the secondary side L_2 and the tertiary side L_3. The inductivity is given in henries (H).

Winding resistance

A three-element vector containing the resistance of the primary winding R_1, the secondary winding R_2 and the tertiary winding R_3, in ohms (Ω).

No.of turns

A three-element vector containing the number of turns of the primary winding n_1, the secondary winding n_2 and the tertiary winding n_3.

Magnetizing inductance

The magnetizing inductance L_{m}, in henries (H). The value is referred to the primary side.

Core loss resistance

An equivalent resistance R_{m} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

Initial current

A three-element vector containing the initial currents on the primary side i_1, the secondary side i_2 and the tertiary side i_3, in amperes (A). The currents are considered positive if flowing into the transformer at the marked terminals. The default is `[0 0 0]`.

Logical Operator

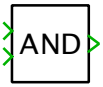
Purpose

Combine input signals logically

Library

Control / Logical

Description



The selected Logical Operator is applied to the input signals. The output of the Logical Operator is 1 if the logical operation returns true, otherwise 0. In case of a single input, the operator is applied to all elements of the input vector.

Parameters

Operator

Chooses which logical operator is applied to the input signals. Available operators are:

- `AND`: $y = u_n \& u_{n-1} \& \ldots \& u_1 \& u_0$
- `OR`: $y = u_n | u_{n-1} | \ldots | u_1 | u_0$
- `NAND`: $y = \neg(u_n \& u_{n-1} \& \ldots \& u_1 \& u_0)$
- `NOR`: $y = \neg(u_n | u_{n-1} | \ldots | u_1 | u_0)$
- `XOR`: $y = u_n \oplus u_{n-1} \oplus \ldots \oplus u_1 \oplus u_0$
- `NOT`: $y = \neg u$

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 Multiplexer

Purpose

Combine several magnetic connections into single vector

Library

Magnetic / Connectivity

Description



This multiplexer combines several magnetic connections into one vector connection. The individual connections may themselves be vectors. In the block icon, the first individual connection is marked with a dot.

Parameters

Width

This parameter allows you to specify the number and/or width of the individual connections. You can choose between the following formats for this parameter:

Scalar: A scalar specifies the number of individual connections each having a width of 1.

Vector: The length of the vector determines the number of individual connections. Each element specifies the width of the corresponding individual connections.

↪ *Added in PLECS 4.6.*

Magnetic Permeance

Purpose

Linear magnetic permeance

Library

Magnetic / Components

Description



This component provides a magnetic flux path. It establishes a linear relationship between the magnetic flux Φ and the magneto-motive force $\mathcal{F}$. Magnetic permeance $\mathcal{P}$ is the reciprocal of magnetic reluctance $\mathcal{R}$:

$$\mathcal{P} = \frac{1}{\mathcal{R}} = \frac{\Phi}{\mathcal{F}}$$

Parameters

Permeance

Magnetic permeance of the flux path, in webers per ampere-turn $(\mathrm{Wb/A})$.

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 / Connectivity

Description



Magnetic ports are used to establish magnetic connections between a schematic and the subschematic of a Subsystem (p. 977). 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.

A terminal can be moved around the edges of the subsystem by dragging it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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.

Parameters

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 / Components

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

$$P_{\mathrm{loss}} = \mathcal{F}\,\dot{\Phi} = \mathcal{F}^2/R_{\mathrm{m}}$$

converted into heat in a magnetic resistance corresponds to the power lost in the electrical circuit.

Parameters

Resistance

Effective magnetic resistance R_{m}, in $(\mathrm{A}\cdot(\mathrm{Wb/s})^{-1})$.

Probe Signals

MMF

The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).

Magnetic Selector

Purpose

Select or reorder elements from vector connection

Library

Magnetic / Connectivity

Description



The Magnetic Selector block connects the individual elements of the output connection to the specified elements of the input connection. The input connection is marked with a dot.

Parameters

Input width

The width of the input connection.

Output indices

A vector with the indices of the input elements that the output connection should contain.

↪ *Added in PLECS 4.6.*

Manual Double Switch

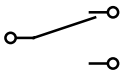
Purpose

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.

Parameters

Switch position

The current position of the switch. Possible values are 0 (upper position) and 1 (lower position).

Probe Signals

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

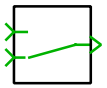
Purpose

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.

Parameters

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.

Parameters

Switch position

The current position of the switch. Possible values are `0` (upper position, off) and `1` (lower position, on).

Probe Signals

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

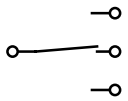
Purpose

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.

Parameters

Switch position

The current position of the switch. Possible values are `-1` (upper position), `0` (middle position) and `1` (lower position).

Probe Signals

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 $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$.

Initial position

The initial position x_0, in meters (m). 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

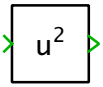
Purpose

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.

Parameters

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`: $y = \mathrm{mod}(u, v)$
- `rem`: $y = \mathrm{rem}(u, v)$

`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.

Matrix Bidirectional Phase-Shifted Full-Bridge Converter

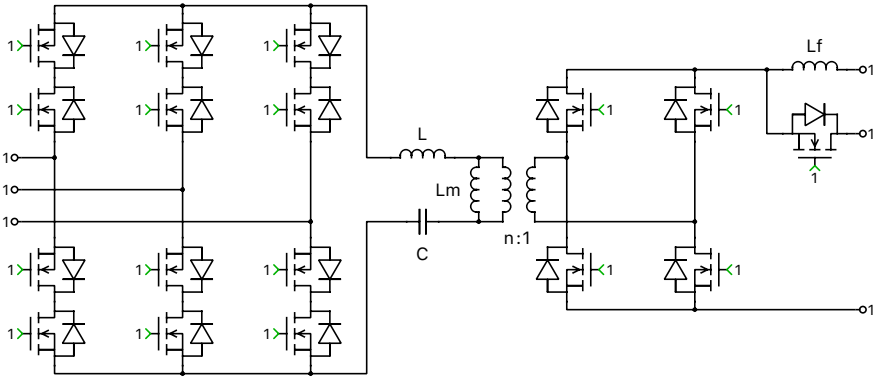
Purpose

Bidirectional phase-shifted full-bridge converter with 3-phase matrix converter frontend and optional series capacitor, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a bidirectional phase-shifted full-bridge converter with a 3-phase matrix converter frontend and free-wheeling on the low voltage side. A series DC blocking capacitor to prevent transformer saturation is optional.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> Inductive currents must not be interrupted abruptly. Therefore, at all times, the combination of switch signals on the matrix front-end must provide a path for the inductive current of the primary winding. This can be achieved by ensuring that at least one high-side switch and one low-side switch are simultaneously ON in the direction of the current flow.
> The voltage sources connected to the converter terminals must not be short-circuited. Therefore, combinations of switching signals that cause phase-to-phase short circuits or shoot-through conditions are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.64 on p. 809.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a DC blocking capacitor at the primary side, to prevent transformer saturation.

Primary capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C, in farads (F).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Filter inductance
A non-zero scalar specifying the filter inductance L_f, in henries (H).

Filter resistance
A scalar specifying the filter winding resistance R_{Lf}, in ohms (Ω).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through or interruption of inductive currents.

Probe Signals

See Fig. 16.64 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary currents
A vector containing the currents at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Filter current
The current through the inductor L_f, in amperes (A), defined positive in the direction of the arrow.

Freewheeling current
The current through the snubber switch, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Primary gate signals
A vector for each phase containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals

A vector containing the gate signals of the secondary-side semiconductor switches.

Snubber gate signal

The gate signal of the snubber semiconductor switch.

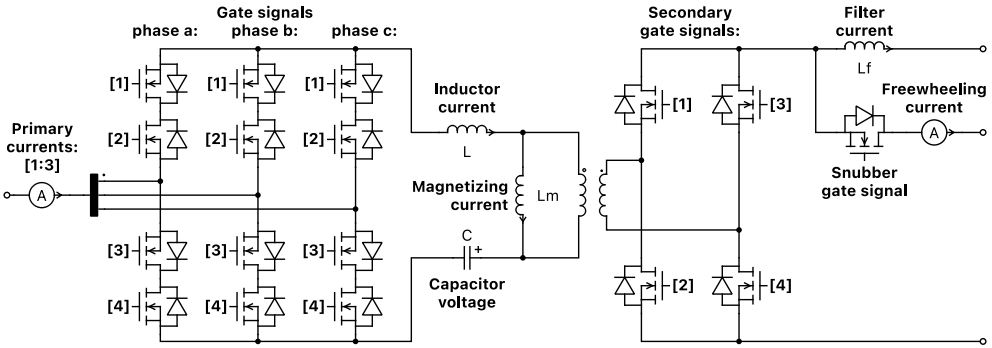


Fig. 16.64: Probe Signals for the 3x1 Matrix Bidirectional Phase-Shifted Full-Bridge Converter

---

↪ *Added in PLECS 5.0.*

Matrix CLLLC Converter

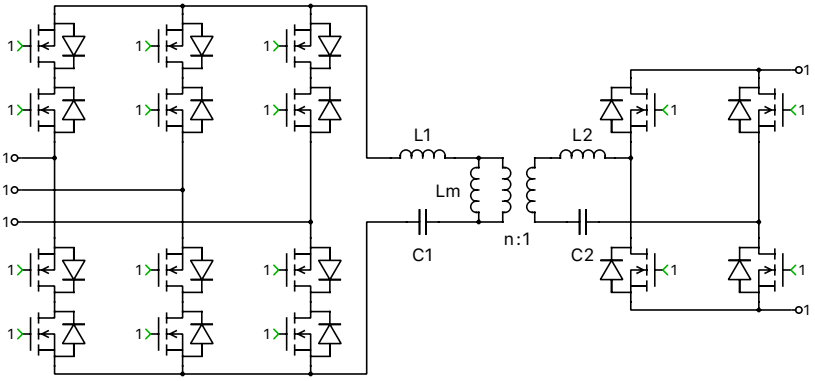
Purpose

CLLLC converter with 1/2/3/4-phase matrix converter frontend, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This help page applies to a set of components implementing isolated, resonant CLLLC converters with a 1-, 2-, 3-, or 4-phase matrix front-end. The topology can be configured as a dual active bridge (DAB) by omitting the resonant capacitors, or as an LLC converter by omitting the secondary-side inductor and capacitor.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> Inductive currents must not be interrupted abruptly. Therefore, at all times, the combination of switch signals on the matrix front-end must provide a path for the inductive current of the primary winding. This can be achieved by ensuring that at least one high-side switch and one low-side switch are simultaneously ON in the direction of the current flow.
> The voltage sources connected to the converter terminals must not be short-circuited. Therefore, combinations of switching signals that cause phase-to-phase short circuits or shoot-through conditions are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.65 on p. 813.

Primary inductance
A non-zero scalar specifying the primary-side inductance L_1, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Primary winding resistance
A scalar specifying the primary-side winding resistance R_{L1}, in ohms (Ω).

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a resonant capacitor at the primary side.

Primary resonant capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C_1, in farads (F).

Magnetizing inductance
A non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Include secondary inductance
Allows you to include (`yes`) or remove (`no`) a series leakage inductance on the secondary side.

Secondary inductance
If the **Include secondary inductance** option is set to `yes`, this parameter requires a non-zero scalar specifying the secondary-side series inductance L_2, including the transformer leakage inductance and any external inductors, in henries (H).

Secondary winding resistance
If the **Include secondary inductance** option is set to yes, this parameter requires a scalar specifying the secondary-side winding resistance R_{L2}, in ohms (Ω).

Include secondary capacitor
Allows you to include (yes) or remove (no) a resonant capacitor at the secondary side.

Secondary resonant capacitance
If the **Include secondary capacitor** option is set to yes, this parameter requires a non-zero scalar for the capacitance C_2, in farads (F).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through or interruption of inductive currents.

Probe Signals

See Fig. 16.65 for the probe signal positions.

Inductor currents
A vector containing the currents through the inductors L_1 and L_2, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltages
A vector containing the voltages across the capacitors C_1 and C_2, in volts (V), defined positive at the sides marked with (+).

Nanostep Probe Signals

Inductor currents
A vector containing the currents through the inductors L_1 and L_2, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor L_m, in amperes (A), defined positive in the direction of the arrow.

Primary currents
A vector containing the currents at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current

The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltages

A vector containing the voltages across the capacitors C_1 and C_2, in volts (V), defined positive at the sides marked with (+).

Primary gate signals

A vector for each phase containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals

A vector containing the gate signals of the secondary-side semiconductor switches.

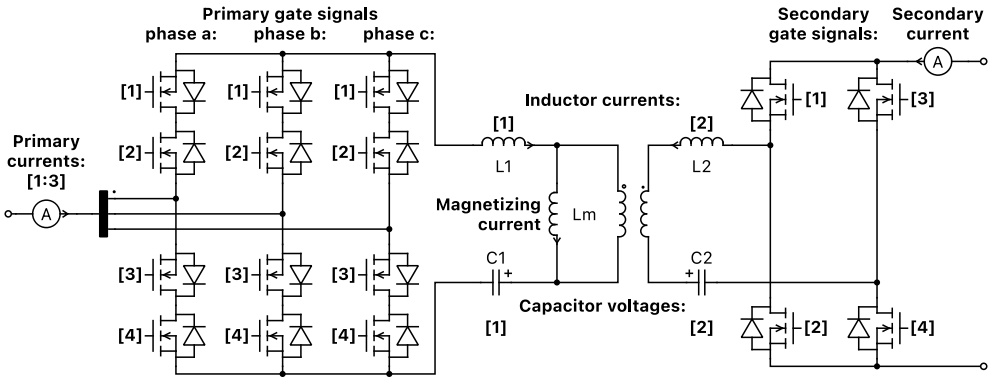


Fig. 16.65: Probe Signals for the 3x1 Matrix CLLLC Converter

↪ *Added in PLECS 5.0.*

Matrix Phase-Shifted Full-Bridge Converter

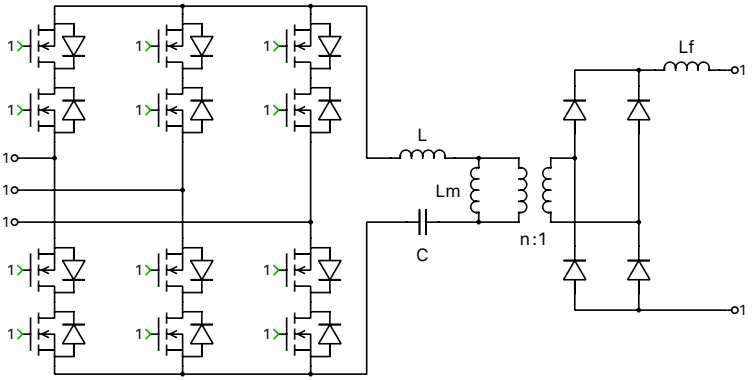
Purpose

Phase-shifted full-bridge converter with 3-phase matrix converter frontend and optional series capacitor, optimized for “Real-Time Simulations” (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated phase-shifted full-bridge converter with a 3-phase matrix converter frontend. A series DC blocking capacitor to prevent transformer saturation is optional.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> Inductive currents must not be interrupted abruptly. Therefore, at all times, the combination of switch signals on the matrix front-end must provide a path for the inductive current of the primary winding. This can be achieved by ensuring that at least one high-side switch and one low-side switch are simultaneously ON in the direction of the current flow.
> The voltage sources connected to the converter terminals must not be short-circuited. Therefore, combinations of switching signals that cause phase-to-phase short circuits or shoot-through conditions are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.66 on p. 817.

Output rectifier
Use this setting to choose between center-tapped `Half bridge` and `Full bridge` output rectifier topology.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a DC blocking capacitor at the primary side, to prevent transformer saturation.

Primary capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C, in farads (F).

Include magnetizing inductance
Allows you to include (`yes`) or remove (`no`) the magnetizing inductance L_m on the transformer primary side.

Magnetizing inductance
If the **Include magnetizing inductance** option is set to `yes`, this parameter requires a non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Filter inductance
A non-zero scalar specifying the filter inductance $\mathrm{L_f}$, in henries (H).

Filter resistance
A scalar specifying the filter winding resistance $\mathrm{R_{Lf}}$, in ohms (Ω).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through or interruption of inductive currents.

Probe Signals

See Fig. 16.66 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Primary currents
A vector containing the currents at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Filter current
The current through the inductor $\mathrm{L_f}$, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Gate signals

A vector for each phase containing the gate signals of the semiconductor switches.

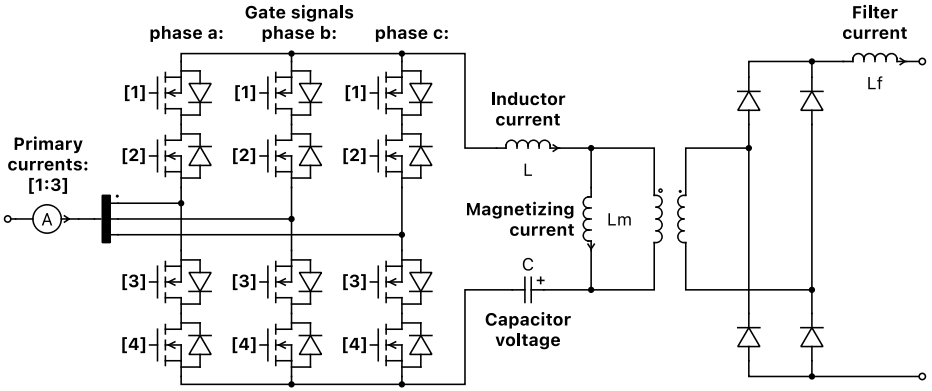


Fig. 16.66: Probes for the 3x1 Matrix Phase-Shifted Full-Bridge Converter

---

↪ *Added in PLECS 5.0.*

Memory

Purpose

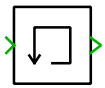
Provide input signal from previous major time step

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 sample time. If you need to specify a sample time explicitly, please use the Delay (p. 639) block instead.
- 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.

Example Model

- The example model **"Could not solve the algebraic loop - Example 2"** features the decoupling of an algebraic loop.
- Find it in PLECS under **Help > PLECS Documentation** > List of Example Models.

Parameters

Initial condition
The initial output during the first major time step.

Probe Signals

Input
The input signal.

Output
The output signal.

Meter (3-Phase)

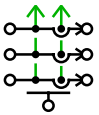
Purpose

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.

Parameters

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

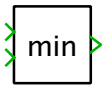
Purpose

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.

Example Model

- See the example model **"Minimum / Maximum"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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

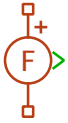
Purpose

Output the measured magneto-motive force

Library

Magnetic / Meters

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 Signal Outport block (p. 952) or by dragging the component into the dialog box of a Probe block.

> **Note**
>
> The MMF Meter is ideal, i.e. it has an zero internal permeance. Hence, if multiple meters are connected in series the MMF across an individual meter is undefined. This produces a run-time error.

Probe Signals

MMF

The measured magneto-motive force in ampere-turns (A).

MMF Source (Constant)

Purpose

Generate a constant magneto-motive force

Library

Magnetic / Sources

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.

Parameters

Voltage

The magnitude of the MMF, in ampere-turns (A). The default value is 1.

Probe Signals

MMF

The magneto-motive force of the source, in ampere-turns (A).

MMF Source (Controlled)

Purpose

Generate a variable magneto-motive force

Library

Magnetic / Sources

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.

Parameters

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 Signals

MMF

The magneto-motive force of the source, in ampere-turns (A).

Model Reference

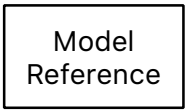
Purpose

Reference a subsystem or netlist from the same or another model

Library

System

Description



The Model Reference block allows you to reference a Subsystem (p. 977), Netlist (p. 845) or Netlist File Reference (p. 847) from the same or another model. The reference is synchronized when the original component has changed. This concept is similar to library links (p. 72) with the difference that the referenced model file path can be specified relative to the referencing model.

↪ *Changed in PLECS 4.8:* Added support for references into the same model,which allows self-referencing libraries.

The model reference is defined in the Model Reference dialog. Double-click on the Model Reference block to open the dialog, then drag the component that you want to reference into it. The labels **Model file** and **Component path** will be updated to reflect the component that is referenced. By default, the referenced model file is specified with a path relative to the referencing model. You can change this with the **Model file reference** selector.

When you click **OK** to apply the settings, the Model Reference block is replaced with a copy of the referenced component that is marked with a small solid curved arrow (↗) in the lower left corner of the component icon. A right-click on this arrow opens a context menu that allows you to break the link, to edit the model reference, or to show the original component. This context menu can also be accessed by right-clicking the component and selecting **Link**.

If the referenced component is in the same model, the **Model file** label shows the text *local reference*.

Note

- For technical reasons, you can only reference components in the same model (local references) or PLECS Standalone models. See “Opening a PLECS Standalone Model” (p. 58) to learn how to open a PLECS Standalone model in PLECS Blockset.
  ↪ *Added in PLECS 4.6.*
- Synchronization is performed when the model is loaded, when the reference is changed in the dialog, or before a simulation. To force an update, choose **Synchronize all external links...** from the **Edit** menu.
  ↪ *Added in PLECS 4.8.*
- Synchronization loads and updates all referenced models and libraries as necessary.
- The model file is marked as modified if an update was required.
- Only the reference to the model file and subsystem is stored. To simulate a model, all referenced model files are required.

↪ *Added in PLECS 4.5.*

Monoflop

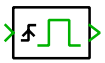
Purpose

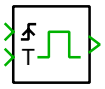
Generate pulse of specified duration when triggered

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 pulse duration can be specified statically via a parameter or dynamically during the simulation via an input signal. The pulse duration must be greater than or equal to zero.

The Monoflop can be retriggered, i.e. if the trigger condition is fulfilled again while the output is 1, the turn-off time is recalculated based on the simulation time and the current pulse duration. If the pulse duration is zero when the Monoflop is retriggered, the output is reset to 0 immediately.

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*).

  ↪ *Added in PLECS 4.3.*

Pulse duration
: The duration for which the output is set to 1, in seconds (s). 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

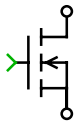
Ideal MOSFET with optional on-resistance

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 (Ω). The default is `0`.

Initial conductivity

Initial conduction state of the MOSFET. The MOSFET is initially blocking if the parameter evaluates to zero, otherwise it is conducting.

Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter "Thermal Modeling" (p. 171). If no thermal description is given, the losses are calculated based on the voltage drop $v_{\mathrm{on}} = R_{\mathrm{on}} \cdot i$.

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in $(\mathrm{K/W})$. The default is `0`.

↪ *Added in PLECS 4.6.*

Number of parallel devices

This parameter is used to simulate the effect of connecting multiple *identical* devices in parallel while adding only one single switch element to the electrical system equations. Other component parameters such as the **Thermal description**, the **Thermal interface resistance** and also the electrical **On-resistance** refer to the characteristics of an individual device, while the calculated losses and other probe signals refer to the sum over all devices.

If N_{p} is the number of parallel devices, each device will conduct $1/N_{\mathrm{p}}$ of the total current through the component, and the total component loss

will be N_p times the loss of an individual device. The effective electrical on-resistance and the effective thermal interface resistance of the component are also $1/N_p$ times the respective parameter values. The default is 1.

↪ *Added in PLECS 5.0.*

> **Note**
>
> If **Number of parallel devices** is greater than 1, the component itself cannot be vectorized, i.e. all other component parameters must have scalar values, and the component terminals must be connected to a scalar connection (see also "Vectorizing Physical Components" (p. 78)).

> **Example Model**
>
> - See the example model **"Thermal Modeling with Parallel Devices"**.
> - Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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" (p. 177).

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)

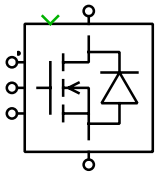
Purpose

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. 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 (p. 828) and Diode (p. 641) 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, 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, the MOSFET (p. 828) and the Diode (p. 641).

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

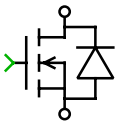
Purpose

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 (p. 828) with an individual anti-parallel Diode (p. 641), or allow a small non-zero load current to flow by connecting a large load resistance to the converter.

Parameters

On-resistance
The resistance R_{on} of the conducting device, in ohms (Ω). The default is `0`.

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” (p. 171) and “Losses of Semiconductor Switch with Diode” (p. 206).

Thermal interface resistance
The thermal resistance of the interface material between case and heat sink, in $(\mathrm{K/W})$. The default is `0`.

↪ *Added in PLECS 4.6.*

Number of parallel devices

This parameter is used to simulate the effect of connecting multiple *identical* devices in parallel while adding only one single switch element to the electrical system equations. Other component parameters such as the **Thermal description**, the **Thermal interface resistance** and also the electrical **On-resistance** refer to the characteristics of an individual device, while the calculated losses and other probe signals refer to the sum over all devices.

If N_p is the number of parallel devices, each device will conduct $1/N_\mathrm{p}$ of the total current through the component, and the total component loss will be N_p times the loss of an individual device. The effective electrical on-resistance and the effective thermal interface resistance of the component are also $1/N_\mathrm{p}$ times the respective parameter values. The default is 1.

↪ *Added in PLECS 5.0.*

Note

If **Number of parallel devices** is greater than 1, the component itself cannot be vectorized, i.e. all other component parameters must have scalar values, and the component terminals must be connected to a scalar connection (see also “Vectorizing Physical Components” (p. 78)).

Example Model

- See the example model **“Thermal Modeling with Parallel Devices”**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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” (p. 177).

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

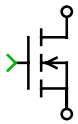
Purpose

Dynamic MOSFET model with finite current slopes during turn-on and turn-off

Library

Electrical / Power Semiconductors

Description



In contrast to the ideal MOSFET (p. 828) model 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 (p. 754).

> **Note**
>
> - Due to the small time-constants introduced by the turn-on and turn-off transients a stiff solver is recommended for this device model.
> - If multiple MOSFETs are connected in series, the off-resistance may not be infinite.

Parameters

Blocking voltage
Maximum voltage V_{DSS} in volts (V) that under any conditions should be applied between drain and source.

Continuous drain current
Maximum dc current I_{D} in amperes (A) that the MOSFET can conduct.

On-resistance
The resistance R_{on} of the conducting device, in ohms (Ω). The default is `0`.

Off-resistance
The resistance R_{off} of the blocking device, in ohms (Ω). The default is `1e6`. This parameter may be set to `inf` unless multiple MOSFETs are connected in series.

Rise time
Time t_{r} in seconds (s) between instants when the drain current has risen from $10\,\%$ to $90\,\%$ of the continuous drain current I_{DS}.

Fall time
Time t_f in seconds (s) 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.

Stray inductance
Internal inductance L_s 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

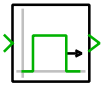
Purpose

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 (p. 860).

Parameters

Initial Condition

The initial condition describes the input signal before simulation start. If the input is a scalar signal, the parameter must be a scalar too. If input and output are vectorized signals, a vector can be used. The number of elements in the vector must match the number of input signals. The default value of this parameter is `0`.

Averaging time

A scalar specifying the period or a two-element vector specifying the period and offset of the averaging interval, in seconds (s). See also the **Discrete-Periodic** sample time type in "Sample Times" (p. 43).

Probe Signals

Input

The input signal.

Output

The filtered output signal.

Multiport Signal Switch

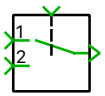
Purpose

Select one of multiple input signals depending on control signal

Library

Control / Discontinuous

Description



The block output is connected to the block input specified by the scalar value at the control terminal. If the control terminal signal is not in the range of the available input terminals, the default input terminal is used.

Example Model

- See the example model **"Multiport Signal Switch"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Parameters

Number of inputs
The number of input terminals. It has to be a literal integer value bigger than 2.

Default input
The default input terminal if the control signal is out of range.

Probe Signals

Inputs
The block input signals.

Output
The block output signal.

Switch position
The state of the switch position. This is the control signal or, in case the control signal is out of range, the default input parameter.

Out of range
A boolean value that is true if the control input is out of range.

↪ *Added in PLECS 4.5.*

Mutual Inductance (2 Windings)

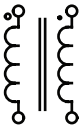
Purpose

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 in Fig. 16.67.

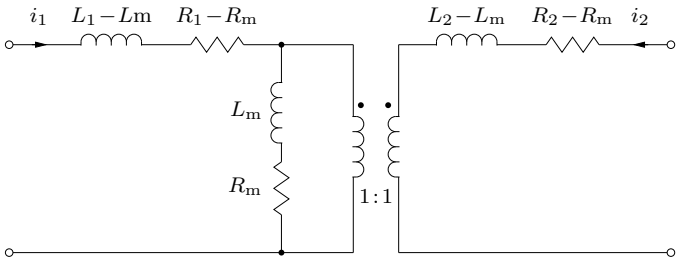


Fig. 16.67: Electrical circuit for Mutual Inductance (2 Windings).

In the symbol of the mutual inductance, the positive terminal of the primary winding is marked with a little circle. The positive terminal of the secondary winding is marked with a dot.

Parameters

Self inductance

A two-element vector containing the self inductance for the primary winding L_1 and the secondary winding L_2. The inductivity is given in henries (H).

Winding resistance

A two-element vector containing the self resistance of the primary winding R_1 and the secondary winding R_2, in ohms (Ω).

Mutual inductance

The mutual inductance L_m, in henries (H).

Mutual resistance

The mutual resistance R_m, in ohms (Ω).

Initial current

A two-element vector containing the initial currents on the primary side i_1 and the secondary side i_2, in amperes (A). The direction of the initial current inside the component is from the positive to the negative terminal. The default value is `[0 0]`.

Probe Signals

Winding *i* current

The current flowing through winding *i*, in amperes (A).

Winding *i* voltage

The voltage measured across winding *i*, in volts (V).

Mutual Inductance (3 Windings)

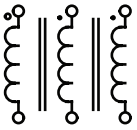
Purpose

Magnetic coupling between three lossy windings

Library

Electrical / Transformers

Description



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 in Fig. 16.68.

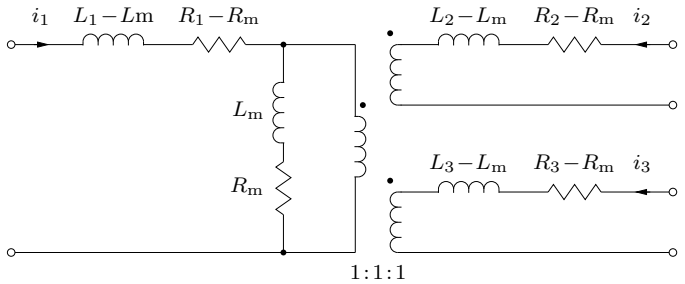


Fig. 16.68: Electrical circuit for Mutual Inductance (3 Windings).

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 (Ω).

Mutual inductance

The mutual inductance L_m, in henries (H).

Mutual resistance

The mutual resistance R_m, in ohms (Ω).

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).

Mutual Inductor

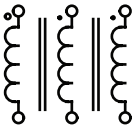
Purpose

Ideal mutual inductor

Library

Electrical / Passive Components

Description



This component provides two or more coupled inductors. Electrically, it is equivalent with a vectorized Inductor (p. 779). 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.

> **Example Model**
>
> - See the example model **"Mutual Inductor"**.
> - Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} L_1 & M_{1,2} & \cdots & M_{1,n} \\ M_{2,1} & L_2 & \cdots & M_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & L_n \end{bmatrix} \begin{bmatrix} \frac{di_1}{dt} \\ \frac{di_2}{dt} \\ \vdots \\ \frac{di_n}{dt} \end{bmatrix}$$

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 (p. 744).

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).

Netlist

Purpose

Import or edit a SPICE netlist

Library

SPICE Components

Description



A SPICE netlist can be included and connected to other components in a schematic using the netlist component. See section “Netlist Component” (p. 397) for a detailed overview.

The Netlist dialog consists of five tabbed panes that are described below.

Netlist

The first tab is equipped with a text editor in which the netlist can be either directly written/edited or imported from file using the **Import Netlist** button.

When importing, the file can contain multiple circuits (netlist descriptions). You can choose which circuit to import using the selection wizard. If an imported circuit has dependencies (its netlist description refers to subcircuits) within the same file, these are also imported automatically. The netlist description that results from an import does not reference the original imported file (the netlist description is hard-copied).

The first subcircuit in the text editor will be treated as main circuit. Nodes from the main subcircuit are automatically exposed as terminals. See “Netlist Cheat Sheets” (p. 411) for basic usage and “Netlist Syntax Reference” (p. 429) for detailed syntax rules.

Terminals

In the Terminals tab you can override the name of the Netlist terminals. There is a table entry for each exposed node of the netlist. Terminal names must be unique.

To move a terminal on the schematic, hold **Shift** and drag the terminal along the edges of the component.

Probes

The Probes tab allows you to define probe signals for the Netlist component. These signals will be accessible from the PLECS Probe (p. 116)

To define a probe signal, use the plus button next to the table. A dialog will guide you through the creation of either a terminal-based or an internal probe.

- Terminal-based probes measure either the current flowing into a terminal or the voltage difference between two terminals. Terminals are identified using the labels defined in the **Terminals** pane.
- Internal probes can be used to monitor internal nodal voltages and two-terminal internal components. Probable elements are automatically identified using the call hierarchy of the parsed netlist.

Parameters

Parameters of the main subcircuit can be overwritten in this tab. You can specify specify any value here, including variables from your schematic's **Initialization** tab.

Icon

You can define a custom icon for your Netlist component via this tab. Details regarding the supported Lua syntax (p. 107) can found in the description of the Mask Icon (p. 84).

See also

"Using PLECS Spice" (p. 397)
How to use PLECS Spice and the Netlist component in a circuit.

"Netlist Cheat Sheets" (p. 411)
Example netlists introducing the basic syntax.

"Netlist Syntax Reference" (p. 429)
Documentation of supported netlist syntax.

Netlist File Reference
Alternative component that imports a netlist from a referenced file at runtime.

↪ *Added in PLECS 5.0.*

Netlist File Reference

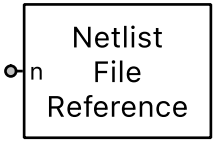
Purpose

Import a SPICE netlist from a file at runtime

Library

SPICE Components

Description



This component is an alternative to the Netlist (p. 845) component. Instead of offering a netlist editor holding a hard copy of a previously imported netlist, this component imports netlists at runtime from a referenced file. See "Netlist File Reference Component" (p. 403) for a comparison against the Netlist (p. 845) component.

The Netlist File Reference dialog consists of five tabbed panes that are described below.

Reference

The Reference tab lets you specify a file, a type and a name for the netlist to be imported. During the initialization of a new simulation, the `.subckt` or `.model` (depending on the selected type) with the given name is automatically imported from the selected file. The import process works identical to the import functionality of the Netlist (p. 845) component (see "Importing Netlists" (p. 400)). File, type and name can be set via variables.

The file path can be absolute, relative to the current .plecs file or relative to any path specified as library search path in the preferences (see "Library" (p. 63)).

In contrast to the Netlist (p. 845), detailed diagnostics are not shown. Should there be parsing errors while importing, the simulation will be aborted.

Terminals

In the Terminals tab, you need to define the nodes of the subcircuit that you wish to import. The node names you enter must exactly match those in the imported netlist - order does not matter, but the sets must be identical. An error is thrown if they do not match. For each node, you must specify the name of the corresponding terminal in the schematic.

To move a terminal on the schematic, hold **Shift** and drag the terminal along the edges of the component.

Note

When importing a `.model` from a file, the import automatically generates a default `.subckt` containing a single element using the imported compact model description.
The node names are summarized in the following table.

Model type	Node names	Generated statement
D	+, -	`D + - modelName`
R	+, -	`R + - modelName`
C	+, -	`C + - modelName`
NMOS	d, g, s	`M d g s s modelName`
PMOS	d, g, s	`M d g s s modelName`
VDMOS	d, g, s	`M d g s modelName`
NPN	c, b, e	`Q c b e modelName`
PNP	c, b, e	`Q c b e modelName`
PJF	d, g, s	`J d g s modelName`
NJF	d, g, s	`J d g s modelName`
SW	+, -, c+, c-	`S + - c+ c- modelName`
VSWITCH	+, -, c+, c-	`S + - c+ c- modelName`

Probes

The Probes tab allows you to define probe signals based on the Netlist component's terminals. In contrast to the Netlist (p. 845) component, only terminal-based probes are available.

Parameters

You can add parameters here that will be passed on to the imported subcircuit (see "(X) Subcircuit Instance Statement" (p. 469)). The values of these parameters can be set via variables. The order of these parameters does not matter.

The parameters overwrite parameters defined after `params:` analog to the Parameters tab of the Netlist component (p. 402). It is not necessary to add all parameters defined after `params:` in the netlist to the table. Any parameter not specified in the table will use its default value as specified in the netlist.

In contrast to the Netlist component (p. 845), you can also specify parameters that are not defined after `params:` in the netlist. Through this netlists that expect to access parameters from an outer scope can be used (see "Shadowing" (p. 434)). In this case, the parameter will be passed to the netlist as if it would be listed after `params:` and a warning is shown. This must not conflict with a `.param` definition in the netlist. A redefinition within the netlist always throws an error.

Icon

You can define a custom icon for your Netlist File Reference component via this tab. Details regarding the supported Lua syntax (p. 107) can found in the description of the Mask Icon (p. 84).

See also

"Using PLECS Spice" (p. 397)
How to use PLECS Spice.

Netlist (p. 845)
Alternative component that allows local in-place editing of a netlist.

"Netlist File Reference Component" (p. 403)
Comparison between the Netlist File Reference and the Netlist component.

↪ *Added in PLECS 5.0.*

Non-Excited Synchronous Machine

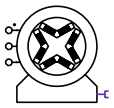
Purpose

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 **Edit > Subsystem** menu or the block's context 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 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:

$$\vec{v}_\mathrm{s} = R_\mathrm{s} \cdot \vec{i}_\mathrm{s} + (\mathbf{L}_{\sigma\mathrm{s}} + \mathbf{L}_\mathrm{mi}) \cdot (\frac{d\vec{i}_\mathrm{s}}{dt} + j \cdot \omega \cdot \vec{i}_\mathrm{s}) + \vec{e}_\mathrm{s}$$

$$\vec{e}_\mathrm{s} = j \cdot \omega \cdot \vec{\varphi}_\mathrm{m} - \mathbf{L}_\mathrm{mi} \cdot j \cdot \omega \cdot \vec{i}_\mathrm{s}$$

with

$$\vec{x} = \begin{bmatrix} x_\mathrm{d} \\ x_\mathrm{q} \end{bmatrix}$$

where $j \cdot \vec{x}$ rotates the synchronous frame vector, $\vec{x}$, clockwise by $90°$.

These equations are transformed back into the stationary frame to control the VBR network. The zero-sequence impedance of the machine is set to Lss.

The VBR network implementation uses the Variable Inductor (p. 1099) to model the time-varying characteristic of the machine inductance. PLECS does not support code generation for the Variable Inductor component. When generating code for the VBR model the machine equations are reformulated to interface electrically with a constant inductance and emulate the variable portion of the inductance with a voltage source.

Electro-Mechanical System

Electromagnetic torque:

$$T_\mathrm{e} = \frac{3}{2}\, p\, (\varphi_\mathrm{d}\, i_\mathrm{q} - \varphi_\mathrm{q}\, i_\mathrm{d})$$

Mechanical System

Mechanical rotor speed ω_m:

$$\dot{\omega}_\mathrm{m} = \frac{1}{J}(T_\mathrm{e} - F\omega_\mathrm{m} - T_\mathrm{m})$$

$$\dot{\theta}_\mathrm{m} = \omega_\mathrm{m}$$

Parameters

General

Stator resistance
Armature or stator resistance R_s in Ω.

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 $i_{\mathrm{a},0}$ and $i_{\mathrm{b},0}$ of phase a and b in amperes (A). $i_{\mathrm{c},0}$ 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. $[0 \dots i_{\mathrm{d,max}}]$. 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. $[0 \dots i_{\mathrm{q,max}}]$. 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 (Nms^2).

Friction coefficient

Viscous friction F in (Nms).

Initial rotor speed

Initial mechanical rotor speed $\omega_{\mathrm{m},0}$ in radians per second $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Initial rotor position

Initial mechanical rotor angle $\theta_{\mathrm{m},0}$ in radians.

Probe Signals

Stator phase currents

The three-phase stator winding currents i_{a}, i_{b} and i_{c}, in amperes (A). Currents flowing into the machine are considered positive.

Stator flux (dq)

The stator flux linkages φ_{d} and φ_{q} in the rotating reference frame.

Rotational speed

The rotational speed ω_{m} of the rotor in radians per second $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Rotor position
The mechanical rotor angle θ_{m} in radians.

Electrical torque
The electrical torque T_{e} of the machine in newton-meters (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.

↪ *Added in PLECS 4.0.*

Offset

Purpose

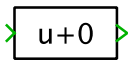
Add constant offset to input signal

Library

Control / Math

Description

The Offset block adds a constant to the input signal.



Parameters

Offset
The offset to add to the input signal. This value may be negative to subtract an offset from the signal.

Output data type
The data type of the output signal. See “Data Types” (p. 48). If you choose `inherited`, the minimum data type is `int8_t`.

Data type overflow handling
Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if **Output data type** is not set to a floating-point data type.

Probe Signals

Input
The input signal.

Output
The output signal.

Op-Amp

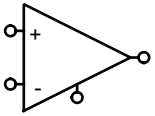
Purpose

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.

There are two demo models specifically about Op-Amp circuits:

> **Demo Model**
>
> - The demo model **"Operational Amplifier Circuits"** features Op-Amp circuits.
> - Find it in PLECS under **Window** > **Demo Models** > Electronics.

> **Demo Model**
>
> - The demo model **"Low Pass Filter Circuits"** features Op-Amp circuits for low-pass filtering.
> - Find it in PLECS under **Window** > **Demo Models** > Electronics.

Parameters

Open-loop gain

The voltage gain of the Op-Amp. The default is `1e6`.

Op-Amp with Limited Output

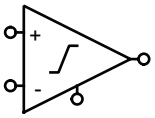
Purpose

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 following demo model shows a possible application of the Limited Op-Amp.

> **Demo Model**
>
> - See the demo model **"Operational Amplifier Circuits"**.
> - Find it in PLECS under **Window** > **Demo Models** > Electronics.

Parameters

Open-loop gain

The voltage gain of the amplifier if operating in linear mode, in volts (V). 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

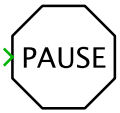
Pauses or stops the simulation when the input becomes non-zero

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.

Parameters

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

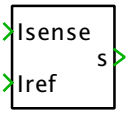
Purpose

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_{L} input exceeds the I_{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 $I_{\mathrm{L}} < I_{\mathrm{ref}}$, 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_{slope}, reduces I_{ref} during each switching cycle as follows: $I'_{\mathrm{ref}} = I_{\mathrm{ref}} - I_{\mathrm{slope}} t/T_{\mathrm{s}}$, where t is the time elapsed from the start of the switching cycle and T_{s} is the switching period. Slope compensation can be omitted by setting I_{slope} to `0`.

Periodic Average

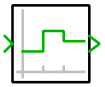
Purpose

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:

$$y(t) = \frac{1}{T}\int_{(n-1)T}^{nT} u(\tau)d\tau, \quad t \in [nT, (n+1)T) \quad n \geq 0$$

However, the implementation of Periodic Average filter is computationally less expensive and more accurate than the continuous Moving Average (p. 837) filter. The block is suited to determine average conduction losses of power semiconductors. To determine average switching losses, use the Periodic Impulse Average.

Parameters

Averaging time

A scalar specifying the period or a two-element vector specifying the period and offset of the averaging interval, in seconds (s). See also the **Discrete-Periodic** sample time type in section "Sample Times" (p. 43).

Probe Signals

Input

The input signal.

Output

The filtered output signal.

Periodic Impulse Average

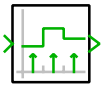
Purpose

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. 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:

$$y(t) = \frac{1}{T}\int_{(n-1)T}^{nT} u(\tau)d\tau, \quad t \in [nT, (n+1)T), \quad n \geq 0$$

If the input signal u consists of switching loss pulses and the averaging period T equals to the switching period, the formula simplifies to:

$$y(t) = \frac{1}{T}(E_{\text{on}}(n-1) + E_{\text{off}}(n-1)), \quad t \in [nT, (n+1)T), \quad n \geq 0$$

The block is suited to determine average switching losses of power semiconductors. To determine average conduction losses, use the Periodic Average.

Parameters

Averaging time

A scalar specifying the period or a two-element vector specifying the period and offset of the averaging interval, in seconds (s). See also the **Discrete-Periodic** sample time type in “Sample Times” (p. 43).

Probe Signals

Input

The input signal.

Output

The filtered output signal.

Permanent Magnet Synchronous Machine

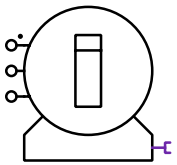
Purpose

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

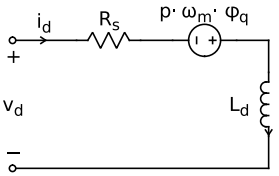


Fig. 16.69: d-axis

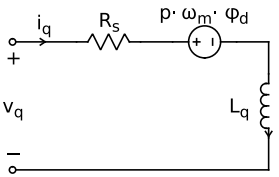


Fig. 16.70: q-axis

Stator flux linkages:

$$\varphi_\mathrm{q} = L_\mathrm{q}\, i_\mathrm{q}$$

$$\varphi_\mathrm{d} = L_\mathrm{d}\, i_\mathrm{d} + \varphi'_\mathrm{m}$$

The machine model offers two different implementations of the electrical system: a traditional rotor reference frame and a voltage behind reactance (VBR) 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.

PLECS does not support code generation for models with time-varying inductance matrices. When generating code for the VBR model the machine equations are reformulated to interface electrically with a constant inductance and emulate the variable portion of the inductance with a voltage source.

Electro-Mechanical System

Electromagnetic torque:

$$T_\mathrm{e} = \frac{3}{2}\, p\, (\varphi_\mathrm{d}\, i_\mathrm{q} - \varphi_\mathrm{q}\, i_\mathrm{d})$$

Mechanical System

Mechanical rotor speed ω_m:

$$\dot{\omega}_\mathrm{m} = \frac{1}{J}(T_\mathrm{e} - F\omega_\mathrm{m} - T_\mathrm{m})$$

$$\dot{\theta}_\mathrm{m} = \omega_\mathrm{m}$$

Parameters

Model
Implementation in the rotor reference frame or as a voltage behind reactance.

Stator resistance
Armature or stator resistance R_s in ohms (Ω).

Stator inductance
A two-element vector containing the combined stator leakage and magnetizing inductance. L_d is referred to the d-axis and L_q to the q-axis of the rotor. The values are in henries (H).

Flux induced by magnets
Constant flux linkage φ'_m in (Vs) induced by the magnets in the stator windings.

Inertia
Combined rotor and load inertia J in (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 $\omega_{\mathrm{m},0}$ in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Initial rotor position
Initial mechanical rotor angle $\theta_{\mathrm{m},0}$ in radians.

Initial stator currents
A two-element vector containing the initial stator currents $i_{\mathrm{a},0}$ and $i_{\mathrm{b},0}$ 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 amperes (A). Currents flowing into the machine are considered positive.

Stator flux (dq)
The stator flux linkages φ_d and φ_q in the rotating reference frame in (Vs):

$$\varphi_\mathrm{d} = L_\mathrm{q}\, i_\mathrm{q}$$

$$\varphi_\mathrm{d} = L_\mathrm{d}\, i_\mathrm{d} + \varphi'_\mathrm{m}$$

Rotational speed
The rotational speed ω_m of the rotor in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Rotor position
The mechanical rotor angle θ_{m} in radians.

Electrical torque
The electrical torque T_{em} of the machine in (Nm).

See also

If the stator inductance is independent of the rotor angle, i.e. $L_{\mathrm{d}} = L_{\mathrm{q}}$, it is computationally more efficient to use the simplified Brushless DC Machine (p. 587) with a sinusoidal back EMF. The parameters need to be converted as follows:

$$L - M = L_{\mathrm{d}} = L_{\mathrm{q}}$$

$$K_{\mathrm{E}} = \varphi'_{\mathrm{m}} \cdot p$$

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 (p. 584). It can be configured as a PMSM with sinusoidal back EMF if the parameters are converted as follows:

$$K_{\mathrm{c},n} = [0]$$

$$K_{\mathrm{s},n} = [-\varphi'_{\mathrm{m}} \cdot p]$$

$$L_0 - M = \frac{L_{\mathrm{d}} + L_{\mathrm{q}}}{2}$$

$$L_{\mathrm{c},n} = [0 \;\; L_{\mathrm{d}} - L_{\mathrm{q}}]$$

$$L_{\mathrm{s},n} = [0 \;\; 0]$$

Permanent Magnet Synchronous Machine (Open Winding)

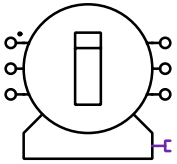
Purpose

Synchronous machine excited by permanent magnets with open stator windings.

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, positive terminal of phase a is marked with a dot.

Electrical System

The equivalent electrical circuits for the d, q and 0 axes are shown in Fig. 16.71 on p. 869.

Stator flux linkages:

$$\varphi_{\mathrm{q}} = L_q \, i_{\mathrm{q}}$$

$$\varphi_{\mathrm{d}} = L_d \, i_{\mathrm{d}} + \varphi'_{\mathrm{m}}$$

$$\varphi_0 = L_{\mathrm{ls}} \, i_0$$

The stator inductance parameters L_{d}, L_{q}, and L_{ls} are related as follows:

$$L_{\mathrm{d}} = L_{\mathrm{md}} + L_{\mathrm{ls}}$$

$$L_{\mathrm{q}} = L_{\mathrm{mq}} + L_{\mathrm{ls}}$$

With L_{md} and L_{mq} representing the magnetizing inductances and L_{ls} the stator leakage inductance.

The machine model offers two different implementations of the electrical system: a traditional rotor reference frame and a voltage behind reactance (VBR) formulation.

Rotor Reference Frame

Using Park's transformation, the 3-phase circuit equations in physical variables are transformed to the dq0 rotor reference frame. This results in constant coefficients in the differential equations making the model numerically efficient. However, interfacing the dq0 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.

> **Note**
>
> In the rotor reference frame implementation the voltage across each open ended stator winding is an input to the machine equations. The two electrical connections for each phase cannot be galvanically isolated, otherwise the phase voltage measurement is undefined. Therefore the rotor reference frame model does not support a floating star (y) connection.

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.

PLECS does not support code generation for models with time-varying inductance matrices. When generating code for the VBR model the machine equations are reformulated to interface electrically with a constant inductance and emulate the variable portion of the inductance with a voltage source.

Electro-Mechanical System

Electromagnetic torque:

$$T_\mathrm{e} = \frac{3}{2}\, p\, (\varphi_\mathrm{d}\, i_\mathrm{q} - \varphi_\mathrm{q}\, i_\mathrm{d})$$

Mechanical System

Mechanical rotor speed ω_m:

$$\dot{\omega}_\mathrm{m} = \frac{1}{J}(T_\mathrm{e} - F\omega_\mathrm{m} - T_\mathrm{m})$$

$$\dot{\theta}_{\mathrm{m}} = \omega_{\mathrm{m}}$$

Parameters

Most parameters for the Permanent Magnet Synchronous Machine (p. 862) are also applicable for this machine. Only the following parameters differ:

Stator leakage inductance

Leakage inductance of stator windings in henries (H). Stator leakage must be set to a non-zero value.

Initial stator currents

A three-element vector containing the initial stator currents $i_{\mathrm{a},0}$, $i_{\mathrm{b},0}$ and $i_{\mathrm{c},0}$ of phase a, b and c in amperes (A).

Probe Signals

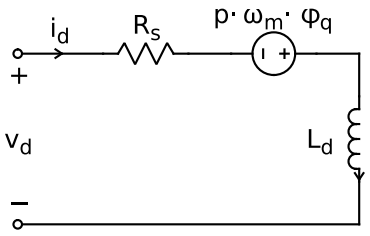
Most probe signals for the Permanent Magnet Synchronous Machine (p. 862) are also available with this machine. Only the following probe signal is different:

Stator flux (dq0)

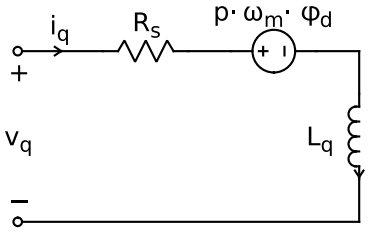
The stator flux linkages φ_{d}, φ_{q} and φ_0 in the rotating reference frame in (Vs).

↪ *Added in PLECS 4.7.*

(a) d-axis



(b) q-axis



(c) 0-axis

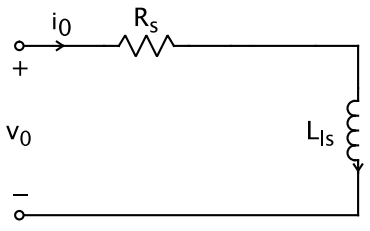


Fig. 16.71: Equivalent electrical circuits of the d,q and 0 axes of an open winding PMSM

Phase-Shifted Full-Bridge Converter

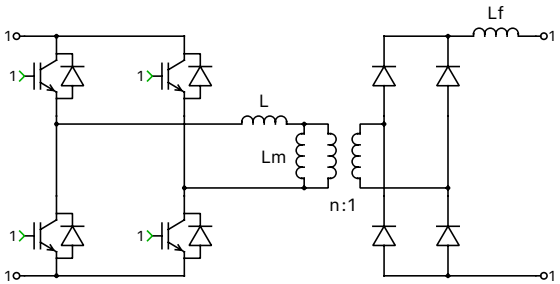
Purpose

Phase-shifted full-bridge converter with optional series capacitor, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated phase-shifted full-bridge converter with an output filter inductor. A series DC blocking capacitor to prevent transformer saturation is optional.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.72 on p. 873.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Output rectifier
Use this setting to choose between center-tapped `Half bridge` and `Full bridge` output rectifier topology.

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a DC blocking capacitor at the primary side, to prevent transformer saturation.

Primary capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C, in farads (F).

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Include magnetizing inductance
Allows you to include (`yes`) or remove (`no`) the magnetizing inductance L_m on the transformer primary side.

Magnetizing inductance
If the **Include magnetizing inductance** option is set to `yes`, this parameter requires a non-zero scalar specifying the magnetizing inductance of the transformer L_m, referred to the primary side, in henries (H).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Filter inductance
A non-zero scalar specifying the filter inductance $\mathrm{L_f}$, in henries (H).

Filter resistance
A scalar specifying the filter winding resistance $\mathrm{R_{Lf}}$, in ohms (Ω).

Assertions
When set to on, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.72 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Magnetizing current
The current through the inductor $\mathrm{L_m}$, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Filter current
The current through the inductor $\mathrm{L_f}$, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Gate signals
A vector containing the gate signals of the active semiconductor switches.

↪ *Added in PLECS 4.9.*

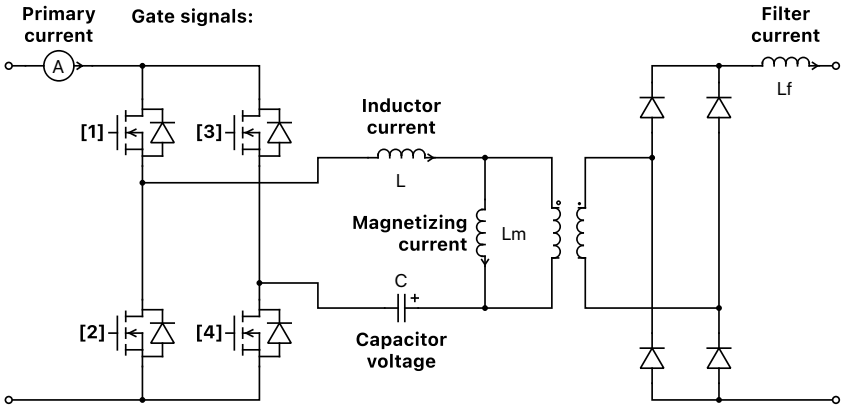


Fig. 16.72: Probes for the Phase-Shifted Full-Bridge Converter

Pi-Section Line

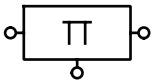
Purpose

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. Fig. 16.73 illustrates the electrical circuit used for the line model.

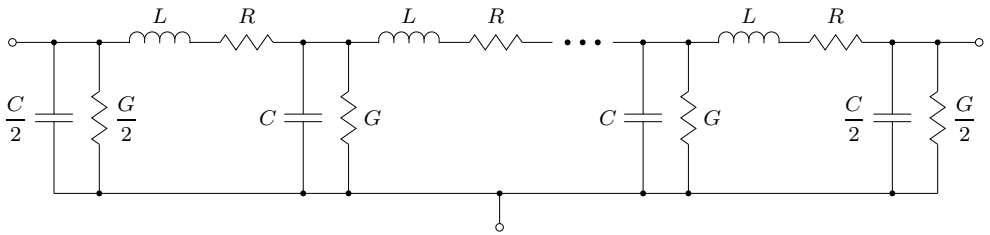


Fig. 16.73: Pi-section transmission line circuit diagram

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:

$$L = \frac{l}{n}L', \qquad R = \frac{l}{n}R', \qquad C = \frac{l}{n}C', \qquad G = \frac{l}{n}G'$$

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 $(\mathrm{H/m})$.

Resistance per unit length

The series line resistance R' per unit length. If the length l is specified in meters (m), the unit of R' is ohms per meter (Ω/m).

Capacitance per unit length

The capacitance C' between the line conductors per unit length. If the length l is specified in meters (m), the unit of C' is farads per meter $(\mathrm{F/m})$.

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 $(\mathrm{S/m})$.

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

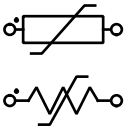
A scalar value specifying the initial voltage of all capacitors at simulation start, in volts (V).

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 and is shown in Fig. 16.74.



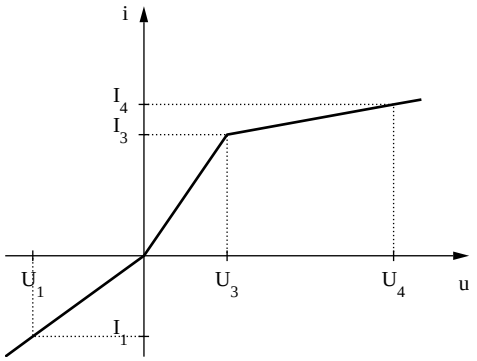


Fig. 16.74: Piece-wise linear resistor characteristic curve

The operating mode of the piece-wise linear resistor is illustrated in Fig. 16.75.

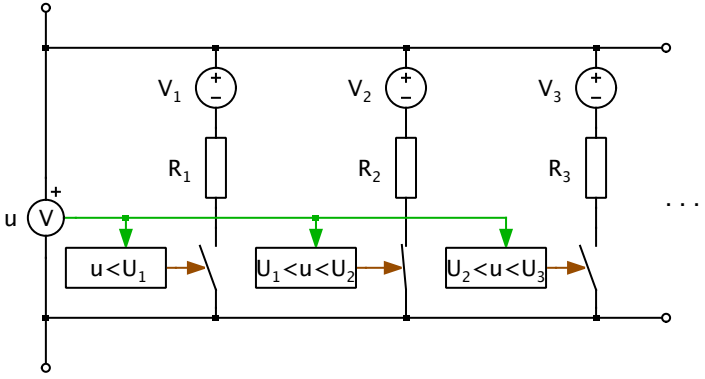


Fig. 16.75: Piece-wise linear resistor internal schematic

The voltage across the device dictates which internal switch is closed. The values $0\,\mathrm{V}$ / $0\,\mathrm{A}$ 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

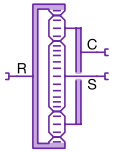
The power consumed by the resistor, in watts (W).

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 subcomponents: a ring-planet gear subsystem and a sun-planet gear subsystem, as shown in Fig. 16.76.

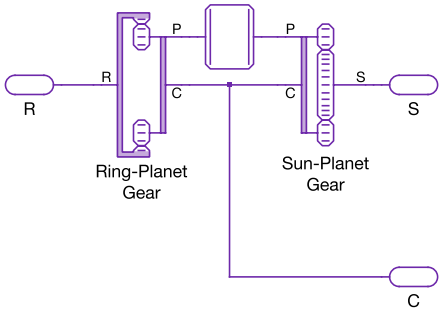


Fig. 16.76: Planetary gear set system diagram

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 ω_r, ω_s, ω_p, and ω_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 (p. 781) 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 (Nms^2).

Initial speed of panet gear
Initial angular speed of each planet gear ω_p in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Probe Signals

Sun gear speed
Angular speed of sun gear ω_s in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Planet gear speed
Angular speed of each planet gear ω_p in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Carrier
Angular speed of carrier ω_c in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Ring gear speed
Angular speed of ring gear ω_r in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

$\hookrightarrow$ *Added in PLECS 4.0.*

PLL (Single-Phase)

Purpose Implementation of a single phase PLL

Library Control / Continuous

Description This block implements two different types of a single-phase Phase Locked Loop (PLL). The single-phase PLL component provides at its output terminals an estimation of the input signal frequency, signal amplitude and the phase-angle.

PLL Types

Three fundamental blocks can be identified in a basic single-phase PLL structure.

- Phase detector: Generates an output signal that is proportional to the phase difference between the input signal and the signal generated by the PLL itself.
- Controller: Usually a PI controller to attenuate high-frequency components and to eliminate the steady-state phase-error.
- Frequency/phase-angle generator: Generates a phase-angle signal based on the estimated angular frequency. Typically, this block is constituted by a wrapping integrator.

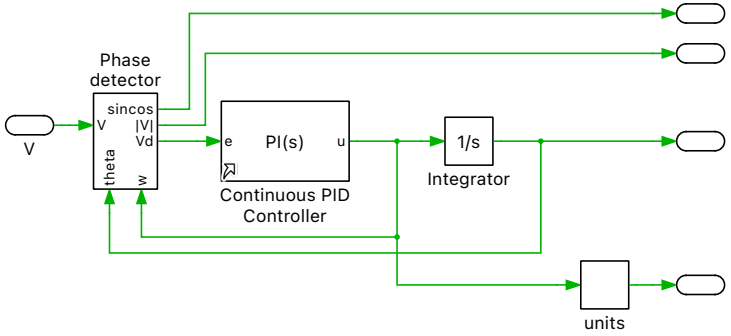


Fig. 16.77: Basic structure of a single-phase PLL

Two different types of a single-phase PLL are implemented. The main difference between the two methods is how the phase detector works. Please note that both PLL implementations use an amplitude normalization scheme. Therefore, information about the nominal input signal amplitude is not needed.

Enhanced PLL

The performance of the basic PLL implementation is enhanced by filtering the input signal with an adaptive notch filter (ANF). When the input signal has the same phase-angle and frequency as the estimation of the PLL itself, the output of the ANF becomes zero. Due to this, oscillations at the output of the phase detector are canceled out and the signal phase-angle is accurately estimated by the basic PLL structure.

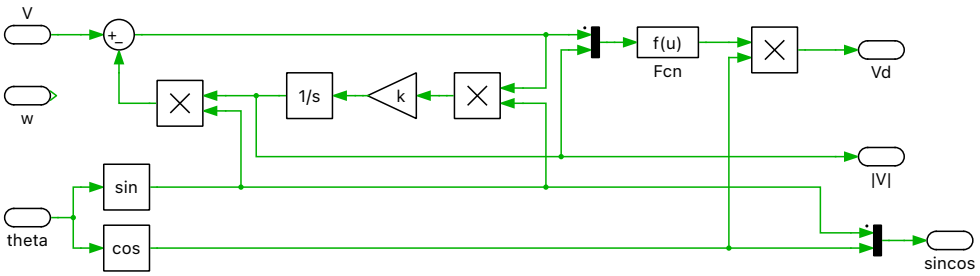


Fig. 16.78: Enhanced structure with adaptive notch filter

SOGI

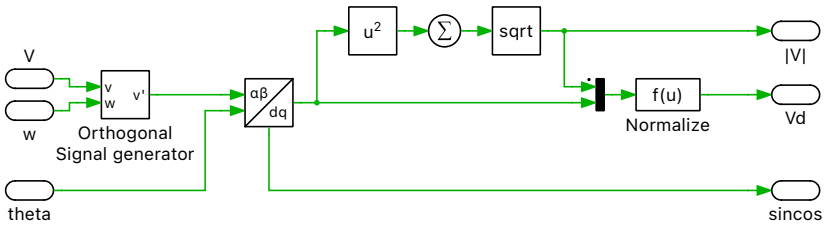


Fig. 16.79: General structure of the SOGI adaptive filter

This implementation uses an ortogonal signal generator based on second order generalized integrator (SOGI). Based on the input signal, two sine waves with a phase shift of 90° are generated. The first of the two components has the same phase and magnitude as the fundamental of the input signal. From the structure given in above figure the following relevant transfer functions of the structure can be derived:

$$\frac{V_\alpha(s)}{V(s)} = \frac{k\omega' s}{s^2 + k\omega' s + \omega'^2}$$

$$\frac{V_\beta(s)}{V(s)} = \frac{k\omega'^2}{s^2 + k\omega' s + \omega'^2}$$

To avoid problems due to input signal frequency fluctuations the resonance frequency ω' is provided by the estimated frequency of the PLL structure. The filtering effort can be adjusted with the filter gain k. If k decreases, the bandpass filter will become narrower resulting in heavier filtering, but resulting in a slower system response. Especially when working under distorted grid conditions reducing the filter gain k‘ may be beneficial.

Parameters

Basic

PLL type
Specifies the PLL type. The single-phase PLL can be of type `Enhanced PLL` or `SOGI`. For further explanations on the different PLL implementations see **PLL types** above.

Nominal frequency
The nominal frequency of the fundamental component in hertz or radians/second, see below.

Nominal input voltage
The estimated nominal peak voltage of the input signal in volts (V). This value is needed to correctly initialize all states in the PLL structure. If this information is not known, this parameter can be set to zero.

Initial phase-angle
The initial phase-angle of the input signal in rad, per unit $(\mathrm{p.u.})$ or degrees, see below. The parameter value should be in the range $[0, 2\pi]$, $[0, 1]$ or $[0, 360]$ respectively.

Sin/Cos output
Activates an additional signal output terminal (`on`) with an unitary signal synchronous with the input signal. In addition also a 90 degree delayed signal is provided. If the parameter is set to `off`, the signal terminal is hidden.

Units for frequency and phase
The frequency and phase can be expressed in terms of (`rad/s, rad`), (`Hz, p.u.`) or (`Hz, degrees`). If the phase is expressed in per unit $(\mathrm{p.u.})$, a value of 1 is equivalent to the period length of the nominal frequency. This parameter also changes the unit of the frequency output of the PLL component.

Controller design

Controller design
Specifies the controller design approach. If the parameter is set to `Basic`, the PI controller gains are automatically set to have a settling time of approximately 0.06 seconds. By using the `Advanced` approach, the user can specify the controller gains `Kp` and `Ki` and the filter gain `k` freely. Please note, that it is not required to scale the controller gains by the rated input signal amplitude since the input signal is normalized.

Proportional gain Kp
The proportional gain of the PI controller. This parameter is shown only if the **Controller design** parameter is set to `Advanced`.

Integral gain Ki
The integral gain of the PI controller. This parameter is shown only if the **Controller design** parameter is set to `Advanced`.

ANF gain k
Changes the filter bandwidth of the adaptive notch filter in the enhanced PLL structure. This parameter is shown only if the **Controller design** parameter is set to `Advanced` and the **PLL type** parameter to `Enhanced PLL`.

SOGI filter gain k
Changes the filter bandwidth of the SOGI structure. Reducing k leads to a narrower bandpass-filter. This parameter is shown only if the **Controller design** parameter is set to `Advanced` and the **PLL type** parameter to `SOGI`.

Probe Signals

Theta
The estimated phase-angle of the input signal.

Amplitude
The estimated amplitude of the input signal.

Frequency
The estimated frequency of the input signal.

Sin/Cos
A unitary fundamental signal running synchronous with the input signal and a 90 degree delayed signal of it.

Input Signal
The input signal of the PLL block.

References

• R. Teodorescu, et. al., “Grid Converters for Photovoltaic and Wind Power Systems”, John Wiley & Sons Ltd., 2011.

↪ *Added in PLECS 4.5.*

PLL (Three-Phase)

Purpose

Implementation of a three-phase PLL

Library

Control / Continuous

Description

This block implements two different types of a three-phase Phase Locked Loop (PLL). The PLL block provides an estimation of the phase-angle, amplitude and frequency of the three-phase input signal.

PLL Types

General PLL Structure

The general PLL structure shown of a three-phase PLL can be divided into three basic blocks:

- Phase detector: Generates an output signal that is proportional to the phase difference between the input signal and the signal generated by the PLL itself.
- Controller: Usually a PI controller to attenuate high-frequency components and to eliminate the steady-state phase-error.
- Frequency/phase-angle generator: Generates a phase-angle signal based on the estimated angular frequency. Typically, this block is constituted by a wrapping integrator.

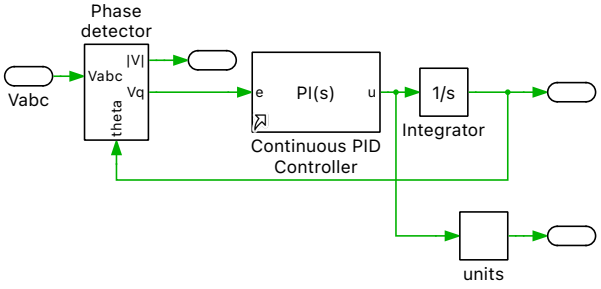


Fig. 16.80: General PLL block diagram

The main difference between the two offered three-phase PLLs lies in the phase detector part.

SRF-PLL

A basic synchronization technique in three-phase applications is the synchronously rotating reference frame PLL (SRF-PLL). The three-phase voltage vector in the natural reference frame is transformed in the rotating reference frame by using the Park transformation. The instantaneous phase angle is controlled by a feedback loop that regulates the **q** component to zero. Under steady-state operation, i.e. when the **q** component is zero, the **d** component depicts the amplitude of the input voltage vector. This simple approach works fine if the grid voltage is not affected by harmonics or unbalances. To avoid the impact of voltage harmonics on the accuracy of the estimated phase-angle and frequency, the controller bandwidth has to be set as high as possible. However, under unbalanced grid conditions, a double-frequency ripple of the input signal frequency is visible on the output signals of the PLL if the control-loop bandwidth is set to high. Therefore, a trade-off between those two control goals has to be made.

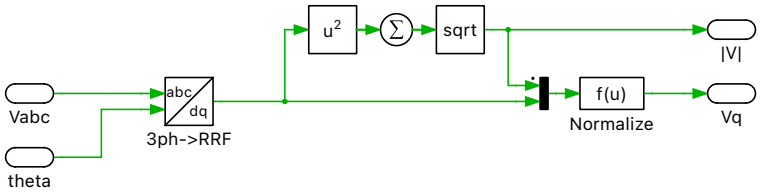


Fig. 16.81: Phase detector of the SRF-PLL

DSRF-PLL

The decoupled double synchronous reference frame PLL uses two synchronous reference frames, rotating with positive and negative synchronous speeds, respectively. This allows the decoupling of the effect of the negative sequence component on the dq-signals. This is of high interest when synchronizing to non-ideal grids, i.e unbalanced voltage conditions. Since the double-frequency ripple caused by the unbalanced grid condition is avoided, a higher control bandwidth compared to the SRF-PLL can be set.

The transfer function of the low-pass filter in Fig. 16.82 can be written as:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{k\omega_0}{s + k\omega_0}$$

where, k is the filter gain, and ω_0 is the angular nominal frequency. By

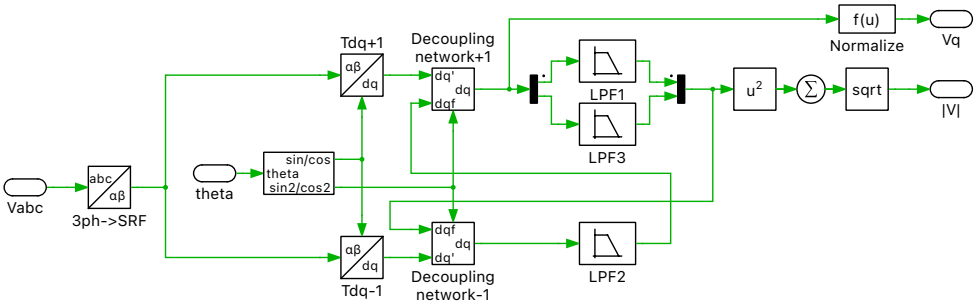


Fig. 16.82: Phase detector of the DSRF-PLL

decreasing the filter gain k, the bandwidth of the LPF can be reduced accordingly.

Phase Convention

The transformation used in the three-phase PLL block results in a 0 degrees position that is aligning with the peak of phase A voltage ($\cos(\theta)$ behaviour), as indicated in Fig. 16.83.

Parameters

Basic

PLL type

Specifies the PLL type. The three-phase PLL can be of type `SRF-PLL` or `DSRF-PLL`. For further explanations on the different PLL implementations see **PLL types** above.

Nominal frequency

The nominal frequency of the fundamental component in hertz or radians/second, see below.

Nominal input voltage

The estimated nominal peak voltage of the input signal in volts (V). This value is needed to correctly initialize all states in the PLL structure. If this information is not known, this parameter can be set to zero.

Initial phase-angle

The initial phase-angle of the input signal in radians, per unit $(\mathrm{p.u.})$ 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/s, rad`), (`Hz,`

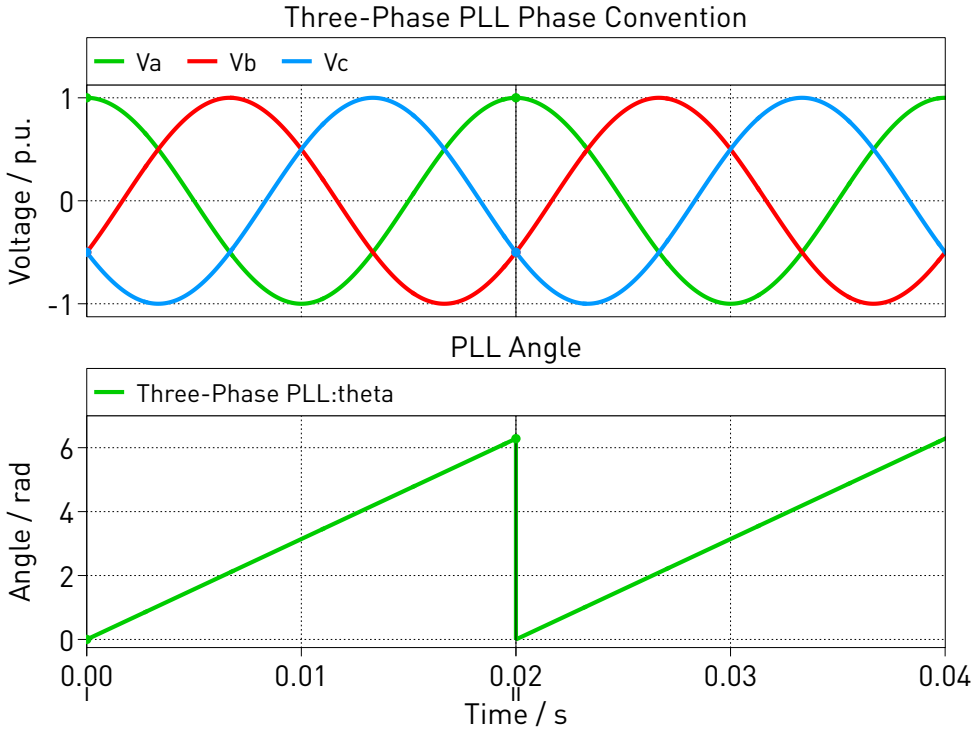


Fig. 16.83: PLL phase aligning with peak of phase A voltage

`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 of the nominal frequency. This parameter also changes the unit of the frequency output of the PLL component.

Controller design

Controller design

Specifies the controller design approach. If the parameter is set to `Basic`, the PI controller gains are automatically set. If the **PLL type** parameter is set to `SRF-PLL`, the PLL is designed to have a controller crossover frequency of half the nominal frequency defined in the parameter **Nominal frequency** and a phase margin of at least 60°. Otherwise, if the **PLL type** parameter is set to `DSRF-PLL`, the PLL is designed to have a con-

troller crossover frequency equal to the nominal frequency defined in the parameter **Nominal frequency** and a phase margin of at least $60°$. By using the `Advanced` approach, the user can specify the controller `Kp` and `Ki` and the filter gain `k` freely. Please note, the controller gains have not to be scaled by the rated input signal amplitude since the both schemes use an amplitude normalization scheme.

Proportional gain Kp
The proportional gain of the PI controller. This parameter is shown only if the **Controller design** parameter is set to `Advanced`.

Integral gain Ki
The integral gain of the PI controller. This parameter is shown only if the **Controller design** parameter is set to `Advanced`.

Filter coefficient k
The filter gain of the PI controller. This parameter is shown only if the **Controller design** parameter is set to `Advanced` and the **PLL type** parameter is set to `DSRF-PLL`.

Probe Signals

Theta
The estimated phase-angle of the input signal.

Amplitude
The estimated amplitude of the input signal.

Frequency
The estimated frequency of the input signal.

Input Signal
The input signal of the three-phase PLL block.

References

- R. Teodorescu, et. al., “Grid Converters for Photovoltaic and Wind Power Systems”, John Wiley & Sons Ltd., 2011.

↪ *Added in PLECS 4.5.*

Polar to Rectangular

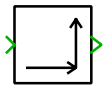
Purpose

Transform polar coordinates to rectangular coordinates

Library

Control / Math

Description



This block transforms a signal representing polar coordinates $[r, \theta]$ into rectangular coordinates $[x, y]$:

$$x = r\cos(\theta)$$
$$y = r\sin(\theta)$$

where θ is in radians.

Example Model

- See the example model **"Polar To Rectangular"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Position Sensor

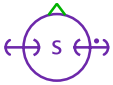
Purpose

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 Signals

Position
The measured position.

Product

Purpose

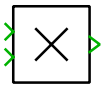
Multiply and divide input signals

Library

Control / Math

Description

The Product block multiplies or divides input signals, which may be scalars or vectors of the same size.



If the block has multiple inputs, it performs an element-wise multiplication or division of the input signals. Scalar inputs are expanded to the necessary width.



If the block has a single input, it calculates the product or the reciprocal of the product of the elements of the input signal. In this case the sign displayed on the block icon changes to $\prod$ or $\frac{1}{\prod}$.

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.

↪ *Added in PLECS 4.6.*

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.

Output data type

The data type of the output signal. See “Data Types” (p. 48). If you choose `inherited`, the minimum data type is `int8_t`.

Data type overflow handling

Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if **Output data type** is not set to a floating-point data type.

Probe Signals

Input i

The i-th input signal.

Output

The block output signal.

Pulse Delay

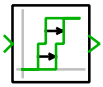
Purpose

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 (p. 1078)
- 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 (p. 1133) or a Delay (p. 639) 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 $T_o = \mathrm{mod}(T_D, T_s)$. 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” (p. 43).

Example Model

- See the example model **“Pulse Delay”**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Parameters

Time delay T_d
The time by which the input signal is delayed, in seconds (s).

Initial output
The output value after simulation start before the input values appear at the output.

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
The data type of the output signal. See “Data Types” (p. 48).

Probe Signals

Output
The output signal of the pulse generator.

Quantizer

Purpose

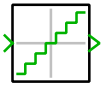
Apply uniform quantization to input signal

Library

Control / Discontinuous

Description

The Quantizer maps the input signal to an integer multiple of the quantization interval:



$$y = q \cdot \text{round}\left(\frac{u}{q}\right)$$

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

Library

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.

Parameters

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, in seconds (s).

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 Signals

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. Fig. 16.84 illustrates the distribution for two different sets of parameters.

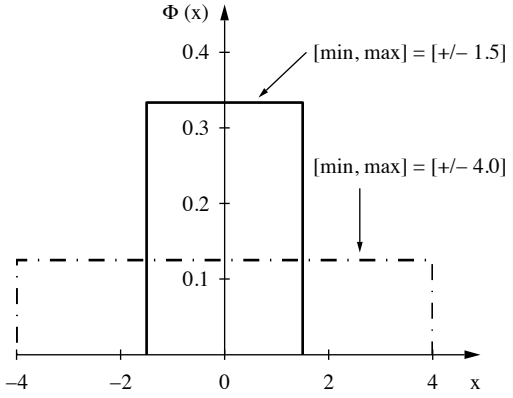


Fig. 16.84: Uniform distribution for different parameter sets

The seed of the generator initializes the algorithm at the simulation 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

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s), used for generating random output values. See also “Sample Times” (p. 43).

References

- Mersenne Twister: http://en.wikipedia.org/wiki/Mersenne_twister

Rate Limiter

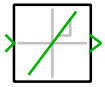
Purpose

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).

Sample time mode
This parameter lets you choose whether the block operates with a continuous sample time, or whether the sample time is inherited from the input. The default is `continuous`. See also “Sample Times” (p. 43).

↪ *Added in PLECS 4.4:* For older models,the default is `inherited`.

Probe Signals

Input
The input signal.

Output
The output signal.

Rectangular to Polar

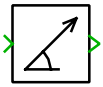
Purpose

Convert Cartesian coordinates to polar coordinates

Library

Control / Transformations

Description



This block transforms a signal representing rectangular coordinates $[x, y]$ into polar coordinates $[r, \theta]$:

$$r = \sqrt{x^2 + y^2}$$
$$\theta = \text{atan2}(y, x)$$

θ is calculated in the range $[-\pi, \pi]$.

Example Model

- See the example model **"Rectangular To Polar"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Relational Operator

Purpose

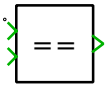
Compare two input signals

Library

Control / Logical

Description

The Relational Operator compares two input signals. If the comparison is true, the block outputs 1, otherwise 0. The first input is marked with a dot.



Parameters

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

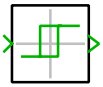
Purpose

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 inputs is less than the lower threshold. The relay does not change for input values between the thresholds.

Parameters

Upper threshold
The highest value that the input signal may reach before the state changes to the on-state.

Lower threshold
The lowest value that the input signal may reach before the state changes to the off-state.

On-state output
The value of the output signal while the relay is in the on-state.

Off-state output
The value of the output signal while the relay is in the off-state.

Initial state
The state of the relay at simulation start. Possible values are `on` and `off`.

Output data type
The data type of the output signal. See “Data Types” (p. 48).

Probe Signals

Input
The block input signal.

Output
The block output signal.

Resistor

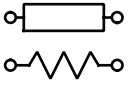
Purpose

Ideal resistor

Library

Electrical / Passive Components

Description



This component provides an ideal resistor between its two electrical terminals. See section “Configuring PLECS” (p. 60) for information on how to change the graphical representation of resistors.

Parameters

Resistance

The resistance in ohms (Ω). All positive and negative values are accepted, including `0` and `inf` (∞). 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 $[R_1\ R_2 \ldots R_n]$. 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).

RMS Value

Purpose

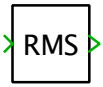
Calculate root mean square (RMS) value of input signal

Library

Control / Filters

Description

This block calculates the RMS value of a periodic input signal. The sample time, fundamental frequency and the initial condition can be specified.



Example Model

- See the example model **"Rms Value"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

Parameters

Initial condition

The initial condition describes the input signal before simulation start. If the input is a scalar signal, the parameter must be a scalar too. If input and output are vectorized signals, a vector can be used. The number of elements in the vector must match the number of input signals. The default value of this parameter is `0`.

Sample time

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in "Sample Times" (p. 43). If a sample time of `0` is specified, a continuous implementation based on the Moving Average block (p. 837) is active. The **Discrete-Periodic** implementation can potentially force a variable-step solver to take small integration steps. This may slow down the simulation. The default value of this parameter is `0`.

Fundamental frequency

The fundamental frequency of the periodic input signal in hertz (Hz).

↪ *Added in PLECS 4.4.*

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 "ω" measures the angular speed of the marked flange with respect to the unmarked one. The output signal "τ" 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 (p. 911) 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, Fig. 16.85 shows a possible implementation of a rotational damper with variable damping constant and prescribed initial displacement.

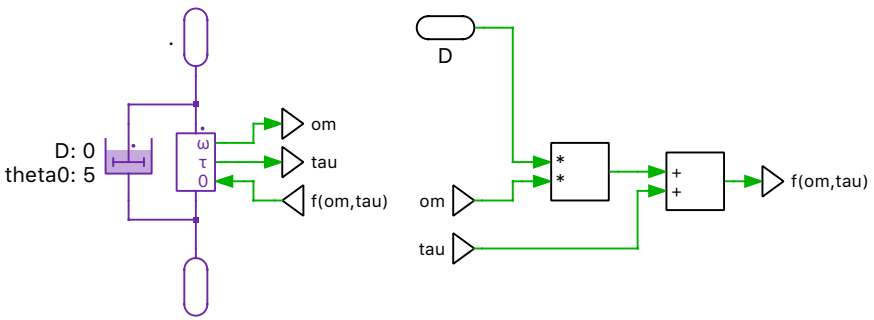


Fig. 16.85: Variable rotational damper implementation example

> **Note**
>
> The Rotational Algebraic Component creates an algebraic loop. See “Block Sorting” (p. 36) 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.

↪ *Added in PLECS 4.1.*

Rotational Backlash

Purpose

Ideal rotational backlash

Library

Mechanical / Rotational / Components

Description



The Rotational Backlash models an ideal symmetrical, two-sided Rotational Hard Stop (p. 915) in a rotational system, which restricts the relative displacement of the two flanges between an upper and lower limit of $\pm\frac{b}{2}$. 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.

Parameters

Total backlash

The total permitted displacement b between the flanges, in radians.

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 flowing from the unmarked to the marked flange, in newton-meters (Nm).

Displacement

The displacement 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 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 τ is the torque flow from the unmarked towards the marked flange and ω is the angular speed of the marked flange with respect to the unmarked one.

Parameters

Damper constant

The damping (viscous friction) constant D, in $(\frac{\mathrm{Nms}}{\mathrm{rad}})$.

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 τ_{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 τ_{C} and a speed-dependent viscous friction torque $c_{\mathrm{v}} \cdot \omega$.

Fig. 16.86 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.

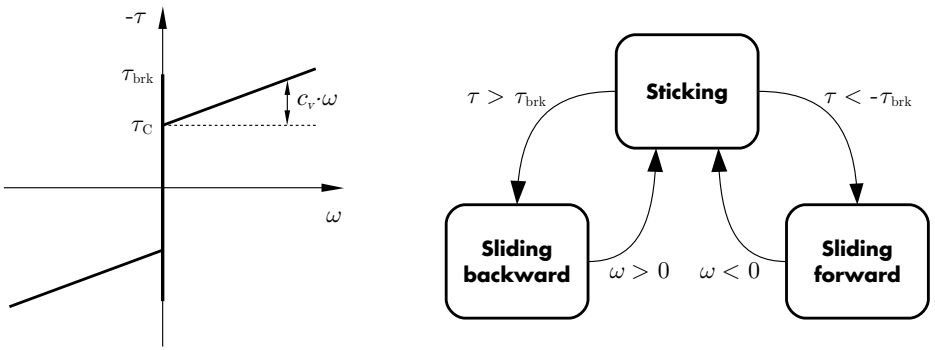


Fig. 16.86: Rotational friction speed/torque characteristic and state chart

Parameters

Breakaway friction torque

The maximum magnitude of the stiction torque τ_{brk}, in newton-meters (Nm). Must be greater than or equal to zero.

Coulomb friction torque

The magnitude of the (constant) Coulomb friction torque τ_{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 $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Probe Signals

Torque

The transmitted torque τ flowing from the unmarked to the marked flange, in newton-meters (Nm).

Speed

The angular speed ω of the marked flange with respect to the unmarked flange, in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

State

The internal state of the component: -1 sliding backward, 0 stuck, $+1$ sliding forward.

Rotational Friction (Controlled)

Purpose

Generate variable rotational friction

Library

Mechanical / Rotational / Components

Description



The Controlled Rotational Friction component models the friction between two flanges in a rotational system. It maintains zero relative velocity between the flanges up to the limit of the breakaway torque τ_{brk}. When the breakaway torque is exceeded, the flanges slide against each other and the component generates the Coulomb friction torque τ_{C}.

This component does not include viscous friction.

The breakaway torque τ_{brk} is determined by the control signal and must always be greater than or equal to zero. The magnitude of the Coulomb friction τ_{C} is proportional to the control signal.

Parameters

Relative Coulomb friction torque

The coefficient c_C determines the magnitude of the Coulomb friction torque $\tau_{\mathrm{C}} = c_C \cdot \tau_{\mathrm{brk}}$ and must be between zero and one.

Probe Signals

Torque

The transmitted torque τ flowing from the unmarked to the marked flange, in newton-meters (Nm).

Speed

The angular speed ω of the marked flange with respect to the unmarked flange, in $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

State

The internal state of the component: -1 sliding backward, 0 stuck, $+1$ sliding forward.

↪ *Added in PLECS 4.9.*

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.

Fig. 16.87 shows the displacement/torque characteristic and the state chart of the component.

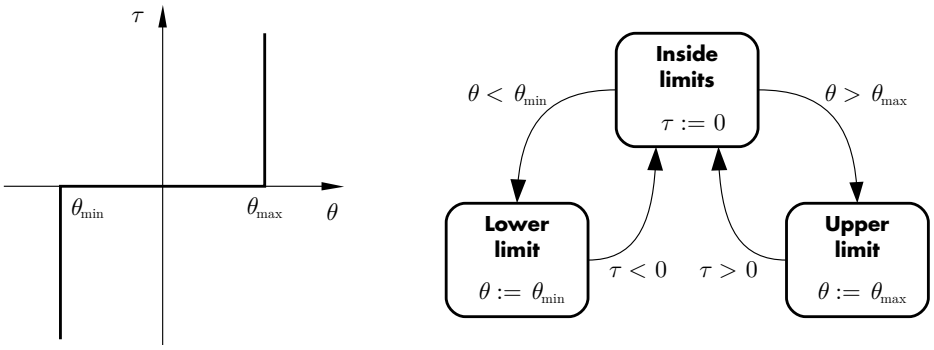


Fig. 16.87: Rotational hard stop displacement/torque characteristic and state chart

Parameters

Upper limit

The maximum displacement θ_{max} between the flanges. Set to `inf` to disable this limit.

Lower limit

The minimum displacement θ_{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 τ flowing from the unmarked to the marked flange, in newton-meters (Nm).

Displacement

The displacement θ 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

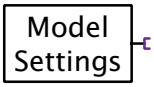
Purpose

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” (p. 342).

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 (p. 897) have coupled model equations. See also the options **Enable state-space splitting** and **Display state-space splitting** in the Simulation Parameters (p. 135).

Parameters

Switching algorithm

This parameter allows you two choose between two algorithms to determine the clutch states in the generated code. See “Switching Algorithm” (p. 343) 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.

↪ *Added in PLECS 4.0.*

Rotational Multiplexer

Purpose

Combine several rotational connections into single vector

Library

Mechanical / Rotational / Connectivity

Description



This multiplexer combines several rotational connections into one vector connection. The individual connections may themselves be vectors. In the block icon, the first individual connection is marked with a dot.

Parameters

Width

This parameter allows you to specify the number and/or width of the individual connections. You can choose between the following formats for this parameter:

Scalar: A scalar specifies the number of individual connections each having a width of 1.

Vector: The length of the vector determines the number of individual connections. Each element specifies the width of the corresponding individual connections.

↪ *Added in PLECS 4.6.*

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 (p. 977). 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.

A terminal can be moved around the edges of the subsystem by dragging it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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.

Parameters

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 input terminals are shown, and `right` refers to the side on which 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 Selector

Purpose

Select or reorder elements from vector connection

Library

Mechanical / Rotational / Connectivity

Description

The Rotational Selector block connects the individual elements of the output connection to the specified elements of the input connection. The input connection is marked with a dot.



Parameters

Input width

The width of the input connection.

Output indices

A vector with the indices of the input elements that the output connection should contain.

↪ *Added in PLECS 4.6.*

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 $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$. The default value is `1`.

Probe Signals

Torque

The generated torque τ flowing from the unmarked to the marked flange, in newton-meters (Nm).

Speed

The angular speed, in $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Rotational Speed (Controlled)

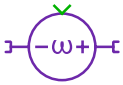
Purpose

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 τ flowing from the unmarked to the marked flange, in newton-meters (Nm).

Speed

The angular speed, in $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

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.

Parameters

Second flange
Controls whether the second flange is accessible or connected to the rotational reference frame.

Probe Signals

Speed
The measured angular speed, in $(\frac{\mathrm{rad}}{\mathrm{s}})$.

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 $(1.7) = 1$ and $(-1.3) = -2$.

ceil
The output is the smallest integer not less than the input, for example $(1.3) = 2$ and $(-1.7) = -1$.

round
The output is the integer nearest to the input, for example $(1.4) = 1$, $(-1.3) = -1$ and $(-1.5) = -2$.

fixed
The output is the integer value of the input with all decimal places truncated, for example $(1.7) = 1$ and $(-1.7) = -1$.

Parameters

Operation
The rounding algorithm as described above.

Output data type
The data type of the output signal. See “Data Types” (p. 48).

Data type overflow handling
Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if is not set to a floating-point data type.

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 / Components

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:

$\tan^{-1}$ **fit**

The $\tan^{-1}$ 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 Langevian equation for bulk magnetization without interdomain coupling, and is given as:

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

Both fitting functions have three degrees of freedom which are set by the coefficients μ_{sat}, B_{sat} and a. μ_{sat} is the fully saturated permeability, which usually corresponds to the magnetic constant μ_0, i.e. the permeability of air. B_{sat} defines the knee of the saturation transition between unsaturated and saturated permeability:

$$B_{\text{sat}} = (B - \mu_{\text{sat}} H)\Big|_{H\to\infty}$$

The coefficient a is determined by the unsaturated permeability μ_0 at $H = 0$:

$$a = B_{\text{sat}} / (\mu_{\text{unsat}} - \mu_{\text{sat}})$$

Fig. 16.88 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 $\tan^{-1}$ fit.

Parameters

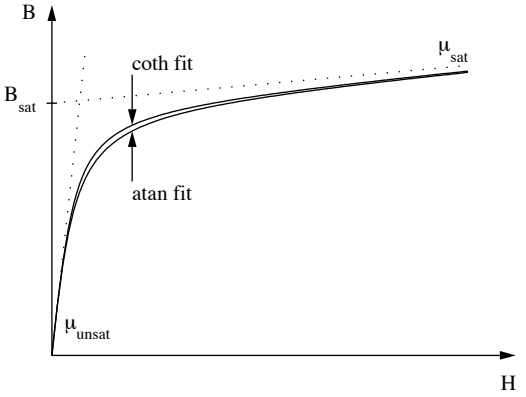


Fig. 16.88: Comparison of $\coth$ and $\tan^{-1}$ saturation characteristics

Fitting functions
Saturation characteristic modeled with $\tan^{-1}$ or $\coth$ fit.

Cross-sectional area
Cross-sectional area A of the flux path, in square meters (m^2).

Length of flux path
Length l of the flux path, in meters (m).

Unsaturated rel. permeability
Relative permeability $\mu_{\mathrm{r,unsat}} = \mu_{\mathrm{unsat}}/\mu_0$ of the core material for $H \to 0$.

Saturated rel. permeability
Relative permeability $\mu_{\mathrm{r,sat}} = \mu_{\mathrm{sat}}/\mu_0$ of the core material for $H \to \infty$.

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 amperes per meter ($\mathrm{A/m}$).

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. It is shown in Fig. 16.89.

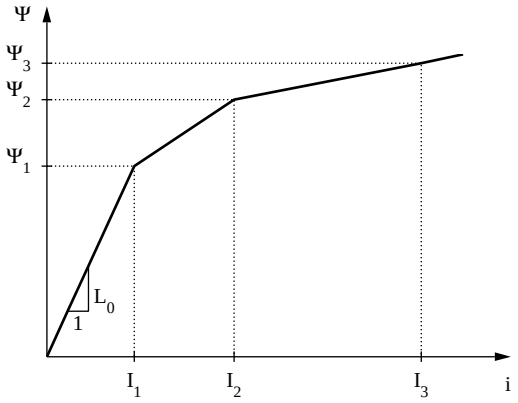


Fig. 16.89: Saturable inductor saturation characteristic

The operating mode of the saturable inductor is illustrated in Fig. 16.90.

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 $\mathrm{d}\Psi/\mathrm{d}i$ of the component thus becomes $L_{\mathrm{diff}} = \frac{L_0 L_1}{L_0 + L_1}$.

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.

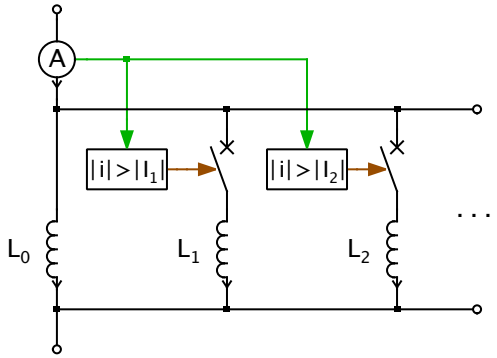


Fig. 16.90: Saturable inductor equivalent circuit

Parameters

Current values

A vector of positive current values I in amperes (A) defining the piece-wise linear saturation characteristic. The current values must be positive and strictly monotonic increasing. At least one value is required.

Flux values

A vector of positive flux values Ψ in (Vs) defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Initial current

The initial current through the inductor at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The direction of a positive initial current is indicated by a small arrow at one of the terminals. The initial current default is `0`.

Probe Signals

Inductor current

The current flowing through the inductor, in amperes (A). The direction of a positive current is indicated with a small arrow at one of the terminals.

Inductor voltage

The voltage measured across the inductor, in volts (V).

Saturation level

The saturation level indicates which sector of the piece-wise linear characteristic is currently applied. During linear operation, i.e. operation in the

first sector, the saturation level is 0. The saturation level is negative for negative flux and current values.

Saturable Transformers

Purpose

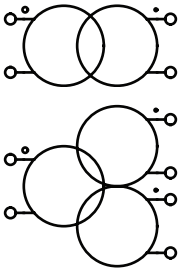
Single-phase transformers with two resp. three windings and core saturation

Library

Electrical / Passive Components

Description

These transformers model two or three coupled windings on the same core.



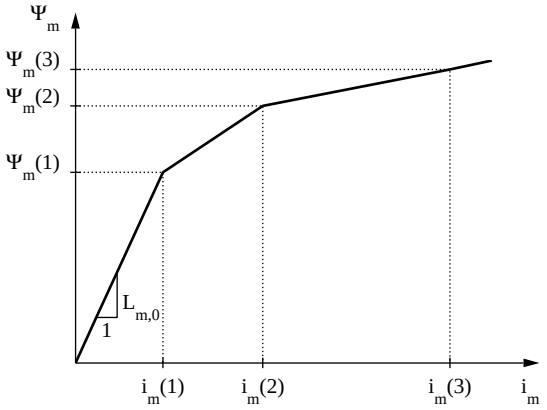


Fig. 16.91: Magnetizing inductance saturation characteristic

The core saturation characteristic (Fig. 16.91) is piece-wise linear and is modeled using the Saturable Inductor (p. 931). The magnetizing current i_m and flux Ψ_m value pairs are referred to the primary side. To model a transformer without saturation enter 1 as the magnetizing current values and the desired magnetizing inductance L_m as the flux values. A stiff Simulink solver is recommended if the iron losses are not negligible, i.e. R_fe is not infinite.

In the transformer symbol, the primary side winding is marked with a little circle. The secondary winding is marked with a dot at the outside terminal, the tertiary winding with a dot at the inside terminal.

Parameters

Leakage inductance

A vector containing the leakage inductance of the primary side L_1, the secondary side L_2 and, if applicable, the tertiary side L_3. The inductivity is given in henries (H).

Winding resistance

A vector containing the resistance of the primary winding R_1, the secondary winding R_2 and, if applicable, the tertiary winding R_3, in ohms (Ω).

No.of turns

A vector containing the number of turns of the primary winding n_1, the secondary winding n_2 and the tertiary winding n_3, if applicable.

Magnetizing current values

A vector of positive current values in amperes (A) defining the piece-wise linear saturation characteristic of the transformer legs. The current values must be positive and strictly monotonic increasing. At least one value is required.

Magnetizing flux values

A vector of positive flux values in (Vs) defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Core loss resistance

An equivalent resistance R_{fe} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

Initial current

A vector containing the initial currents on the primary side i_1, the secondary side i_2 and the tertiary side i_3, if applicable. The currents are given in amperes (A) and considered positive if flowing into the transformer at the marked terminals. The default is `[0 0 0]`.

Saturation

Purpose

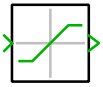
Limit input signal to upper and/or lower value

Library

Control / Discontinuous

Description

The saturation block limits a signal to an upper and/or lower value. If the input signal is within the saturation limits, the output signal is identical to the input signal.



Parameters

Upper limit

The highest value that the input signal may reach before the output signal is clipped. If the value is set to `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

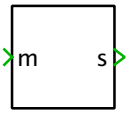
Purpose

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 (3ph) (p. 750) or the Ideal Converter (3ph) (p. 743). In these cases the modulation index must have a width of 3 to match the number of inverter legs.

Fig. 16.92 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.

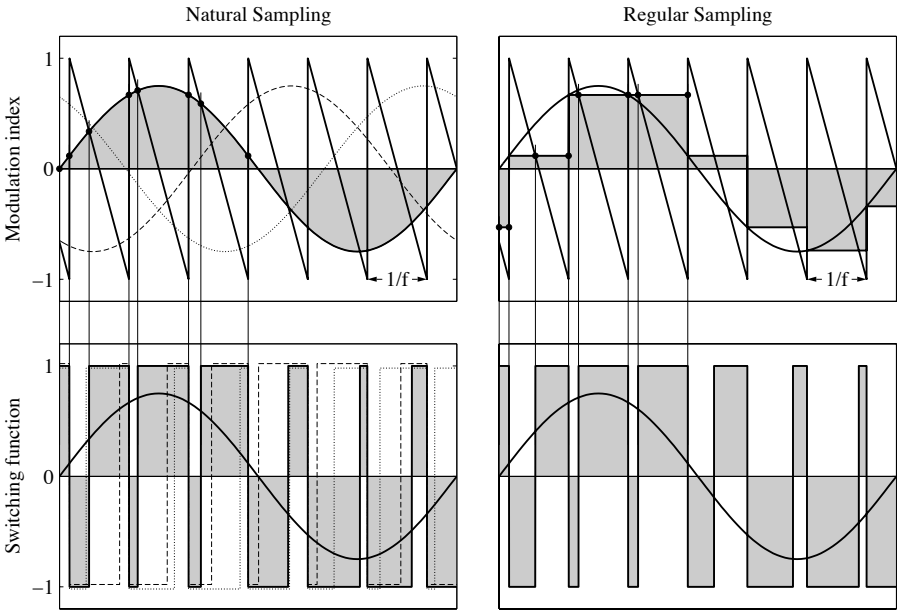


Fig. 16.92: Sawtooth PWM waveforms

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 hertz (Hz).

Carrier phase shift
The time offset of the carrier signal, in per unit $(\mathrm{p.u.})$ of the carrier period.

Carrier limits
The range of the sawtooth carrier. 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)

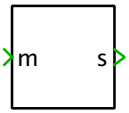
Purpose

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 IGBT 3-Level Converter (3ph) (p. 748) or the Ideal Converter (3ph) (p. 743). In these cases the modulation index must have a width of 3 to match the number of inverter legs.

Fig. 16.93 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.

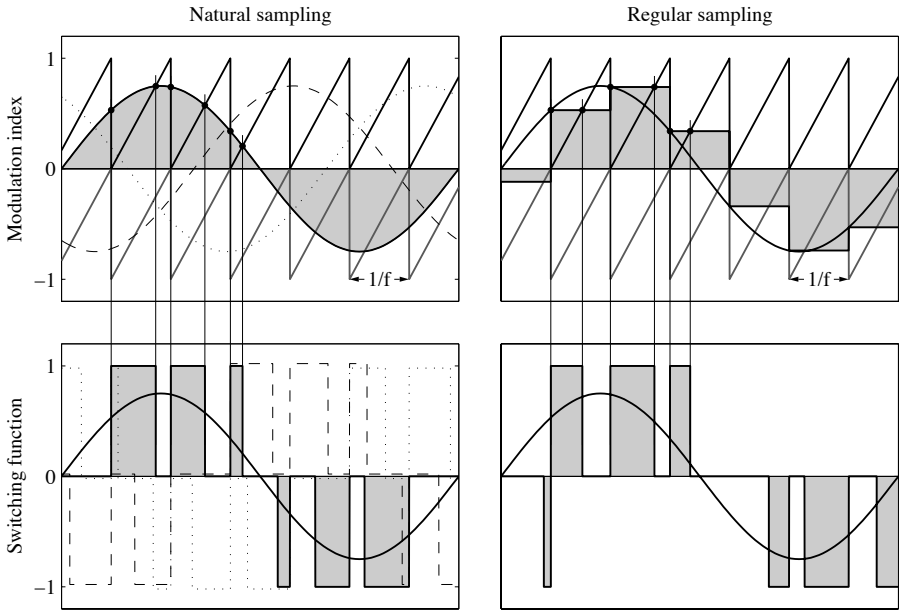


Fig. 16.93: 3-level sawtooth PWM waveforms

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 hertz (Hz).

Carrier phase shift
The time offset of the carrier signal, in per unit $(\mathrm{p.u.})$ of the carrier period.

Carrier limits
The range of the sawtooth carriers. The default is `[-1 1]`.

Scalar Expander

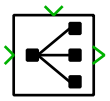
Purpose

Expand a scalar input signal to a vector signal with the width of the reference signal. If the input signal has the same width as the reference signal, it is passed through unchanged.

Library

System

Description



The Scalar Expander block expands a scalar input signal to a vector signal with the width of the reference signal. In that case, every component of the output vector signal equals the input signal. If the input signal happens to be a vector signal of the same width as the reference signal, the input signal is passed through unchanged. If the input signal is a vector signal of a different width than the reference signal, an error will be reported.

↪ *Added in PLECS 4.9.*

Scope

Purpose

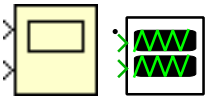
Display simulation results versus time

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” (p. 119).

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

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s), 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 also section “Sample Times” (p. 43).

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, in seconds (s). 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” (p. 171).

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in $(\mathrm{K/W})$. The default is `0`.

↪ *Added in PLECS 4.6.*

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” (p. 177).

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.



Parameters

Number of outputs

This parameter allows you to specify the number and width of the output signals. You can choose between the following formats for this parameter:

Scalar: A scalar specifies the number of scalar outputs. If this format is used, all output signals have a width of 1.

Vector: The length of the vector determines the number of outputs. Each element specifies the width of the corresponding output signal.

Signal From

Purpose

Reference signal from Signal Goto block by name

Library

System

Description



The Signal From block references another signal from a Signal Goto block. All Signal From blocks connect to the Signal Goto block with the same tag within the given scope. If no matching Signal Goto block is found, PLECS can report an error or warning message when a simulation is started.

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.

To quickly create a Signal Goto block with matching tag name and scope, choose **Create "Signal Goto"** from the context menu.

↪ *Added in PLECS 4.9.*

To quickly change the tag names of all matching Signal From and Signal Goto blocks, choose **Rename all matching tags...** from the context menu.

↪ *Added in PLECS 4.9.*

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. The setting `Code-Gen Subsystem` causes a lookup within the hierarchy of the CodeGen subsystem in which the block is contained. If the block is not contained in a CodeGen subsystem, a global lookup is done.

Action if there is no matching counterpart

A Signal From block that does not have a matching Signal Goto block typically indicates a modeling error. This parameter lets you specify whether PLECS should report an error or a warning or ignore this condition when a simulation is started.

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.

To quickly create a Signal From block with matching tag name and scope, choose **Create "Signal From"** from the context menu.

↪ *Added in PLECS 4.9.*

To quickly change the tag names of all matching Signal From and Signal Goto blocks, choose **Rename all matching tags...** from the context menu.

↪ *Added in PLECS 4.9.*

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 Goto 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. The setting `Code-Gen Subsystem` causes a lookup within the hierarchy of the CodeGen subsystem in which the block is contained. If the block is not contained in a CodeGen subsystem, a global lookup is done.

Action if there is no matching counterpart

A Signal Goto block that does not have a matching Signal From block often indicates a modeling error. This parameter lets you specify whether PLECS should report an error or a warning or ignore this condition when a simulation is started.

Signal Inport

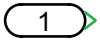
Purpose

Add signal input connector to subsystem

Library

System

Description



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 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 around the edges of a subsystem, drag it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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
The data type of the output signal. See “Data Types” (p. 48). This parameter appears only if the block is placed in a top-level schematic or in an atomic subsystem (see “Virtual and Atomic Subsystems” (p. 978)).

Data type overflow handling
Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if **Output data type** is not set to a floating-point data type.

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.



Parameters

Number of inputs

This parameter allows you to specify the number and width of the input signals. You can choose between the following formats for this parameter:

Scalar: A scalar specifies the number of scalar inputs to the block. If this format is used, the block accepts only signals with a width of 1.

Vector: The length of the vector determines the number of inputs. Each element specifies the width of the corresponding input signal.

Signal Outport

Purpose

Add signal output connector to subsystem

Library

System

Description



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 around the edges of a subsystem, drag it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

Parameters

Width

The width of the output signal. The default `auto` means that the width is inherited from connected blocks.

Output when disabled

This parameter appears only if the block is placed in an enabled subsystem (see the Enable block (p. 678)) and determines the output signal while the subsystem is disabled. If you select `held`, the output will hold the most recent value before the subsystem was disabled. If you select `reset`, the output will be reset to the initial output value (see below).

Initial output

This parameter appears only if the block is placed in an enabled and/or triggered subsystem (see the Enable block (p. 678) and the Trigger block (p. 1083)) and specifies the initial output before the subsystem is first enabled or triggered.

↪ *Added in PLECS 4.4.*

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.



See also the Dynamic Signal Selector block (p. 669).

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

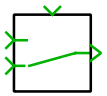
Purpose

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 >= Threshold`
- `u > Threshold`
- `u = Threshold`

Threshold

The threshold value used for the switch criteria.

Probe Signals

Inputs

Input signal and control signal.

Output

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

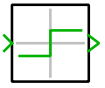
Provide sign of input signal

Library

Control / Math

Description

The Signum block outputs 1 for positive, -1 for negative and 0 for zero input values.



Probe Signals

Input
The block input signal.

Output
The block output signal.

Simple Dual Active Bridge

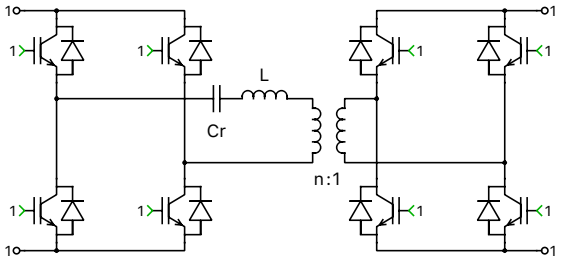
Purpose

Dual active bridge converter without magnetizing inductance, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated dual active bridge (DAB) converter with an optional resonant capacitor. For a DAB with magnetizing inductance, refer to the CLLLC Converter (p. 607) component.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 1, indicating that it occupies $\frac{1}{3}$ of a Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.94.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Primary inductance
A non-zero scalar specifying the primary-side inductance L, including the transformer leakage inductance and any external series-connected inductor, in henries (H).

Winding resistance
A scalar specifying the winding resistance R_L, in ohms (Ω).

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a resonant capacitor at the primary side.

Resonant capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C, in farads (F).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.94 for the probe signal positions.

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor current
The current through the inductor L, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

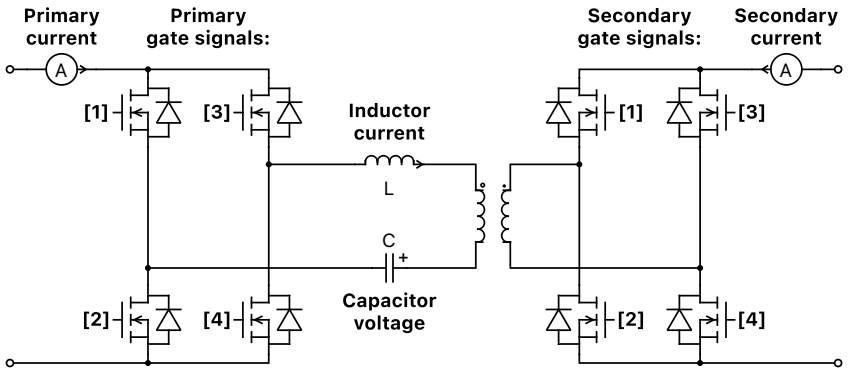


Fig. 16.94: Probes for the Simple Dual Active Bridge

↪ *Added in PLECS 4.9.*

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

$$y = A \cdot \sin(\omega \cdot t + \varphi) + C \qquad \omega = 2\pi f$$

If a variable-step solver is used, the solver step size is automatically limited to ensure that a smooth waveform is produced.

Example Model

- See the example model **"Sine Wave Generator"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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 radians/second (ω), see below.

Phase
The phase φ 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/s, 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 Signals

Output
The block output signal.

Small Signal Gain

This block is included only in the PLECS Standalone library.

Purpose

Measure loop gain of closed control loop using small-signal analysis

Library

Control / Small Signal Analysis

Description



This block uses the Small Signal Perturbation block and the Small Signal Response (p. 963) block to inject a perturbation into a feedback loop and measure the system response. To see the implementation choose **Look under mask** from the **Edit > Subsystem** menu or the block's context menu.

For detailed information regarding small-signal analysis see chapter "Analysis Tools" (p. 227).

Parameters

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.

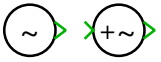
Purpose

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” (p. 227).

Parameters

Show feed-through input

When set to `on`, the block displays an input port. The output signal is the sum of the input signal and the perturbation. The default is `off`.

Small Signal Response

This block is included only in the PLECS Standalone library.

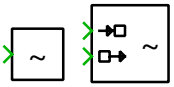
Purpose

Measure system response for small-signal analysis

Library

Control / Small Signal Analysis

Description



During a small-signal analysis that references this block, it records the signal(s) that are connected to the block input(s) in order to calculate the transfer function

$$G(s) = \frac{Y(s)}{U(s)}$$

If the reference input is shown, $U(s)$ is calculated from the signal that is connected to it. Otherwise, $U(s)$ is calculated from the perturbation signal generated by the corresponding Small Signal Perturbation block.

For detailed information regarding small-signal analysis see chapter “Analysis Tools” (p. 227).

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

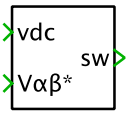
Purpose

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, $\overrightarrow{V}_{\mathrm{s}}$, at the ac terminals of a three-phase voltage source converter shown below. The reference vector is defined in the $\alpha\beta$ coordinate system: $\overrightarrow{V}_{\mathrm{s}} = V_{\alpha}^{*} + j\,V_{\beta}^{*}$.

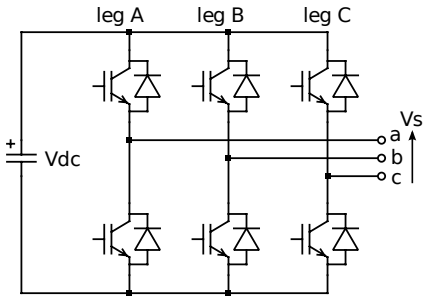


Fig. 16.95: Three-phase voltage source converter

Operation

The conventional construction of the reference voltage vector, $\overrightarrow{V}_{\mathrm{s}}$, is graphically depicted in Fig. 16.96.

Rather than calculating the relative on-times $\tau_{\mathrm{a}}, \tau_{\mathrm{b}}, \tau_{0}$ for the switching vectors $\overrightarrow{V}_{\mathrm{a}}, \overrightarrow{V}_{\mathrm{b}}$ and the zero vector, this block uses an equivalent index-based modulation approach to achieve different SVPWM modulation strategies by manipulating the three-phase sinusoidal modulation indices with different injected zero-sequence patterns.

This block is implemented with a 3-Phase Index-Based Modulation (p. 524) block in series with a Symmetrical PWM (p. 988) block. The Symmetrical PWM block is configured to use the `Regular (single update at min.)` sampling scheme, where the incoming modulation index is sampled only at the minimum of the symmetrical triangular carrier.

The following figures show the switching patterns for different SVPWM strategies. Every figure shows only one switching cycle that resides in the first half of

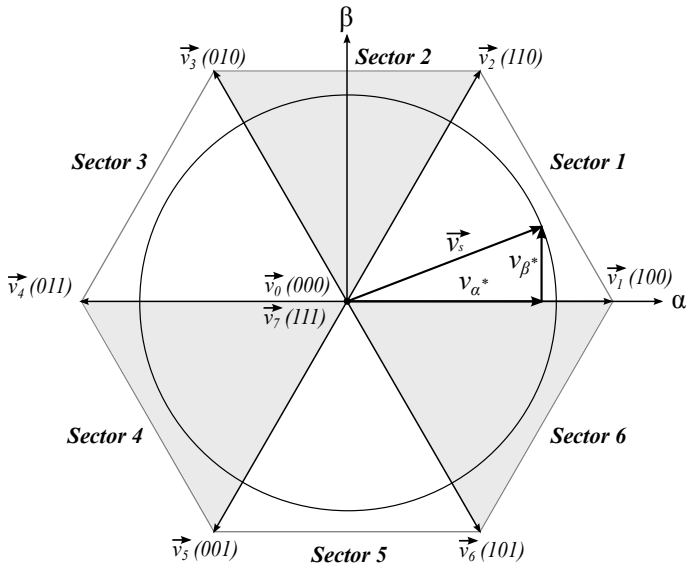


Fig. 16.96: Construction of the reference vector

Sector 1, i.e. between 0 and 30 degrees of the AC period. This is when $u_{\mathrm{a}}(t) > 0 > u_{\mathrm{b}}(t) > u_{\mathrm{c}}(t)$.

- The Symmetrical strategy always uses both zero vectors $\overrightarrow{V_0}$ (000) and $\overrightarrow{V_7}$ (111), by placing one in the center of the period and the other one evenly split at the beginning and the end of the period.
- DPWM1, DPWM2 and DPWMMAX happen to result in the same switching pattern inside this 30-degree interval. In this interval they only utilize one zero vector $\overrightarrow{V_7}$ (111), evenly split at the beginning and the end of the period.
- DPWM0, DPWM3 and DPWMMIN happen to result in the same switching pattern inside this 30-degree interval. In this interval they only utilize one zero vector $\overrightarrow{V_0}$ (000), placed in the center of the period.

To see how the modulation index pattern evolves throughout the other sectors for each modulation strategy, please refer to the documentation of the 3-Phase Index-Based Modulation block (p. 524).

Parameters

Modulation strategy

The modulation strategy can be set to `Symmetrical`, `DPWM0`, `DPWM1`, `DPWM2`, `DPWM3`, `DPWMMIN`, and `DPWMMAX` using a combo box. All these strategies follow the same space vector modulation theory, the major difference is in

the utilization of the two available zero vectors - $\overrightarrow{V_0}$ (000) and $\overrightarrow{V_7}$ (111) inside each switching period regarding different sector intervals.

Switching frequency

The switching frequency in hertz (Hz).

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, $[S_{\mathrm{a}}, S_{\mathrm{b}}, S_{\mathrm{c}}]$, which control the inverter legs A, B, and C. Each switch signal controls the upper and lower switches in the respective leg.

Probe Signals

3-phase modulation index

A vector signal consisting of the three-phase modulation indices, $[m_{\mathrm{a}}, m_{\mathrm{b}}, m_{\mathrm{c}}]$ for the chosen space vector modulation strategy.

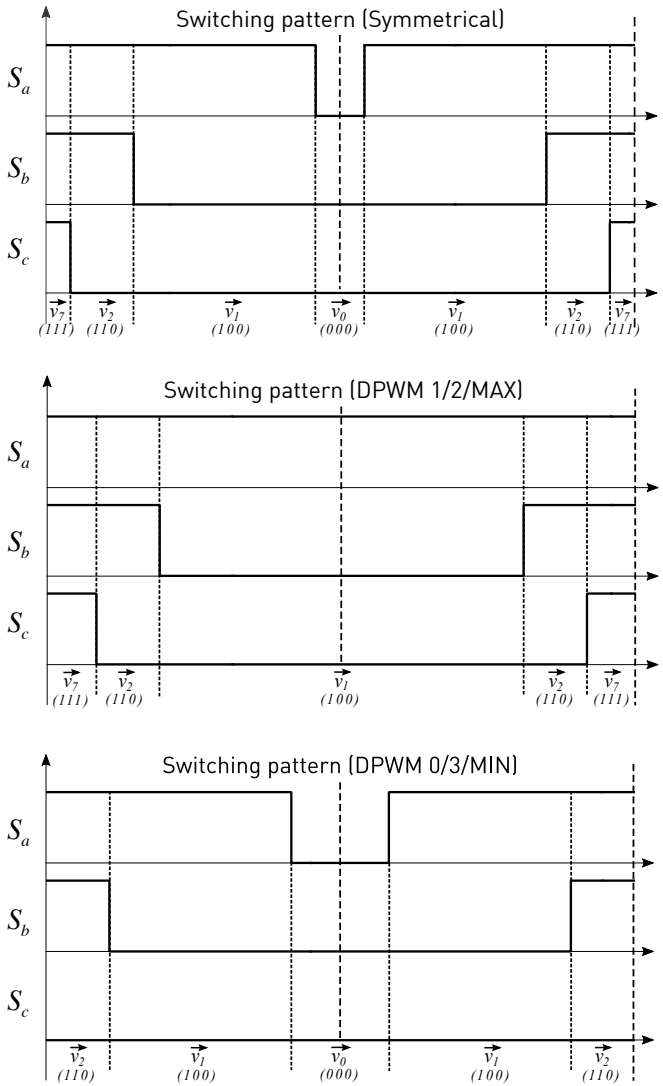


Fig. 16.97: Switching pattern for different SVPWM modulation strategies, example for one switching cycle inside the first half of Sector 1 with an angle between 0 to 30 degrees of the AC period (i.e. $u_{\mathrm{a}}(t) > 0 > u_{\mathrm{b}}(t) > u_{\mathrm{c}}(t)$)

Space Vector PWM (3-Level)

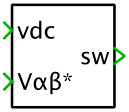
Purpose

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.

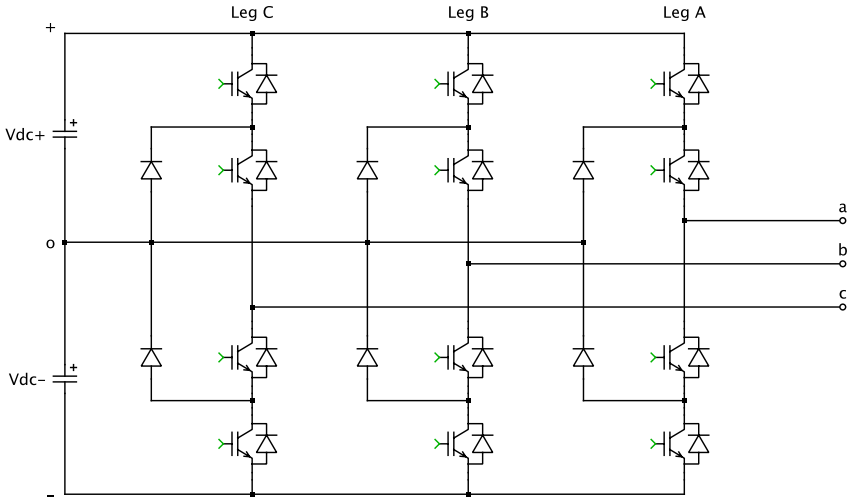


Fig. 16.98: Neutral-point clamped 3-phase inverter schematic

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 $V_{\mathrm{dc}+}$ and $V_{\mathrm{dc}-}$ the space-vector diagram is graphically depicted in Fig. 16.99.

The hexagon area can be divided in to 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

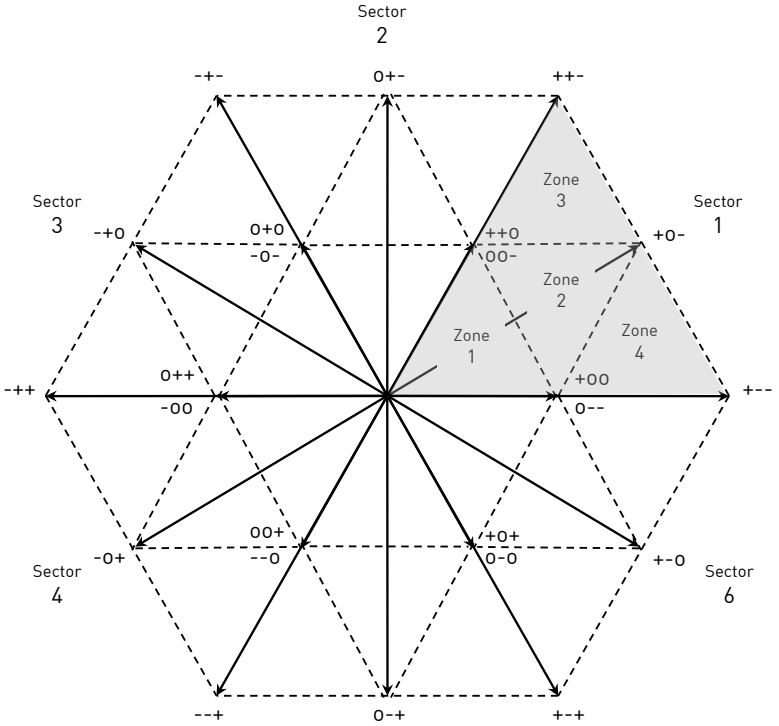


Fig. 16.99: 3-level space vector diagram

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:

$$\tau_\mathrm{a} = 1 - 2k\,\sin\left(\theta\right)$$

$$\tau_\mathrm{b} = 2k\,\sin\left(\frac{\pi}{3} + \theta\right) - 1$$

$$\tau_\mathrm{c} = 1 - 2k\,\sin\left(\frac{\pi}{3} - \theta\right)$$

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 (+oo) or (o–). In order to keep the dc

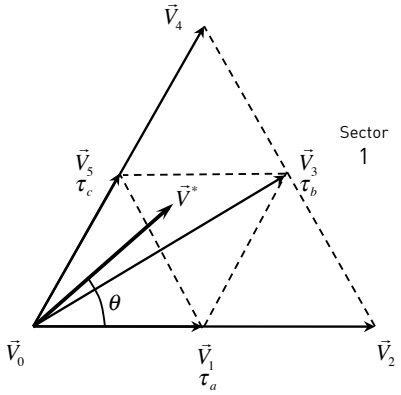


Fig. 16.100: Sector 1 Zone 2 vector diagram

link voltages balanced, both switch states must be applied for the same duration during one switching period. The resulting switch pattern is illustrated in Fig. 16.101.

Parameters

Switching frequency
The switching frequency in hertz (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 $V_{\mathrm{dc}+}$ and $V_{\mathrm{dc}-}$.

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 $[S_{\mathrm{a}}, S_{\mathrm{b}}, S_{\mathrm{c}}]$, 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.

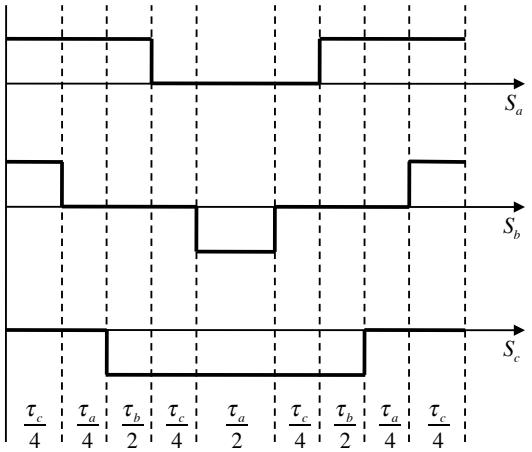


Fig. 16.101: 3-level switching pattern

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, $[\tau_\mathrm{a}, \tau_\mathrm{b}, \tau_\mathrm{c}]$.

sw

A vector signal consisting of the three gate signals for the inverter legs, $[S_\mathrm{a}, S_\mathrm{b}, S_\mathrm{c}]$.

SR Flip-flop

Purpose

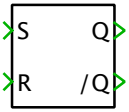
Implement set-reset flip-flop

Library

Control / Logical

Description

The SR Flip-flop behaves like a pair of cross-coupled `NOR` logic gates. The output values correspond to the following truth table:



S	R	Q	/Q
0	0	No change	No change
0	1	0	1
1	0	1	0
1	1	Restricted (0)	Restricted (0)

The combination $S = R = 1$ is restricted because both outputs will be set to 0, violating the condition $Q = \mathbf{not}(/Q)$. If both inputs change from 1 to 0 in the same simulation step, Q will be set to 0 and $/Q$ to 1.

Parameters

Initial state

The state of the flip-flop at simulation start.

Probe Signals

S

The input signal S.

R

The input signal R.

Q

The output signals Q.

/Q

The output signals $/Q$.

State 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” (p. 295).

↪ *Added in PLECS 4.0.*

State Space

Purpose

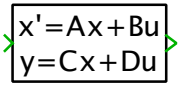
Model linear time-invariant system as state-space model

Library

Control / Continuous

Description

The State Space block models a state-space system of the form



$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the input vector, and $\mathbf{y}$ is the output vector. The widths of the input and output signals are determined by the number of columns of the $\mathbf{B}$ matrix and the number of rows of the $\mathbf{C}$ matrix, respectively.

For a single-input system (i.e. if the $\mathbf{B}$ matrix has only one column), the State Space block performs scalar expansion: If the input signal is a vector, the state-space model is applied to each element of the input vector individually, and the output signal is a concatenated vector of all system outputs.

↪ *Added in PLECS 4.9.*

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 Fig. 16.102:

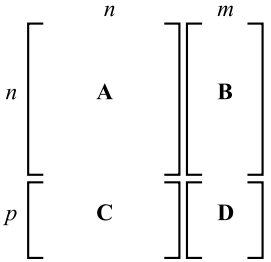


Fig. 16.102: State space system matrix dimensions

where n is the number of states, m is the width of the input signal and p is the width of the output signal.

If the matrix D is empty, it is internally expanded to a zero matrix of the size $p \times m$.

Initial condition
A vector of initial values for the state vector, $\mathbf{x} = [x_0, x_1 \ldots x_n]$.

> **Note**
>
> A scalar initial condition will be applied to all internal states.

> **Note**
>
> In case of scalar expansion (the system is a single-input system with multiple input signals), the initial condition can also be a matrix, where each row defines the initial condition for the individual inputs.
> ↪ *Added in PLECS 4.9.*

Probe Signals

Input
The input vector, $\mathbf{u}$.

Output
The output vector, $\mathbf{y}$.

State
The state vector, $\mathbf{x}$.

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, in seconds (s).

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
The data type of the output signal. See “Data Types” (p. 48).

Probe Signals

Output
The block output signal.

Subsystem

Purpose

Create functional entity in hierarchical simulation model

Library

System

Description



A subsystem block represents a system within another system. In order to create a subsystem, copy the subsystem block from the library into your schematic. You can then open the subsystem block and copy components into the subsystem's window.

The input, output, and 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.

A terminal can be moved around the edges of the subsystem by dragging it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

Note

A subsystem can be converted to a configurable subsystem (see the Configurable Subsystem block (p. 620)) by selecting **Convert to configurable subsystem** from the **Edit > Subsystem** menu or the block's context menu.

Example Model

- The example model **"Enabled Subsystem"** features *enabled* subsystems.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Example Model

- The example model **"Triggered Subsystem"** features *triggered* subsystems.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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" (p. 36)).

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 (p. 678) and the Trigger block (p. 1083)).

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 **Edit > Subsystem** 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").

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" (p. 36)). 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

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). 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. See also section "Sample Times" (p. 43).

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" (p. 348)). Note that it is not possible to enable code generation for a subsystem that is contained by or that itself contains other subsystems that enable code generation. 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, in seconds (s), and is used to discretize the physical model equations (see "Physical Model Discretization" (p. 39)) 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 `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" (p. 123)), 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 Dialog" (p. 98) for more details).

Sum

Purpose

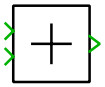
Add and subtract input signals

Library

Control / Math

Description

The Sum block adds or subtracts input signals, which may be scalars or vectors of the same size.



If the block has multiple inputs, it performs an element-wise summation or subtraction of the input signals. Scalar inputs are expanded to the necessary width.



If the block has a single input, it calculates the positive or negative sum of the elements of the input signal. In this case the sign displayed on the block icon changes to $\sum$ or $-\sum$.

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 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.

Output data type
The data type of the output signal. See “Data Types” (p. 48). If you choose `inherited`, the minimum data type is `int8_t`.

Data type overflow handling
Specifies how a data type overflow is handled. See “Data Types” (p. 48). This parameter only appears if **Output data type** is not set to a floating-point data type.

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.



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`.

Switch model (CPU code generation)

Select the switch model in the generated code.

Fundamental grid frequency

Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is `0`.

Probe Signals

Switch conductivity

Conduction state of the switch. The signal outputs 0 if the switch is open, and 1 if it is closed.

Switched Reluctance Machine

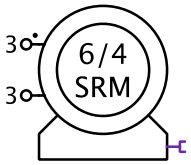
Purpose

Detailed model of switched reluctance machine with open windings

Library

Electrical / Machines

Description



These components represent analytical models of three common switched reluctance machine types: three-phase 6/4 SRM, four-phase 8/6 SRM and five-phase 10/8 SRM.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed, the machine is operating in motor mode, otherwise in generator mode. In the component icon, the positive terminals of the stator windings are marked with a dot.

> **Note**
>
> The Switched Reluctance Machine models can only be simulated with the Continuous State-Space Method.

The machine flux linkage is modeled as a 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.

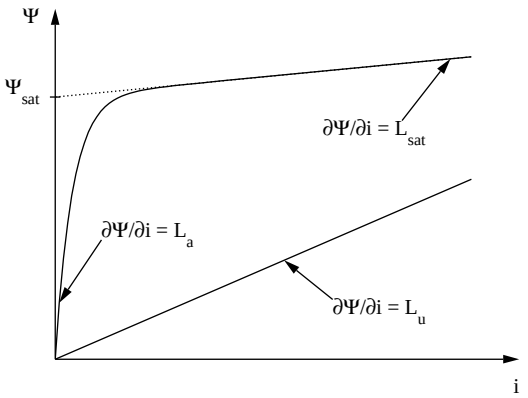


Fig. 16.103: Magnetization characteristics

In the unaligned rotor position the flux linkage is approximated as a linear function:

$$\Psi_{\mathrm{u}}(i) = L_{\mathrm{u}} \cdot i$$

In the aligned rotor position the flux linkage is a non-linear function of the stator current:

$$\Psi_{\mathrm{a}}(i) = \Psi_{\mathrm{sat}} \cdot (1 - e^{-K \cdot i}) + L_{\mathrm{sat}} \cdot i$$

where

$$K = \frac{L_{\mathrm{a}} - L_{\mathrm{sat}}}{\Psi_{\mathrm{sat}}}$$

For intermediate rotor positions the flux linkage is written as a weighted sum of these two extremes

$$\Psi(i, \theta) = \Psi_{\mathrm{u}}(i) + f(\theta) \cdot (\Psi_{\mathrm{a}}(i) - \Psi_{\mathrm{u}}(i))$$

using the weighting function

$$f(\theta) = \frac{1}{2} + \frac{1}{2} \cos\left(N_r \left[\theta + 2\pi \frac{x}{N_{\mathrm{s}}}\right]\right)$$

where N_r is the number of rotor poles, N_s is the number of stator poles, and $x = 0 \ldots (N_s/2 - 1)$ is the index of the stator phase.

Electrical System

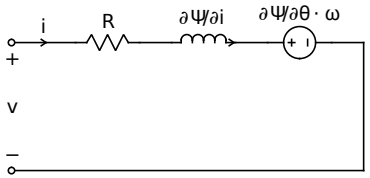


Fig. 16.104: SRM equivalent circuit schema

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 coenergy 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:

$$\frac{d}{dt}\omega = \frac{1}{J}(T_\mathrm{e} - F\omega - T_\mathrm{m})$$

Rotor angle:

$$\frac{d}{dt}\theta = \omega$$

Parameters

Stator resistance
Stator resistance R in ohms (Ω).

Unaligned stator inductance
Stator inductance L_u in the unaligned rotor position, in henries (H).

Initial aligned stator inductance
Initial stator inductance L_a in the aligned rotor position, in henries (H).

Saturated aligned stator inductance
Saturated stator inductance L_sat in the aligned rotor position, in henries (H).

Aligned saturation flux linkage
Flux linkage Φ_sat at which the stator saturates in the aligned position, in (Vs).

Inertia
Combined rotor and load inertia J in (Nms^2).

Friction coefficient
Viscous friction F in (Nms).

Initial rotor speed
Initial mechanical speed $\omega_{\mathrm{m},0}$ in radians per second ($\frac{\mathrm{rad}}{\mathrm{s}}$).

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 amperes (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 ω_m of the rotor in radians per second ($\frac{rad}{s}$).

Rotor position
The mechanical rotor angle θ_m in radians.

Electrical torque
The electrical torque T_e of the machine in newton-meters (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.

Switch Loss Calculator

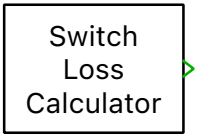
Purpose

Calculate the average sum of the switch losses of all probed components over the specified averaging period

Library

System

Description



This block provides a shortcut for calculating the summed average switch losses of one or more switch components. You can choose to calculate the *conduction losses*, the *switching losses* (optionally separated into *turn-on losses* and *turn-off losses*) and the *total losses* (comprising the conduction and switching losses). The output signal is a vector comprising the selected loss types.

Optionally, you may group the losses by switch type, i.e. calculate individual sums for the separate switch types by checking the box **Group by switch type**. If, for example, the list of components contains a number of diodes and MOSFETs and you have chosen to calculate conduction and switching losses, the output vector will be ordered as follows:

- Sum of diode conduction losses
- Sum of diode switching losses
- Sum of MOSFET conduction losses
- Sum of MOSFET switching losses

To add switch components to the list of components, select them in the schematic editor and drag them onto the list or the Switch Loss Calculator block. The block accepts all switch components that implement the thermal loss model (see “Supported Devices” (p. 176)).

When you drag a subsystem, masked or unmasked, onto the list of components or the Switch Loss Calculator block, PLECS descends into the subsystem and adds all supported switch components that the subsystem contains.

Operating Principle

The internal calculations of the Switch Loss Calculator are illustrated by the equivalent schematic in Fig. 16.105. The example uses two switch components, for which all loss signal types are calculated.

Parameters

Averaging time

A scalar specifying the period or a two-element vector specifying the period and offset of the averaging interval, in seconds (s). See also the **Discrete-Periodic** sample time type in section “Sample Times” (p. 43).

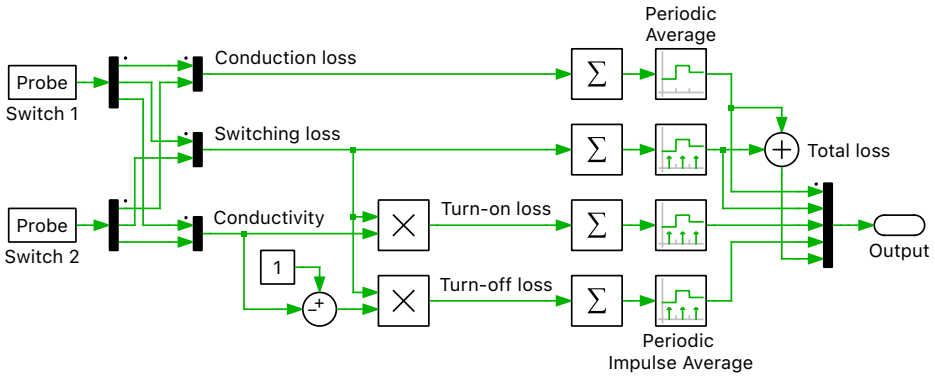


Fig. 16.105: Switch Loss Calculator equivalent schematic

Equivalent schematic of the Switch Loss Calculator block.

↪ *Added in PLECS 4.7.*

Symmetrical PWM

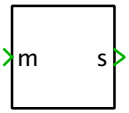
Purpose

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 (3ph) (p. 750) or the Ideal Converter (3ph) (p. 743). 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 next three figures illustrate Natural Sampling with carrier starting at minimum, center or maximum.

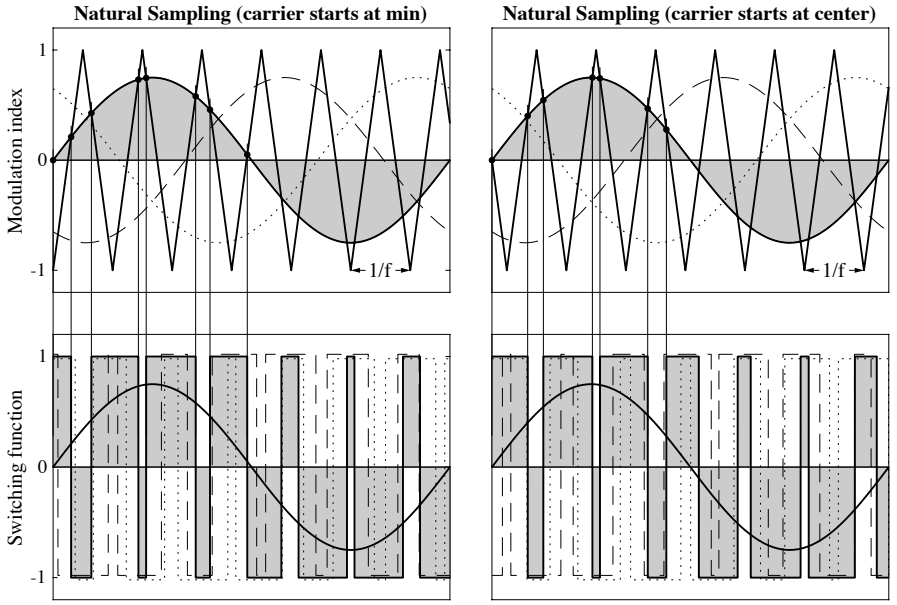


Fig. 16.106: Natural Sampling - carrier starting at minimum or center

The next two figures illustrate Regular Sampling with single update, where the modulation index is updated only at minimum or the maximum of the carrier.

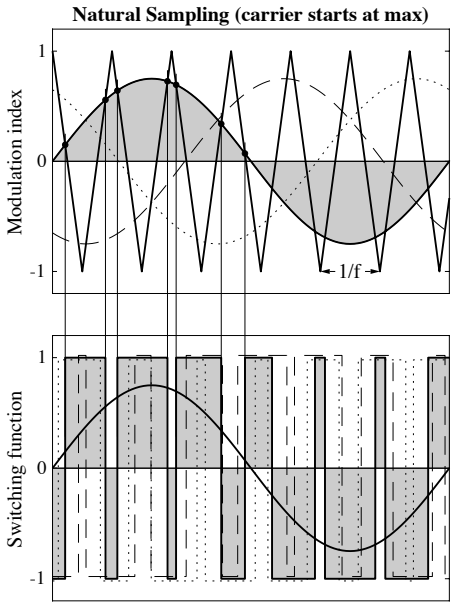


Fig. 16.107: Natural Sampling - carrier starting at maximum

In the last figure, Regular Sampling with double update is illustrated, i.e. the modulation index is updated at both tips of the triangular carrier.

Parameters

Sampling

Select a sampling method. If you select Natural Sampling, the carrier signal may begin at mininum, center, or maximum at simulation start. The Regular Sampling method lets you choose among single update at either min or max, and double update at both min and max.

↪ *Changed in PLECS 4.7:* Implemented additional sampling schemes.

Carrier frequency

The frequency f of the triangular carrier signal, in hertz (Hz).

Carrier phase shift

The time offset of the carrier signal, in per unit (p.u.) of the carrier period.

Carrier limits

The range of the triangular carrier. The default is `[-1 1]`.

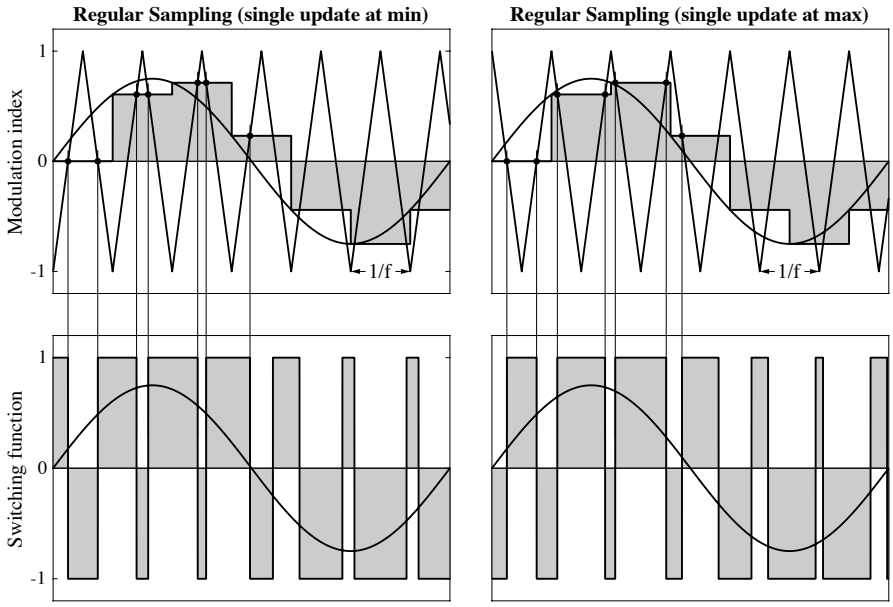


Fig. 16.108: Regular Sampling with single update

Output values

Values of the switching function in off-state and on-state. The default is `[-1 1]`.

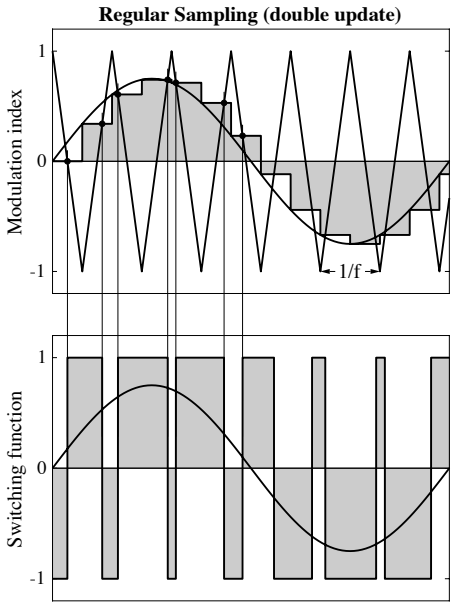


Fig. 16.109: Regular Sampling with double update

Symmetrical PWM (3-Level)

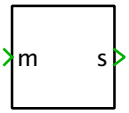
Purpose

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 IGBT 3-Level Converter (3ph) (p. 748) or the Ideal 3-Level Converter (3ph) (p. 742). In these cases the modulation index must have a width of 3 according to the number of inverter legs.

Fig. 16.110 illustrates 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.

Fig. 16.111 on p. 994 illustrates 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 hertz (Hz).

Carrier phase shift
The time offset of the carrier signal, in per unit $(\mathrm{p.u.})$ of the carrier period.

Carrier limits
The range of the triangular carriers. The default is `[-1 1]`.

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.

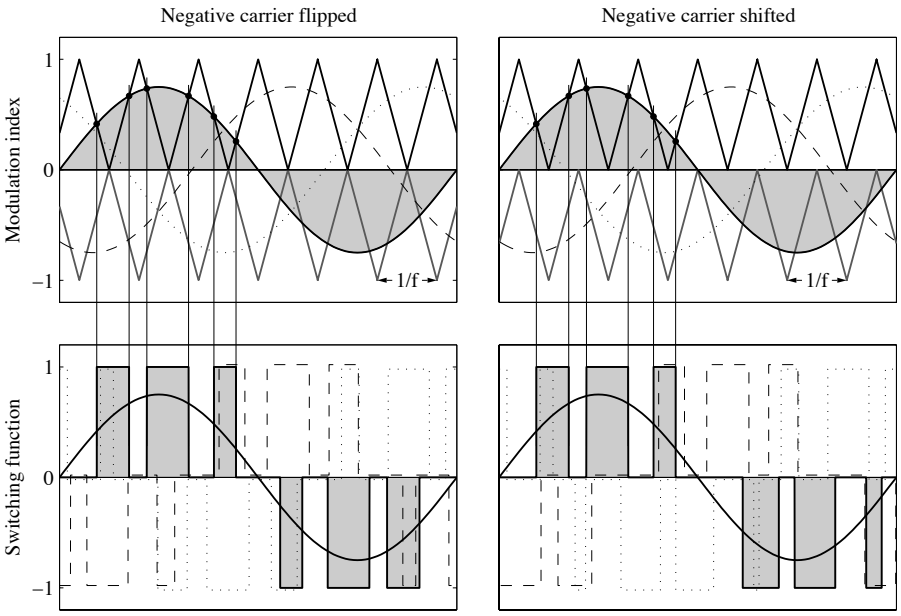


Fig. 16.110: Natural sampling for 3-level PWM

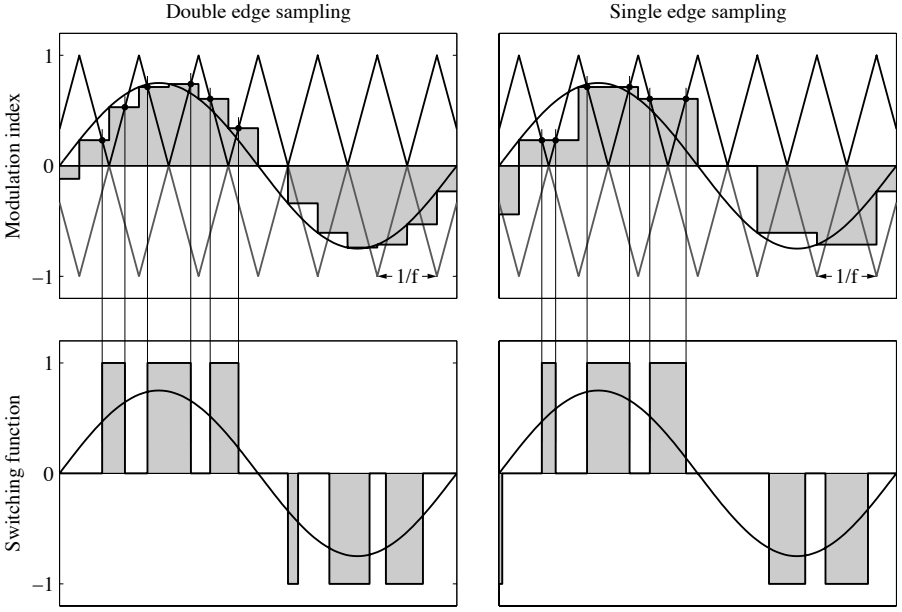


Fig. 16.111: Regular sampling for 3-level PWM

Synchronous Machine (Round Rotor)

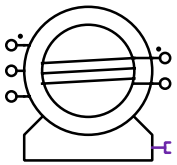
Purpose

Smooth air-gap synchronous machine with main-flux saturation

Library

Electrical / Machines

Description



This synchronous machine has one damper winding on the direct axis and two damper windings on the quadrature axis of the rotor. Main flux saturation is modeled by means of a continuous function.

The machine operates as a motor or generator; if the mechanical torque has the same sign as the rotational speed, the machine is operating in motor mode, otherwise in generator mode. All electrical variables and parameters are viewed from the stator side. In the component icon, phase a of the stator winding and the positive pole of the field winding are marked with a dot.

In order to inspect the implementation, please select the component in your circuit and choose **Look under mask** from the **Edit > Subsystem** menu or the block's context 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 equivalent electrical circuits for the d and q axes are shown in Fig. 16.112 on p. 999.

Stator flux linkages:

$$\Psi_{\mathrm{d}} = L_{\mathrm{ls}}\, i_{\mathrm{d}} + L_{\mathrm{m,d}}\,(i_{\mathrm{d}} + i'_{\mathrm{f}} + i'_{\mathrm{k,d}})$$

$$\Psi_{\mathrm{q}} = L_{\mathrm{ls}}\, i_{\mathrm{q}} + L_{\mathrm{m,q}}\,(i_{\mathrm{q}} + i'_{\mathrm{g}} + i'_{\mathrm{k,q}})$$

The machine model offers two different implementations of the electrical system: a traditional rotor reference frame and a voltage behind reactance formulation.

Rotor Reference Frame

Using Park's transformation, the 3-phase circuit equations in physical variables are transformed to the dq rotor reference frame. This results in constant coefficients in the stator and rotor equations making the model numerically efficient. However, interfacing the dq model with the external 3-phase network may be difficult. Since the coordinate transformations are based on voltage-controlled current sources, inductors and naturally commutated devices such as diode rectifiers may not be directly connected

to the stator terminals. In these cases, fictitious RC snubbers are required to create the necessary voltages across the terminals.

Voltage Behind Reactance

This formulation allows for direct interfacing of arbitrary external networks with the 3-phase stator terminals. The rotor dynamics are expressed using explicit state-variable equations while the stator branch equations are described in circuit form. However, due to the resulting time-varying inductance matrices, this implementation is numerically less efficient than the traditional rotor reference frame.

In both implementations, the value of the main flux inductance L_{m} is not constant but depends on the main flux linkage Ψ_{m} as illustrated in the $\Psi_{\mathrm{m}}/i_{\mathrm{m}}$ diagram. For flux linkages Ψ_{m} far below the transition flux Ψ_T, the relationship between flux and current is almost linear and determined by the unsaturated magnetizing inductance $L_{\mathrm{m},0}$. For large flux linkages the relationship is governed by the saturated magnetizing inductance $L_{\mathrm{m,sat}}$. Ψ_{T} defines the knee of the transition between unsaturated and saturated main flux inductance. The tightness of the transition is defined with the form factor f_{T}. If you do not have detailed information about the saturation characteristic of your machine, $f_{\mathrm{T}} = 1$ 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:

$$T_{\mathrm{e}} = \frac{3}{2}\, p(i_{\mathrm{q}}\, \Psi_d - i_{\mathrm{d}}\, \Psi_q)$$

Mechanical System

Mechanical rotor speed ω_{m}:

$$\dot{\omega}_{\mathrm{m}} = \frac{1}{J}(T_{\mathrm{e}} - F\omega_{\mathrm{m}} - T_{\mathrm{m}})$$

$$\dot{\omega}_{\mathrm{m}} = \omega_{\mathrm{m}}$$

Parameters

Most parameters for the Salient Pole Synchronous Machine (p. 1001) are also applicable to this round rotor machine. The following parameters are different:

Unsaturated magnetizing inductance

The unsaturated magnetizing inductance $L_{\mathrm{m},0}$. The value in henries (H) is referred to the stator side.

Saturated magnetizing inductance

The saturated magnetizing inductance $L_{\mathrm{m,sat}}$, in henries (H). If no saturation is to be modeled, set $L_{\mathrm{m,sat}} = L_{\mathrm{m},0}$.

Damper resistance

A three-element vector containing the damper winding resistance $R'_{\mathrm{k,d}}$, $R'_{\mathrm{k,q1}}$ and $R'_{\mathrm{k,q2}}$ of the d-axis and the q-axis. The values in ohms (Ω) are referred to the stator side.

Damper leakage inductance

A three-element vector containing the damper winding leakage inductance $L'_{\mathrm{lk,d}}$, $L'_{\mathrm{lk,q1}}$ and $L'_{\mathrm{lk,q2}}$ 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 $i_{\mathrm{f},0}$ in the field winding and $i'_{\mathrm{k,q1},0}$ 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 (p. 1001) are also available with this machine. Only the following probe signal is different:

Damper currents

The damper currents $i'_{\mathrm{k,d}}$, $i'_{\mathrm{k,q1}}$ and $i'_{\mathrm{k,q2}}$ in the stationary reference frame in amperes (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.

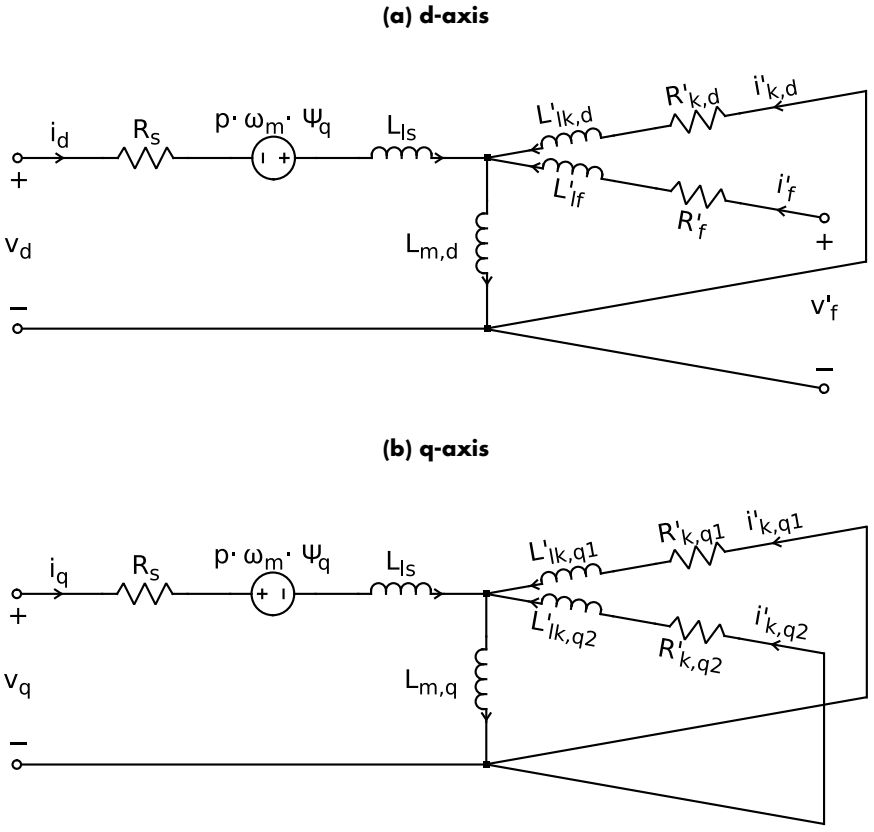


Fig. 16.112: Equivalent electrical circuits of the d and q axes of a round rotor synchronous machine.

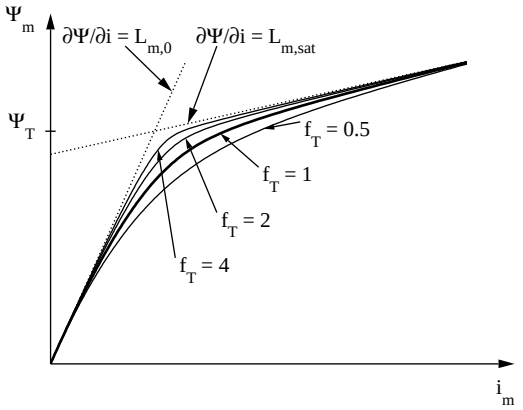


Fig. 16.113: Magnetic saturation characteristic

Synchronous Machine (Salient Pole)

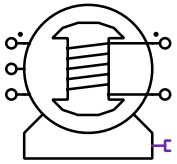
Purpose

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

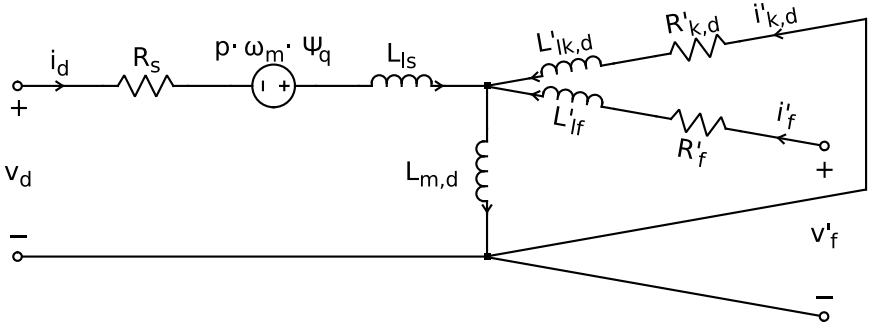


Fig. 16.114: d-axis

Stator flux linkages:

$$\Psi_{\mathrm{d}} = L_{\mathrm{ls}}\, i_{\mathrm{d}} + L_{\mathrm{m,d}}\,(i_{\mathrm{d}} + i'_{\mathrm{f}} + i'_{\mathrm{k,d}})$$

$$\Psi_{\mathrm{q}} = L_{\mathrm{ls}}\, i_{\mathrm{q}} + L_{\mathrm{m,q}}\,(i_{\mathrm{q}} + i'_{\mathrm{k,q}})$$

The machine model offers two different implementations of the electrical system: a traditional rotor reference frame and a voltage behind reactance formulation.

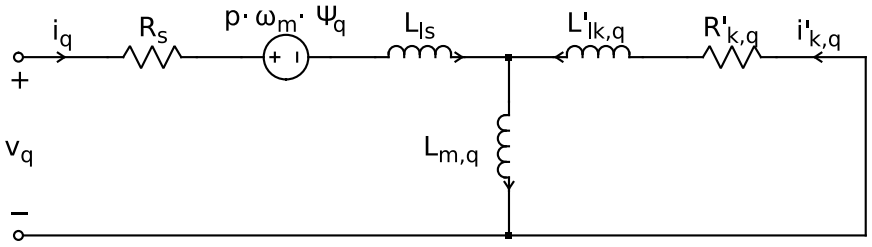


Fig. 16.115: q-axis

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 $L_{\mathrm{m,d}}$ and $L_{\mathrm{m,q}}$ are not constant but depend on the main flux linkage Ψ_{m} as illustrated in the $\Psi_{\mathrm{m}}/i_{\mathrm{m}}$ diagram.

In this machine model, the anisotropic factor

$$m = \sqrt{L_{\mathrm{m,q,0}}/L_{\mathrm{m,d,0}}} \equiv \sqrt{L_{\mathrm{m,q}}/L_{\mathrm{m,d}}} = \text{const.}$$

is assumed to be constant at all saturation levels. The equivalent magnetizing flux Ψ_{m} in an isotropic machine is defined as

$$\Psi_{\mathrm{m}} = \sqrt{\Psi_{\mathrm{m,d}}^2 + \Psi_{\mathrm{m,q}}^2/m^2}$$

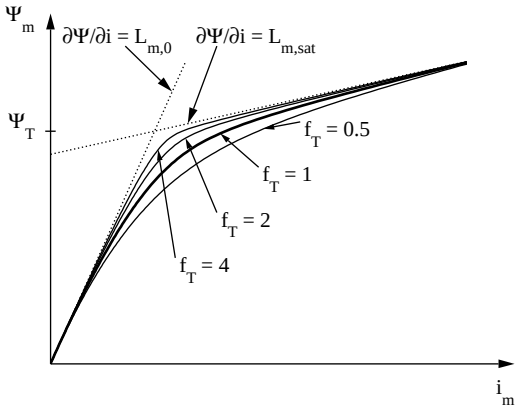


Fig. 16.116: Magnetic saturation characteristic

For flux linkages Ψ_{m} far below the transition flux Ψ_{T}, the relationship between flux and current is almost linear and determined by the unsaturated magnetizing inductance $L_{\mathrm{m},0}$. For large flux linkages the relationship is governed by the saturated magnetizing inductance $L_{\mathrm{m,sat}}$. Ψ_{T} defines the knee of the transition between unsaturated and saturated main flux inductance. The tightness of the transition is defined with the form factor f_{T}. If you do not have detailed information about the saturation characteristic of your machine, $f_{\mathrm{T}} = 1$ 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:

$$T_{\mathrm{e}} = \frac{3}{2}\, p(i_{\mathrm{q}}\, \Psi_{\mathrm{d}} - i_{\mathrm{d}}\, \Psi_{\mathrm{q}})$$

Mechanical System

Mechanical rotor speed ω_{m}:

$$\dot{\omega}_{\mathrm{m}} = \frac{1}{J}(T_{\mathrm{e}} - F\omega_{\mathrm{m}} - T_{\mathrm{m}})$$

$$\dot{\theta}_{\mathrm{m}} = \omega_{\mathrm{m}}$$

Parameters

Model
Implementation in the rotor reference frame or as a voltage behind reactance.

Stator resistance
Armature or stator winding resistance R_{s} in ohms (Ω).

Stator leakage inductance
Armature or stator leakage inductance L_{ls} in henries (H).

Unsaturated magnetizing inductance
A two-element vector containing the unsaturated stator magnetizing inductance $L_{\mathrm{m,d,0}}$ and $L_{\mathrm{m,q,0}}$ 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 $L_{\mathrm{m,d,sat}}$ along the d-axis, in henries (H). If no saturation is to be modeled, set $L_{\mathrm{m,d,sat}} = L_{\mathrm{m,d,0}}$.

Magnetizing flux at saturation transition
Transition flux linkage Ψ_{T}, in (Vs), defining the knee between unsaturated and saturated main flux inductance.

Tightness of saturation transition
Form factor f_{T} defining the tightness of the transition between unsaturated and saturated main flux inductance. The default is `1`.

Field resistance
d-axis field winding resistance R'_{f} in ohms (Ω), referred to the stator side.

Field leakage inductance
d-axis field winding leakage inductance L'_{lf} in henries (H), referred to the stator side.

Damper resistance
A two-element vector containing the damper winding resistance $R'_{\mathrm{k,d}}$ and $R'_{\mathrm{k,q}}$ of the d-axis and the q-axis. The values in ohms (Ω) are referred to the stator side.

Damper leakage inductance
A two-element vector containing the damper winding leakage inductance $L'_{\mathrm{lk,d}}$ and $L'_{\mathrm{lk,q}}$ of the d-axis and the q-axis. The values in henries (H) are referred to the stator side.

Inertia
Combined rotor and load inertia J in (Nms^2).

Friction coefficient
Viscous friction F in (Nms).

Number of pole pairs
Number of pole pairs p.

Initial rotor speed
Initial mechanical speed $\omega_{\mathrm{m,0}}$ in radians per second $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Initial rotor position
Initial mechanical rotor angle $\theta_{\mathrm{m,0}}$ in radians. If $\theta_{\mathrm{m,0}}$ 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 $i_{\mathrm{a,0}}$ and $i_{\mathrm{b,0}}$ of phase a and b in amperes (A).

Initial field current
Initial current $i_{\mathrm{f,0}}$ in the field winding in amperes (A).

Initial stator flux
A two-element vector containing the initial stator flux $\Psi'_{\mathrm{d,0}}$ and $\Psi'_{q,0}$ in the rotor reference frame in weber (Vs).

Probe Signals

Stator phase currents
The three-phase stator winding currents i_{a}, i_{b} and i_{c}, in amperes (A). Currents flowing into the machine are considered positive.

Field currents
The excitation current i_{f} in amperes (A).

Damper currents
The damper currents $i'_{\mathrm{k,d}}$ and $i'_{\mathrm{k,q}}$ in the stationary reference frame, in amperes (A).

Stator flux (dq)
The stator flux linkages Ψ_{d} and Ψ_{q} in the stationary reference frame in (Vs).

Magnetizing flux (dq)
The magnetizing flux linkages $\Psi_{\mathrm{m,d}}$ and $\Psi_{\mathrm{m,q}}$ in the stationary reference frame in (Vs).

Rotational speed
The rotational speed ω_{m} of the rotor in radians per second $(\frac{\mathrm{rad}}{\mathrm{s}})$.

Rotor position
The mechanical rotor angle θ_{m} in radians.

Electrical torque
The electrical torque T_{e} of the machine in newton-meters (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

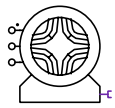
Purpose

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.

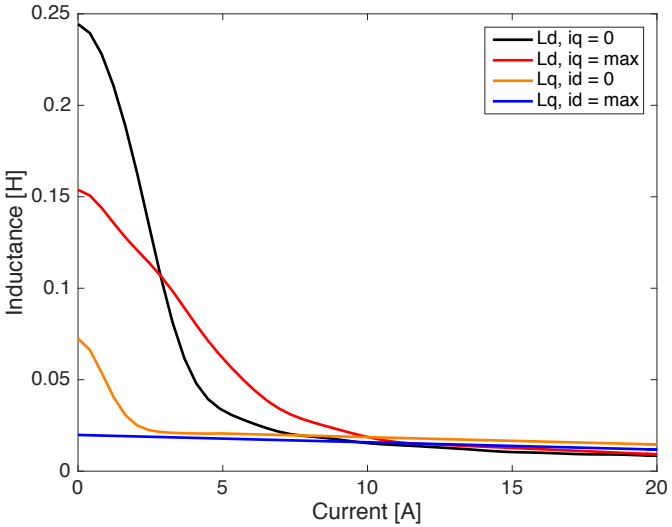


Fig. 16.117: Inductance vs current characteristics for synchronous reluctance machine

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.

Electical System

The model utilizes the Non-Excited Synchronous Machine component (p. 850). 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:

$$T_\mathrm{e} = \frac{3}{2}\, p\, (\varphi_\mathrm{d}\, i_\mathrm{q} - \varphi_\mathrm{q}\, i_\mathrm{d})$$

Mechanical System

Mechanical rotor speed ω_m:

$$\dot{\omega}_\mathrm{m} = \frac{1}{J}(T_\mathrm{e} - F\omega_\mathrm{m} - T_\mathrm{m})$$

$$\dot{\theta}_\mathrm{m} = \omega_\mathrm{m}$$

Parameters

General

Stator resistance
Armature or stator resistance R_s in Ω.

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 $i_{\mathrm{a},0}$ and $i_{\mathrm{b},0}$ of phase a and b in amperes (A). $i_{\mathrm{c},0}$ 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. $[0 \dots i_{\mathrm{d,max}}]$ and $[0 \dots i_{\mathrm{q,max}}]$. The values are in amperes (A).

Ld (iq = 0) lookup vector

d-axis inductance when there is no cross saturation $(i_{\mathrm{q}} = 0)$. 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 $(i_{\mathrm{q}} = i_{\mathrm{q,max}})$. 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 $(i_{\mathrm{d}} = 0)$. 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 $(i_{\mathrm{d}} = i_{\mathrm{d,max}})$. 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 component (p. 850).

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 (ΔW) between zero and maximum cross saturation is calculated for both the d-axis (ΔW_{d}) and q-axis (ΔW_{q}). Configure to check if $\Delta W_{\mathrm{d,q}}$ 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 (Nms^2).

Friction coefficient
Viscous friction F in (Nms).

Initial rotor speed
Initial mechanical rotor speed $\omega_{\mathrm{m},0}$ in radians per second ($\frac{\mathrm{rad}}{\mathrm{s}}$).

Initial rotor position
Initial mechanical rotor angle $\theta_{\mathrm{m},0}$ in radians.

Probe Signals

All probe signals for the Non-Excited Synchronous Machine (p. 850) 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.

↪ *Added in PLECS 4.0.*

Task Frame

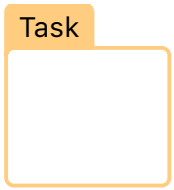
Purpose

Associate the enclosed components with a task in a multi-tasking environment

Library

System

Description



The Task Frame is used to configure a model for execution in a multi-tasking environment. To use a Task Frame, you must first configure a set of tasks on the **Scheduling** tab of the **Coder Options** dialog (see “Scheduling” (p. 351)). Once you have done this, you can double click on the label to open the parameter dialog and choose the desired task from the list.

The execution of all components inside the frame will be scheduled in the selected task. Multiple Task Frames can reference the same task; in this case all components inside these frames will be scheduled in the same task.

If the frame contains a Subsystem (p. 977), the task will also be assigned to the components in the sub-schematic; however, you can override this mechanism by placing Task Frames in the sub-schematic.

To move a Task Frame, press the left mouse button on the label or on an edge and drag the frame to the desired location. To change the size of a Task Frame, select it, then drag one of the selection handles.

↪ *Added in PLECS 4.4.*

Task Transition

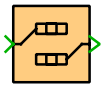
Purpose

Transfer data between tasks using a double buffer.

Library

System

Description



The Task Transition block is used to influence the execution of a model in a multi-tasking environment (see “Task Transitions in Multi-Tasking Mode” (p. 353)).

The schematic in Fig. 16.118 shows a block in a fast task that receives an input from a block in a slow task. If the sample time of the slow task is an integer multiple of the sample time of the fast task, PLECS silently inserts a Delay that operates with the sample time of the slow task in order to ensure that the data is exchanged in a deterministic manner. This introduces a maximum latency of one sample period of the slow task.

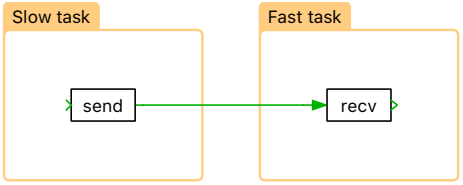


Fig. 16.118: Default task transition with deterministic delay.

If you wish the block in the fast task to receive the data as soon as possible with minimum (but non-deterministic) latency, you need to insert a Task Transition block in front of the receiving block as shown in Fig. 16.119.

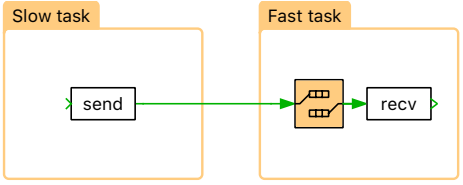


Fig. 16.119: Task transition with minimum latency.

↪ *Added in PLECS 4.6.*

Thermal Capacitor

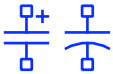
Purpose

Thermal capacitance of piece of material

Library

Thermal / Components

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” (p. 60) for information on how to change the graphical representation of thermal capacitors.

Parameters

Capacitance

The value of the capacitor, in $(\mathrm{J/K})$. 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 degrees Celsius $(^{\circ}\mathrm{C})$. If left blank or if the value is `nan`, PLECS will initialize the value based on a thermal `DC` analysis, see “Temperature Initialization” (p. 177).

Probe Signals

Temperature

The temperature difference measured across the capacitance, in degrees Celsius $(^{\circ}\mathrm{C})$. A positive value is measured when the temperature at the terminal marked with “+” is greater than the temperature at the unmarked terminal.

Thermal Chain

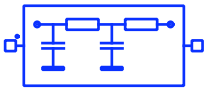
Purpose

Thermal impedance implemented as RC chain

Library

Thermal / Components

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 in Fig. 16.120. The thermal capacitor C1 is connected to the terminal marked with a dot.

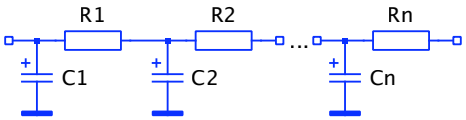


Fig. 16.120: Thermal chain RC network schematic

Parameters

Thermal resistances

A vector containing the values of the thermal resistors R1 . . . Rn, in (K/W).

Thermal capacitances

A vector containing the values of the thermal capacitors C1 . . . Cn, in (J/K).

Initial temperature

A scalar value specifying the initial temperature of all thermal capacitors at simulation start, in degrees Celsius (°C). If left blank or if the value is `nan`, PLECS will initialize the value based on a thermal "DC" analysis, see "Temperature Initialization" (p. 177).

Thermal Ground

Purpose Connect to common reference temperature

Library Thermal / Connectivity

Description The Thermal Ground implements a connection to the thermal reference.



Thermal Model Settings

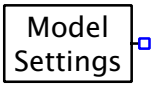
Purpose

Configure settings for an individual thermal model.

Library

Thermal / Model Settings

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” (p. 342).

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.

↪ *Added in PLECS 4.0.*

Thermal Multiplexer

Purpose

Combine several thermal connections into single vector

Library

Thermal / Connectivity

Description



This multiplexer combines several thermal connections into one vector connection. The individual connections may themselves be vectors. In the block icon, the first individual connection is marked with a dot.

Parameters

Width

This parameter allows you to specify the number and/or width of the individual connections. You can choose between the following formats for this parameter:

Scalar: A scalar specifies the number of individual connections each having a width of 1.

Vector: The length of the vector determines the number of individual connections. Each element specifies the width of the corresponding individual connections.

↪ *Added in PLECS 4.6.*

Thermal Package Impedance

Purpose

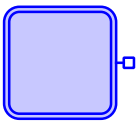
Model thermal coupling in a semiconductor package

Library

Thermal / Components

Description

The Thermal Package Impedance is used in conjunction with a Thermal Package Description to model the thermal coupling between multiple semiconductor devices and the case of a semiconductor module.



For additional information see section “Thermal Package Description” (p. 195).

Parameters

Number of terminals

This parameter allows you to change the number of external thermal connectors of the component.

Package impedance

This parameter defines the state-space model of the thermal network between the individual semiconductor devices and the case of a module.

Initial temperature

The initial temperature difference between the internal states of the impedance and the thermal reference at simulation start, in degrees Celsius (°C). If left blank or if the value is `nan`, PLECS will initialize the value based on a thermal “DC” analysis, see “Temperature Initialization” (p. 177).

↪ *Added in PLECS 4.6.*

Thermal Port

Purpose

Add thermal connector to subsystem

Library

Thermal / Connectivity

Description



Thermal ports are used to establish thermal connections between a schematic and the subschematic of a Subsystem (p. 977). 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.

A terminal can be moved around the edges of the subsystem by dragging it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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 a 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 (p. 557) blocks.

Parameters

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 / Components

Description



This component provides an ideal one-dimensional thermal resistor between its two thermal ports. See section “Configuring PLECS” (p. 60) for information on how to change the graphical representation of thermal resistors.

Parameters

Thermal resistance

The resistance in $(\mathrm{K/W})$. All positive and negative values are accepted, including `0` and `inf` (∞). The default is `1`.

Thermal Selector

Purpose

Select or reorder elements from vector connection

Library

Thermal / Connectivity

Description

The Thermal Selector block connects the individual elements of the output connection to the specified elements of the input connection. The input connection is marked with a dot.



Parameters

Input width

The width of the input connection.

Output indices

A vector with the indices of the input elements that the output connection should contain.

↪ *Added in PLECS 4.6.*

Thermometer

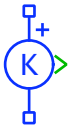
Purpose

Output measured temperature as signal

Library

Thermal / Meters

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 Signal Outport block (p. 952) or by dragging the component into the dialog box of a Probe block.

Probe Signals

Measured temperature

The measured temperature difference in degrees Celsius (°C).

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 (Ω). The default is `0`.

Initial conductivity

Initial conduction state of the thyristor. The thyristor is initially blocking if the parameter evaluates to zero, otherwise it is conducting.

Thermal description}

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (p. 171). If no thermal description is given, the losses are calculated based on the voltage drop $v_\mathrm{on} = V_\mathrm{f} + R_\mathrm{on} \cdot i$.

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink, in (K/W). The default is `0`.

↪ *Added in PLECS 4.6.*

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” (p. 177).

Switch model (CPU code generation)
Select the switch model in the generated code.

Fundamental grid frequency
Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is 0.

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 Pair

Purpose

Thyristor rectifier/inverter with a Sub-cycle average (p. 489) implementation

Library

Electrical / Power Modules

Description



This Power Module implements a sub-cycle averaged (p. 489) model of a series-connected thyristor pair with a single internal switching element and controlled current and voltage sources. One thyristor at most may conduct at a given time. The DC side of the rectifier/inverter has current source behavior. The output terminal is typically connected to an inductor.

The single-switch implementation has fewer conduction states compared to two Thyristor components (p. 1024). This reduces the number of switching combinations when generating code from a simulation model, as described in “Code Generation for Physical Systems” (p. 342).

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver.

> **Note**
>
> The thyristor pair cannot model a shoot-through, i.e. the situation where both thyristors conduct at the same time.

Parameters

Switch model (CPU code generation)
: This parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Fundamental grid frequency
: Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is 0.

Thyristor Rectifier/Inverter

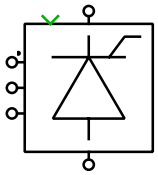
Purpose

3-phase thyristor rectifier/inverter

Library

Electrical / Converters

Description



Implements a three-phase rectifier or inverter based on the Thyristor model (p. 1024). 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. The rectifier is shown on the left side of Fig. 16.121, the inverter on the right.

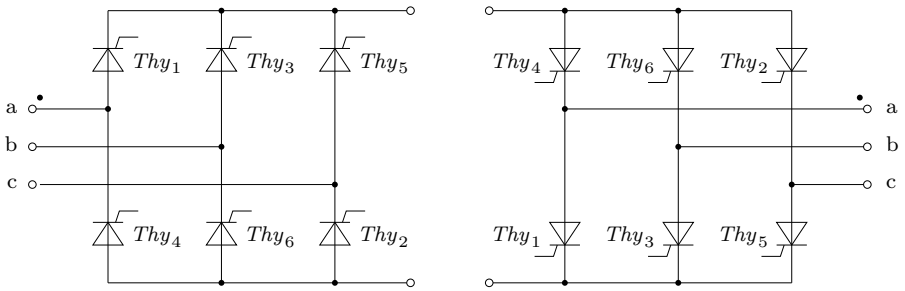


Fig. 16.121: Thyristor rectifier (left) and inverter (right) circuit diagram

Parameters

For a description of the parameters see the documentation of the Thyristor (p. 1024).

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 (p. 646) 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 (Ω). The default is `0`.

Off-resistance
The resistance R_{off} of the blocking device, in ohms (Ω). 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} in amperes (A) under test conditions.

Current slope at turn-off
The turn-off current slope dI_{r}/dt in $(\frac{\mathrm{A}}{\mathrm{s}})$ under test conditions.

Reverse recovery time
The turn-off time t_{rr} in seconds (s) under test conditions.

Peak recovery current
The absolute peak value of the reverse current I_{rrm} in amperes (A) under test conditions.

Reverse recovery charge
The reverse recovery charge Q_{rr} in coulombs (C) 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, in henries (H). The default is `10e-10`.

Probe Signals

Thyristor voltage
The voltage measured between anode and cathode.

Thyristor current
The current through the thyristor flowing from anode to cathode.

Thyristor conductivity
Conduction state of the internal switch. The signal outputs 0 when the thyristor is blocking, and 1 when it is conducting.

References

- A. Courtay, “MAST power diode and thyristor models including automatic parameter extraction”, SABER User Group Meeting Brighton, UK, Sept. 1995.

To File

Purpose

Write time stamps and signal values to a file.

Library

System

Description



While a simulation is running, the To File block writes the time stamps and the values of its input signals to a file. The file format can be either a text file with comma separated values (csv) or a MATLAB data file (mat). CSV files can be imported by all common spreadsheet tools like Microsoft Excel.

In a csv file a new row is appended for each time step. When writing to a MATLAB file the resulting data contains a column for each time step.

The first value for each data record is the simulation time of the current simulation step. The value is followed by the signal values of the input signal.

Parameters

Filename

The name of the data file to write to. Files will be stored relative to the model directory unless an absolute file path is given. The data file will be created if it doesn't exist. An existing file of the same name will be overwritten.

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).

Write signal names

Controls whether the signal names are written to the file as the first row before the data records. This parameter is shown only if the **File type** parameter is set to `csv`.

↪ *Added in PLECS 4.4.*

Sample time

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). For positive values of the sampling period, the input data will be written to the data file once in the given simulation interval. If the sampling period 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" (p. 43).

Torque (Constant)

Purpose

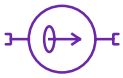
Generate constant torque

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 $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

Torque (Controlled)

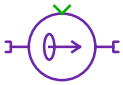
Purpose

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`.

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 $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$.

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.

Parameters

Second flange
Controls whether the second flange is accessible or connected to the rotational reference frame.

Probe Signals

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:

$$\tau = -c \cdot \Delta\phi$$
$$\Delta\theta = \theta - \theta_0$$
$$\frac{d}{dt}\theta = \omega$$

where τ is the torque flow from the unmarked towards the marked flange, θ is the angle of the marked flange with respect to the unmarked one, and θ_0 is the equilibrium flange displacement.

> **Note**
>
> A 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 $(\frac{\mathrm{Nm}}{\mathrm{rad}})$.

Equilibrium (unstretched) displacement
The displacement θ_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 τ, in newton-meters (Nm).

Deformation
The spring deformation $\Delta\theta$, in radians.

Total Harmonic Distortion

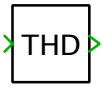
Purpose

Calculate total harmonic distortion of periodic signal

Library

Control / Measurements

Description



This block calculates the total harmonic distortion of a periodic input signal. The sample time and the fundamental frequency can be specified.

The THD is defined as:

$$\mathrm{THD} = \sqrt{\frac{\sum_{\nu \geq 2} U_\nu^2}{U_1^2}} = \sqrt{\frac{U_{\mathrm{rms}}^2 - U_0^2 - U_1^2}{U_1^2}}$$

where U_ν is the RMS value of the ν-th harmonic of the input signal and U_{rms} is its overall RMS value.

Example Model

- See the example model **"Total Harmonic Distortion"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Parameters

Sample time

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s). See also the **Discrete-Periodic** sample time type in section "Sample Times" (p. 43). If a sample time of `0` is specified, a continuous implementation based on the Moving Average block (p. 837) is active. The **Discrete-Periodic** implementation can potentially force a variable-step solver to take small integration steps. This may slow down the simulation. The default value of this parameter is `0`.

Fundamental frequency

The fundamental frequency of the periodic input signal in hertz (Hz).

↪ *Added in PLECS 4.4.*

Transfer Function

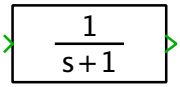
Purpose

Model linear time-invariant system using transfer function

Library

Control / Continuous

Description



The Transfer Function models a linear time-invariant system that is expressed in the Laplace domain in terms of the argument s:

$$\frac{Y(s)}{U(s)} = \frac{n_n s^n + \cdots + n_1 s + n_0}{d_n s^n + \cdots + d_1 s + d_0}$$

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.

The Transfer Function block performs scalar expansion: If the input signal is a vector, the transfer function is applied to each element of the input vector individually, and the output signal is a concatenated vector of all system outputs.

Parameters

Numerator coefficients

A vector of the s term coefficients $[n_n \ldots n_1, n_0]$ 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 Transfer Function supports multiple outputs for a single input by entering a matrix for the numerator. Each row of the matrix defines the numerator coefficients of an output.

Denominator coefficients

A vector of the s term coefficients $[d_n \ldots d_1, d_0]$ 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 $[x_n \ldots x_1, x_0]$.

The initial condition must be specified for the controller normal form, depicted below for the transfer function

$$\frac{Y(s)}{U(s)} = \frac{n_2 s^2 + n_1 s + n_0}{d_2 s^2 + d_1 s + d_0} = b_2 \left(a_2 + \frac{a_1 s + a_0}{s^2 + b_1 s + b_0} \right)$$

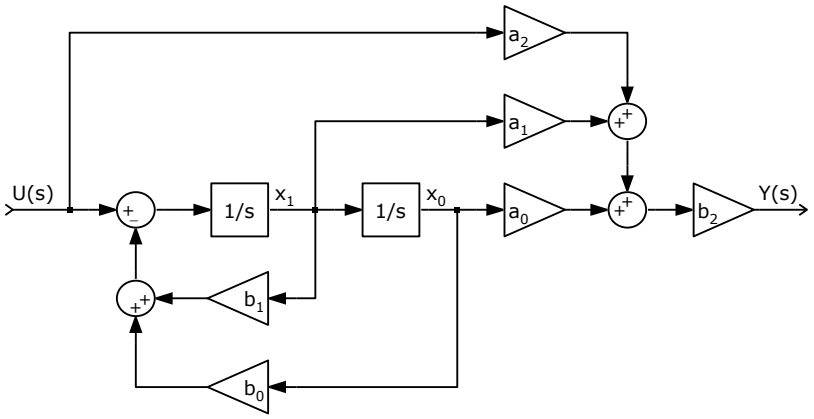


Fig. 16.122: Transfer function controller normal form

where

$$\begin{aligned} b_i &= \frac{d_i}{d_n} && \text{for } i < n \\ b_n &= \frac{1}{d_n} \\ a_i &= n_i - \frac{n_n d_i}{d_n} && \text{for } i < n \\ a_n &= n_n \end{aligned}$$

For the normalized transfer function (with $n_n = 0$ and $d_n = 1$) this simplifies to $b_i = d_i$ and $a_i = n_i$.

Note

The number of internal states is defined by the highest power of s of the denominator.

> **Note**
>
> A scalar initial condition will be applied to all internal states.

> **Note**
>
> In case of scalar expansion (multiple input signals), the initial condition can also be a matrix, where each row defines the initial condition for the individual inputs.

Probe Signals

Input
The input signal.

Output
The output signal.

State
The internal states of the controller normal form.

Transformation 3ph->RRF

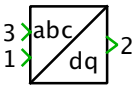
Purpose

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 $[x_d \ x_q]$ in a rotating reference frame. The first input is the three-phase signal. The second input is the rotation angle θ of the rotating reference frame. θ is given in radians.

$$\begin{bmatrix} x_d \\ x_q \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos\theta & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin\theta & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \cdot \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

Any zero-sequence component in the three-phase signals is discarded.

> **Note**
>
> The rotating reference frame transfomation follows the axis alignment and rotation direction conventions presented in “Conventions & Definitions” (p. 52).

Transformation 3ph->SRF

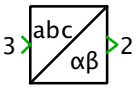
Purpose

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 $[x_\alpha \ x_\beta]$ in the stationary reference frame:

$$\begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix}$$

Any zero-sequence component in the three-phase signals is discarded.

> **Note**
>
> The stationary reference frame transfomation follows the axis alignment and rotation direction conventions presented in “Conventions & Definitions” (p. 52).

Transformation RRF->3ph

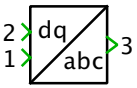
Purpose

Transform vector in rotating reference frame into 3-phase signal

Library

Control / Transformations

Description



This block transforms a two-dimensional vector $[x_\mathrm{d}\ x_\mathrm{q}]$ in a rotating reference frame into a three-phase signal $[x_\mathrm{a}\ x_\mathrm{b}\ x_\mathrm{c}]$. The first input of the block is the vector $[x_\mathrm{d}\ x_\mathrm{q}]$. The second input is the rotation angle θ of the rotating reference frame. θ is given in radians.

$$\begin{bmatrix} x_\mathrm{a} \\ x_\mathrm{b} \\ x_\mathrm{c} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \cos\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta - \frac{2\pi}{3}\right) \\ \cos\left(\theta + \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \cdot \begin{bmatrix} x_\mathrm{d} \\ x_\mathrm{q} \end{bmatrix}$$

The resulting three-phase signal does not have any zero-sequence component.

> **Note**
>
> The inverse rotating reference frame transfomation follows the axis alignment and rotation direction conventions presented in “Conventions & Definitions” (p. 52).

Transformation RRF->SRF

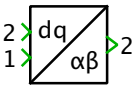
Purpose

Transform vector from rotating to stationary reference frame

Library

Control / Transformations

Description



This block transforms a two-dimensional vector $[x_\mathrm{d}\ \ x_\mathrm{q}]$ from a rotating reference frame into a vector $[x_\alpha\ \ x_\beta]$ in the stationary reference frame. The first input of the block is the vector $[x_\mathrm{d}\ \ x_\mathrm{q}]$. The second input is the angle θ between the rotating and the stationary frame. θ is given in radians.

$$\begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x_\mathrm{d} \\ x_\mathrm{q} \end{bmatrix}$$

Note

This transfomation follows the axis alignment and rotation direction conventions presented in “Conventions & Definitions” (p. 52).

Transformation SRF->3ph

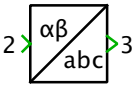
Purpose

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 $[x_\mathrm{a} \; x_\mathrm{b} \; x_\mathrm{c}]$.

$$\begin{bmatrix} x_\mathrm{a} \\ x_\mathrm{b} \\ x_\mathrm{c} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}$$

The resulting three-phase signal does not have any zero-sequence component.

> **Note**
>
> The inverse stationary reference frame transfomation follows the axis alignment and rotation direction conventions presented in “Conventions & Definitions” (p. 52).

Transformation SRF->RRF

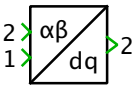
Purpose

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 $[x_\mathrm{d} \; x_\mathrm{q}]$ in a rotating reference frame. The first input is the vector $[x_\alpha \; x_\beta]$. The second input is the angle θ between the rotating and the stationary frame. θ is given in radians.

$$\begin{bmatrix} x_\mathrm{d} \\ x_\mathrm{q} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}$$

> **Note**
>
> This transfomation follows the axis alignment and rotation direction conventions presented in “Conventions & Definitions” (p. 52).

Transformers (3ph, 2 Windings)

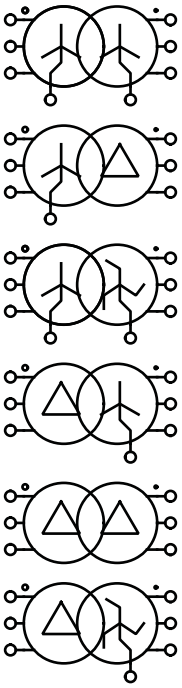
Purpose

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.

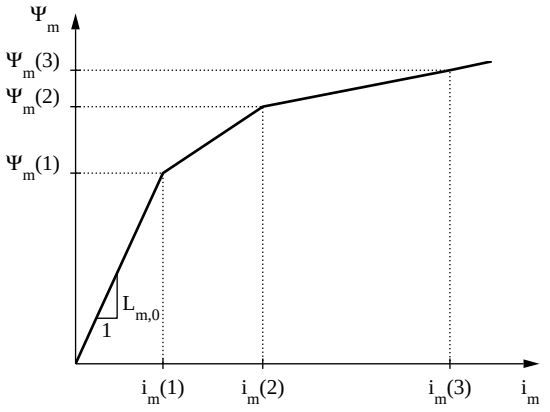


Fig. 16.123: Transformer core saturation characteristic

The core saturation characteristic of the transformer legs (Fig. 16.123) is piecewise linear and is modeled using the Saturable Inductor (p. 931). The magnetizing current i_m and flux Ψ_m value pairs are referred to the primary side. To

model a transformer without saturation enter 1 as the magnetizing current values and the desired magnetizing inductance L_{m} as the flux values. A stiff Simulink solver is recommended if the iron losses are not negligible, i.e. R_{fe} is not infinite.

Parameters

Leakage inductance

A two-element vector containing the leakage inductance of the primary side L_1 and the secondary side L_2. The inductivity is given in henries (H).

Winding resistance

A two-element vector containing the resistance of the primary winding R_1 and the secondary winding R_2, in ohms (Ω).

No. of turns

A two-element vector containing the number of turns of the primary winding n_1 and the secondary winding n_2.

Magnetizing current values

A vector of positive current values in amperes (A) defining the piece-wise linear saturation characteristic of the transformer legs. The current values must be positive and strictly monotonic increasing. At least one value is required.

Magnetizing flux values

A vector of positive flux values in (Vs) defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Core loss resistance

An equivalent resistance R_{fe} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

No. of core legs

The number of legs of the transformer core. This value may either be 3 or 5. In a three phase transformer with 3 legs the sum of the fluxes in the three phases must add up to zero. This constraint is modeled with auxiliary windings on each core, which are connected in series. A 5-leg transformer on the other hand is similar to three uncoupled single-phase transformers. Therefore, the constraint does not apply and the auxiliary windings are deactivated.

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 $i_{1,\mathrm{a}}$, $i_{1,\mathrm{b}}$ and, if the winding has a neutral point, $i_{1,\mathrm{c}}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is `[0 0]`, or `[0 0 0]` in the presence of a neutral point.

Initial currents wdg. 2

A vector containing the initial currents on the secondary side $i_{2,\mathrm{a}}$, $i_{2,\mathrm{b}}$ and, if the winding has a neutral point, $i_{2,\mathrm{c}}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is `[0 0]`, or `[0 0 0]` in the presence of a neutral point.

Transformers (3ph, 3 Windings)

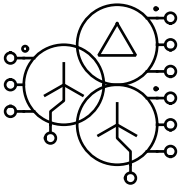
Purpose

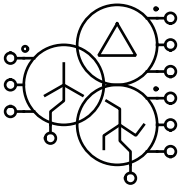
Three-phase transformers in Ydy and Ydz connection.

Library

Electrical / Transformers

Description





This group of components implements three-winding, three-phase transformers with a three-leg or five-leg core. The transformer core is assumed symmetrical, i.e. all phases have the same parameters. The primary winding is in star connection with an accessible neutral point and the secondary winding is in delta connection. Depending on the chosen component, the tertiary winding is wired either in star (y) or zig-zag (z) connection.

The phase angle difference between the primary and the secondary side must be an odd integer multiple of $30\,°$. If the tertiary winding is in star connection, the phase lag against the primary side must be an integer multiple of $60\,°$. If it is in zig-zag connection, the phase lag can be chosen arbitrarily. The windings of the secondary and tertiary side are allocated to the transformer legs according to the phase lags. Please note that the phase-to-phase voltage of delta windings is by a factor of $1/\sqrt{3}$ lower than the voltage of star or delta windings if the number of turns are equal.

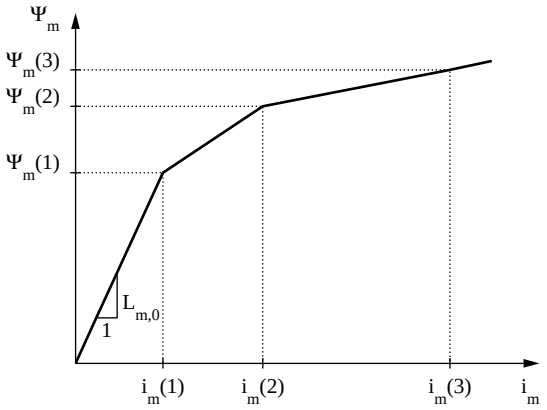


Fig. 16.124: Magnetizing inductance saturation characteristic

The core saturation characteristic of the transformer legs is piece-wise linear and is modeled using the Saturable Inductor (p. 931). The magnetizing current

i_{m} and flux Ψ_{m} value pairs are referred to the primary side. To model a transformer without saturation enter 1 as the magnetizing current values and the desired magnetizing inductance L_{m} as the flux values. A stiff Simulink solver is recommended if the iron losses are not negligible, i.e. R_{fe} is not infinite.

Parameters

Leakage inductance

A three-element vector containing the leakage inductance of the primary side L_1, the secondary side L_2 and the tertiary side L_3. The inductivity is given in henries (H).

Winding resistance

A three-element vector containing the resistance of the primary winding R_1, the secondary winding R_2 and the tertiary winding R_3, in ohms (Ω).

No. of turns

A three-element vector containing the number of turns of the primary winding n_1, the secondary winding n_2 and the tertiary winding n_3.

Magnetizing current values

A vector of positive current values in amperes (A) defining the piece-wise linear saturation characteristic of the transformer legs. The current values must be positive and strictly monotonic increasing. At least one value is required.

Magnetizing flux values

A vector of positive flux values in (Vs) defining the piece-wise linear saturation characteristic. The flux values must be positive and strictly monotonic increasing. The number of flux values must match the number of current values.

Core loss resistance

An equivalent resistance R_{fe} representing the iron losses in the transformer core. The value in ohms (Ω) is referred to the primary side.

No. of core legs

The number of legs of the transformer core. This value may either be 3 or 5. In a three phase transformer with 3 legs the sum of the fluxes in the three phases must add up to zero. This constraint is modeled with auxiliary windings on each core, which are connected in series. A 5-leg transformer on the other hand is similar to three uncoupled single-phase transformers. Therefore, the constraint does not apply and the auxiliary windings are deactivated.

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 $i_{1,\mathrm{a}}$, $i_{1,\mathrm{b}}$ and, if the winding has a neutral point, $i_{1,\mathrm{c}}$. 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 $i_{2,\mathrm{a}}$ and $i_{2,\mathrm{b}}$. The currents are given in amperes (A) and considered positive if flowing into the transformer. The default is `[0 0]`.

Initial currents wdg. 3

A vector containing the initial currents on the tertiary side $i_{3,\mathrm{a}}$, $i_{3,\mathrm{b}}$ and $i_{3,\mathrm{c}}$. 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 (p. 1056) 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, Fig. 16.125 shows a possible implementation of a translational damper with variable damping constant and prescribed initial displacement.

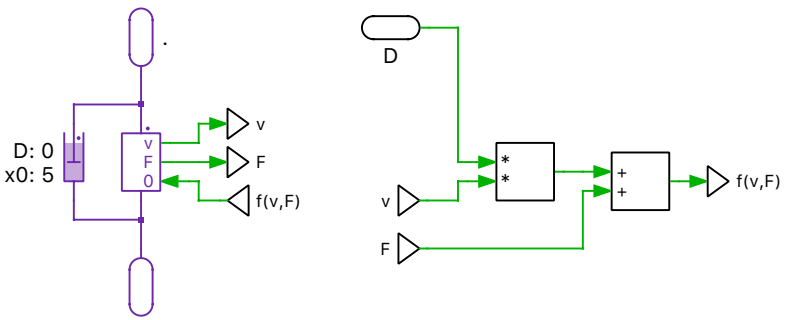


Fig. 16.125: Variable translational damper implementation example

> **Note**
>
> The Translational Algebraic Component creates an algebraic loop. See “Block Sorting” (p. 36) 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.

↪ *Added in PLECS 4.1.*

Translational Backlash

Purpose

Ideal translational backlash

Library

Mechanical / Translational / Components

Description



The Translational Backlash models an ideal, symmetrical, two-sided Translational Hard Stop (p. 1060) in a translational system, which restricts the relative displacement of the two flanges between an upper and lower limit of $\pm\frac{b}{2}$. 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.

Parameters

Total backlash

The total permitted displacement b between the flanges, in meters (m).

Initial displacement

The initial displacement of the flanges, in meters (m). 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

Force

The transmitted force flowing from the unmarked to the marked flange, in newtons (N).

Displacement

The displacement 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 Clutch

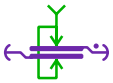
Purpose

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 (m). 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 $(\frac{\mathrm{Ns}}{\mathrm{m}})$.

Initial displacement

The initial displacement of the flanges, in meters (m). 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 $c_{\mathrm{v}} \cdot v$.

Fig. 16.126 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.

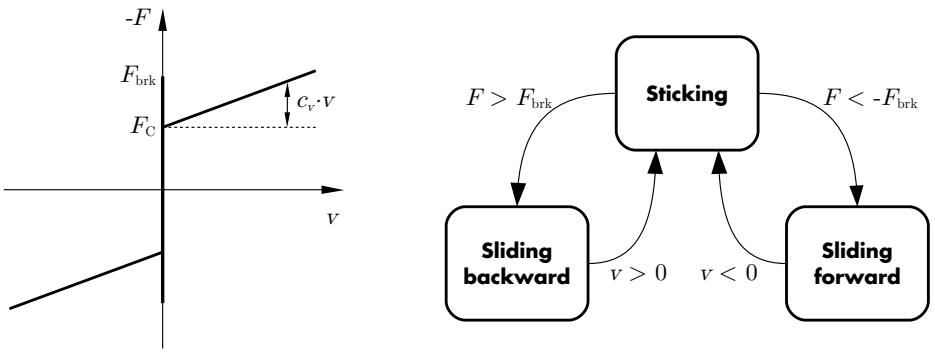


Fig. 16.126: Translational friction speed/force characteristic and state chart

Parameters

Breakaway friction force

The maximum magnitude of the stiction force F_{brk}, in newtons (N). Must be greater than or equal to zero.

Coulomb friction force

The magnitude of the (constant) Coulomb friction force F_{C}, in newtons (N). 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 $(\frac{\mathrm{Ns}}{\mathrm{m}})$.

Probe Signals

Force

The transmitted force F flowing from the unmarked to the marked flange, in newtons (N).

Speed

The speed v of the marked flange with respect to the unmarked flange, in $(\frac{\mathrm{m}}{\mathrm{s}})$.

State

The internal state of the component: -1 sliding backward, 0 stuck, $+1$ sliding forward.

Translational Friction (Controlled)

Purpose

Generate variable translational friction

Library

Mechanical / Translational / Components

Description



The Controlled Translational Friction component models the friction between two flanges in a translational system. It maintains zero relative velocity between the flanges up to the limit of the breakaway force F_{brk}. When the breakaway force is exceeded, the flanges slide against each other and the component generates the Coulomb friction force F_{C}.

This component does not include viscous friction.

The breakaway force F_{brk} is determined by the control signal and must always be greater than or equal to zero. The magnitude of the Coulomb friction F_{C} is proportional to the control signal.

Parameters

Relative Coulomb friction force

The coefficient c_C determines the magnitude of the Coulomb friction force $F_{\mathrm{C}} = c_C \cdot F_{\mathrm{brk}}$ and must be between zero and one.

Probe Signals

Force

The transmitted force F flowing from the unmarked to the marked flange, in newtons (N).

Speed

The speed v of the marked flange with respect to the unmarked flange, in $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$.

State

The internal state of the component: -1 sliding backward, 0 stuck, $+1$ sliding forward.

↪ *Added in PLECS 4.9.*

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.

Fig. 16.127 shows the displacement/force characteristic and the state chart of the component.

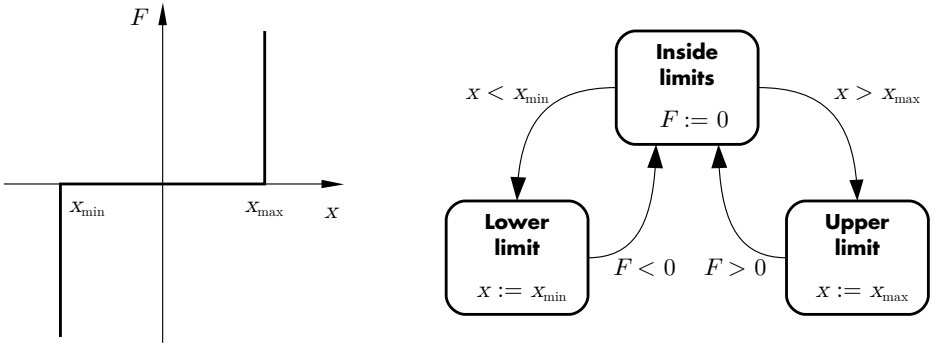


Fig. 16.127: Translational hard stop displacement/force characteristic and state chart

Parameters

Upper limit

The maximum displacement $x_{\max}$ between the flanges, in meters (m). Set to `inf` to disable this limit.

Lower limit

The minimum displacement $x_{\min}$ between the flanges, in meters (m). Set to `-inf` to disable this limit.

Initial displacement

The initial displacement of the flanges, in meters (m).

Probe Signals

Force

The transmitted force F flowing from the unmarked to the marked flange, in newtons (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

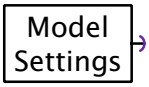
Purpose

Configure settings for an individual mechanical model.

Library

Mechanical / Translational / Model Settings

Description



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" (p. 342).

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 (p. 897) have coupled model equations. See also the options **Enable state-space splitting** and **Display state-space splitting** in the Simulation Parameters (p. 135).

Parameters

Switching algorithm

This parameter allows you two choose between two algorithms to determine the clutch states in the generated code. See "Switching Algorithm" (p. 343) 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.

↪ *Added in PLECS 4.0.*

Translational Multiplexer

Purpose

Combine several translational connections into single vector

Library

Mechanical / Translational / Connectivity

Description

This multiplexer combines several translational connections into one vector connection. The individual connections may themselves be vectors. In the block icon, the first individual connection is marked with a dot.



Parameters

Width

This parameter allows you to specify the number and/or width of the individual connections. You can choose between the following formats for this parameter:

Scalar: A scalar specifies the number of individual connections each having a width of 1.

Vector: The length of the vector determines the number of individual connections. Each element specifies the width of the corresponding individual connections.

↪ *Added in PLECS 4.6.*

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 (p. 977). 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.

A terminal can be moved around the edges of the subsystem by dragging it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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 a 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.

Parameters

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 input terminals are shown, and `right` refers to the side on which 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 Selector

Purpose

Select or reorder elements from a vector connection

Library

Mechanical / Translational / Connectivity

Description

The Translational Selector block connects the individual elements of the output connection to the specified elements of the input connection. The input connection is marked with a dot.



Parameters

Input width

The width of the input connection.

Output indices

A vector with the indices of the input elements that the output connection should contain.

↪ *Added in PLECS 4.6.*

Translational Speed (Constant)

Purpose

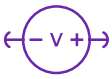
Maintain constant translational speed

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 $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$. The default value is `1`.

Probe Signals

Force

The generated force flowing from the unmarked flange to the marked flange, in newtons (N).

Speed

The speed, in $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$.

Translational Speed (Controlled)

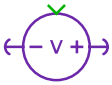
Purpose

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 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 newtons (N).

Speed

The speed, in $\left(\frac{\mathrm{m}}{\mathrm{s}}\right)$.

Translational Speed Sensor

Purpose

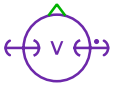
Output measured linear speed as signal

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.

Parameters

Second flange

Controls whether the second flange is accessible or connected to the translational reference frame.

Probe Signals

Speed

The measured speed, in $(\frac{\mathrm{m}}{\mathrm{s}})$.

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**
>
> A 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 $(\frac{\mathrm{N}}{\mathrm{m}})$.

Equilibrium (unstretched) displacement
The displacement x_0 between the two flanges of the unloaded spring (m).

Initial deformation
The initial deformation Δx_0 of the spring, in meters (m).

Probe Signals

Force
The spring force F, in newtons (N).

Deformation
The spring deformation Δx, in meters (m).

Transmission Line (3ph)

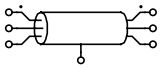
Purpose

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. Fig. 16.128 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.

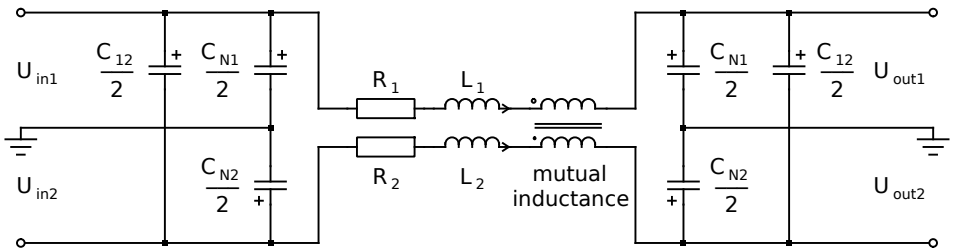


Fig. 16.128: Schematic of a single Pi-Section for a 2-phase line

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:

$$L = \frac{l}{n}L', \qquad R = \frac{l}{n}R', \qquad C_N = \frac{l}{n}C'_N$$

$$C_{ij} = \frac{l}{n} C'_{ij}, \qquad L_{ij} = \frac{l}{n} L'_{ij}$$

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

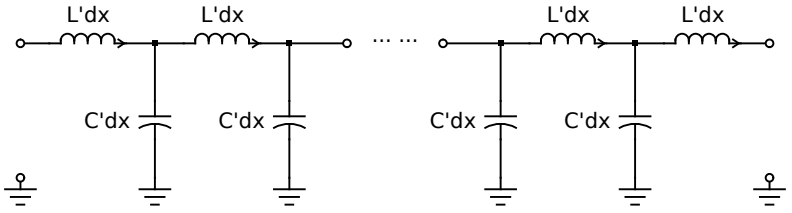


Fig. 16.129: A single-phase lossless line

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:

$$i_{\mathrm{s}} = \frac{1}{Z} \cdot e_{\mathrm{s}}(t) - I_{\mathrm{sh}}$$

$$i_{\mathrm{r}} = \frac{1}{Z} \cdot e_{\mathrm{r}}(t) - I_{\mathrm{rh}}$$

where

$$I_{\mathrm{sh}} = \frac{1}{Z} \cdot e_{\mathrm{r}}(t-\tau) + i_{\mathrm{r}}(t-\tau)$$

$$I_{\mathrm{rh}} = \frac{1}{Z} \cdot e_{\mathrm{s}}(t-\tau) + i_{\mathrm{s}}(t-\tau)$$

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 τ to port "r" via the controlled current source I_{rh} as in Fig. 16.130.

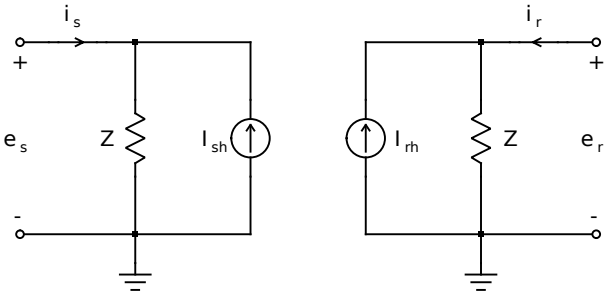


Fig. 16.130: Two-port equivalent circuit for model

Approximation of Series Resistance

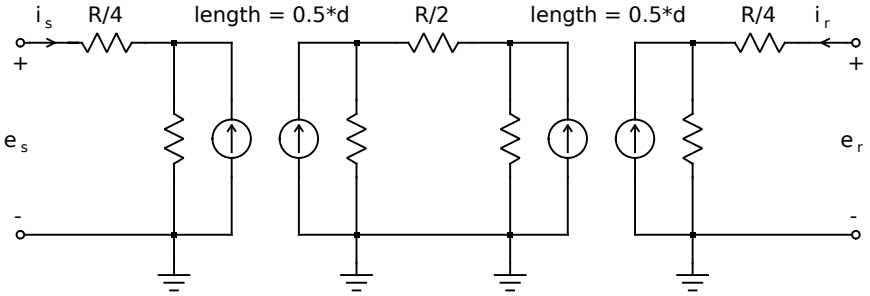


Fig. 16.131: Approximation of Series Resistance

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 (Fig. 16.131). $R = R' \cdot d$ is the total series resistance of the line. After aggregation and substitution, the original expression of the two equivalent current source becomes:

$$I_{\mathrm{sh}} = \frac{1+h}{2} \cdot \left(\frac{1}{Z_{\mathrm{R}}} \cdot e_{\mathrm{r}}(t-\tau) + h \cdot i_{\mathrm{r}}(t-\tau) \right) + \frac{1-h}{2} \left(\frac{1}{Z_{\mathrm{R}}} \cdot e_{\mathrm{s}}(t-\tau) + h \cdot i_{\mathrm{s}}(t-\tau) \right)$$

$$I_{\mathrm{rh}} = \frac{1+h}{2} \cdot \left(\frac{1}{Z_{\mathrm{R}}} \cdot e_{\mathrm{s}}(t-\tau) + h \cdot i_{\mathrm{s}}(t-\tau) \right) + \frac{1-h}{2} \left(\frac{1}{Z_{\mathrm{R}}} \cdot e_{\mathrm{r}}(t-\tau) + h \cdot i_{\mathrm{r}}(t-\tau) \right)$$

with $Z_{\mathrm{R}} = Z + \frac{R}{4}$ and $h = \frac{Z - \frac{R}{4}}{Z + \frac{R}{4}}$.

Three-Phase Line

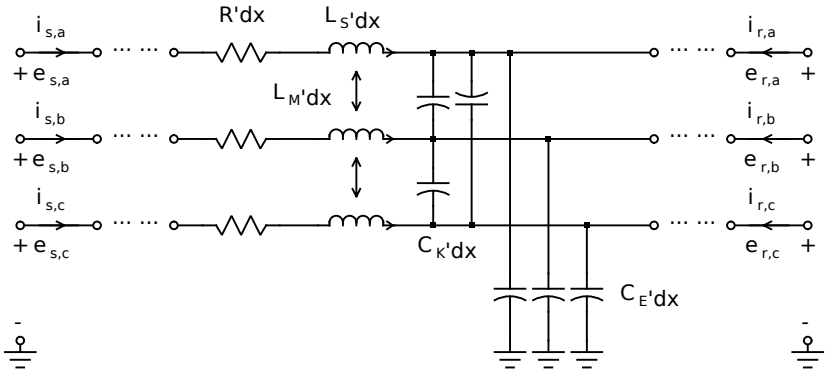


Fig. 16.132: Three-phased line

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:

$$\mathbf{L}'_{\mathrm{mod}} = \mathbf{T} \cdot \mathbf{L}' \cdot \mathbf{T}^{-1} = \begin{bmatrix} L'_u & 0 & 0 \\ 0 & L'_v & 0 \\ 0 & 0 & L'_w \end{bmatrix}$$

$$\mathbf{C'}_{\mathrm{mod}} = \mathbf{T} \cdot \mathbf{C'} \cdot \mathbf{T}^{-1} = \begin{bmatrix} C'_u & 0 & 0 \\ 0 & C'_v & 0 \\ 0 & 0 & C'_w \end{bmatrix}$$

$$\mathbf{R'}_{\mathrm{mod}} = \mathbf{T} \cdot \mathbf{R'} \cdot \mathbf{T}^{-1} = \begin{bmatrix} R' & 0 & 0 \\ 0 & R' & 0 \\ 0 & 0 & R' \end{bmatrix} = \mathbf{R'}$$

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; see Fig. 16.133). They can be treated separately in the same way as the single-phase system.

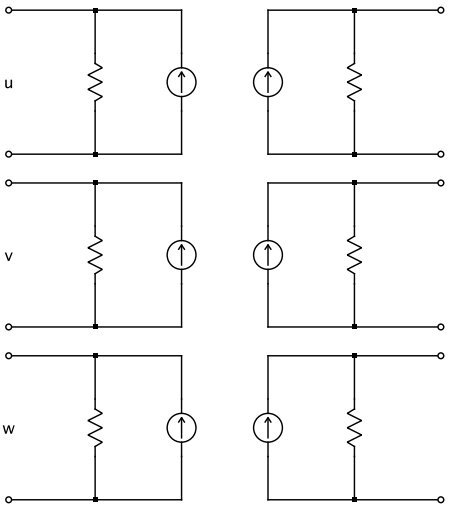


Fig. 16.133: Three decoupled systems in the modal domain

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 $(\mathrm{H/m})$.

For a pi-section line, the self inductance can be specified individually per phase by providing a 3-element vector of the form $L' = [L'_1 \quad L'_2 \quad L'_3]$.

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 $(\mathrm{H/m})$.

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 $\mathrm{M'} = [M'_{12} \quad M'_{13} \quad M'_{23}]$ 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 (Ω/m).

For a pi-section line, the parameter can be a vector of the form $R' = [R'_1 \quad R'_2 \quad R'_3]$.

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 $(\mathrm{F/m})$.

For a pi-section line, this parameter can be a vector of the form $C'_{\mathrm{N}} = [C'_{\mathrm{N1}} \quad C'_{\mathrm{N2}} \quad C'_{\mathrm{N1}}]$.

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 $(\mathrm{F/m})$.

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 $\mathrm{C'} = [C'_{12} \quad C'_{13} \quad C'_{23}]$ 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.

References

- 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

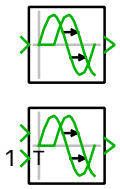
Purpose

Delay continuous input signal by fixed or variable time

Library

Control / Delays

Description



The Transport Delay outputs a signal that a past value of the input signal:

$$y(t) = u(t - T_d)$$

For this purpose, the Transport Delay continuously records the input signals in an internal 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. If the time delay is less than the previous simulation time step, the output signal is calculated by performing a linear interpolation from the preceding two samples. This is because the Transport Delay does not access the current input to calculate the outputs in order to avoid algebraic loops if the block is used in a feedback path.

If extrapolation is undesirable, you need to ensure that the maximum solver step size is less than or equal to the time delay.

Note

The Transport Delay should 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 (p. 895) or the Triangular Wave Generator (p. 1082) and set the parameter appropriately.
- To delay arbitrary signals that only change at discrete instants, use the Pulse Delay (p. 893).

Example Model

- See the example model **"Transport Delay"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

Parameters

Time delay source
Specifies whether the delay is determined by the **Time delay** parameter (`internal`) or by an external input signal (`external`).

↪ *Added in PLECS 4.5.*

Time delay
Time T_d by which the input signal is delayed, in seconds (s).

Maximum time delay
Specifies the maximum possible delay for the external delay input signal, in seconds (s).

Initial output
Output value after simulation start before the input values appear at the output.

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 (p. 1024) 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 (Ω). The default is `0`.

Initial conductivity

Initial conduction state of the TRIAC. The TRIAC is initially blocking if the parameter evaluates to zero, otherwise it is conducting.

Thermal description

Switching losses, conduction losses and thermal equivalent circuit of the component. For more information see chapter “Thermal Modeling” (p. 171). If no thermal description is given, the losses are calculated based on the voltage drop $v_\mathrm{on} = V_\mathrm{f} + R_\mathrm{on} \cdot i$.

Thermal interface resistance

The thermal resistance of the interface material between case and heat sink,in (K/W). The default is `0`.

↪ *Added in PLECS 4.6.*

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” (p. 177).

Switch model (CPU code generation)
Select the switch model in the generated code.

Fundamental grid frequency
Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is 0.

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 (Hz).

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 (s). If the phase is set to 0, the waveform begins at the rising edge.

Probe Signals

Output
The block output signal.

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” (p. 978)) 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, drag it with the middle mouse button. Alternatively, use the left mouse button while holding the **Shift** key.

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 Signals

Output
The output signal of the Trigger block as described for the parameter **Show output port**.

Trigonometric Function

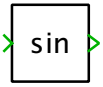
Purpose

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 Active Bridge

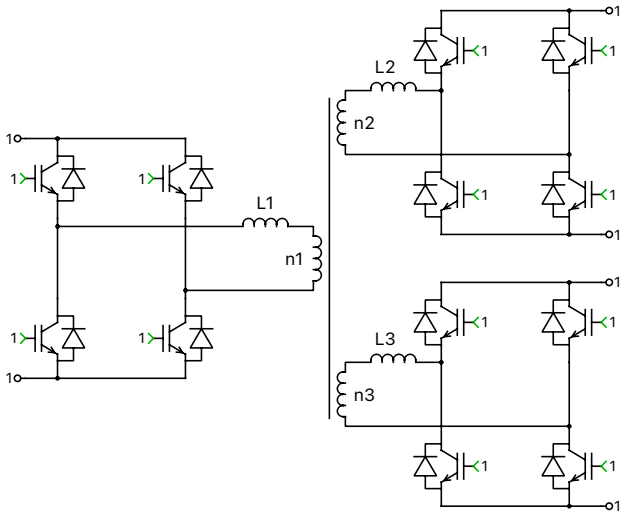
Purpose

Triple active bridge converter without magnetizing inductance, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements an isolated triple active bridge (TAB) converter with an optional resonant capacitor on the primary side.

Both sides of the converter have current source behavior and must each be connected directly to a capacitor or a voltage source.

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> The voltage sources connected to the converter must not be short-circuited. Therefore, combinations of switching signals that cause shoot-through of DC links are not permitted. Additionally, DC voltages across diodes must not be negative, as DC voltage clamping through diodes is not permitted.

Parameters

Certain parameters are associated with specific components. The corresponding component labels such as L, C, etc., are shown in Fig. 16.134 on p. 1089.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Primary inductance
A non-zero scalar or 3-element vector specifying the inductances L, in henries (H).

Winding resistance
A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Include primary capacitor
Allows you to include (`yes`) or remove (`no`) a resonant capacitor at the primary side.

Resonant capacitance
If the **Include primary capacitor** option is set to `yes`, this parameter requires a non-zero scalar for the capacitance C, in farads (F).

Turns ratio
A scalar specifying the ratio of primary-side turns to secondary-side turns.

Assertions
When set to `on`, the block flags an error for combinations of gate signals that cause shoot-through.

Probe Signals

See Fig. 16.134 on p. 1089 for the probe signal positions.

Inductor currents
A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Nanostep Probe Signals

Inductor currents
A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

Primary current
The current at the primary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Secondary current
The current at the secondary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Tertiary current
The current at the tertiary-side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Capacitor voltage
The voltage across the capacitor C, in volts (V), defined positive at the side marked with (+).

Primary gate signals
A vector containing the gate signals of the primary-side semiconductor switches.

Secondary gate signals
A vector containing the gate signals of the secondary-side semiconductor switches.

Tertiary gate signals
A vector containing the gate signals of the tertiary-side semiconductor switches.

↪ *Added in PLECS 4.9.*

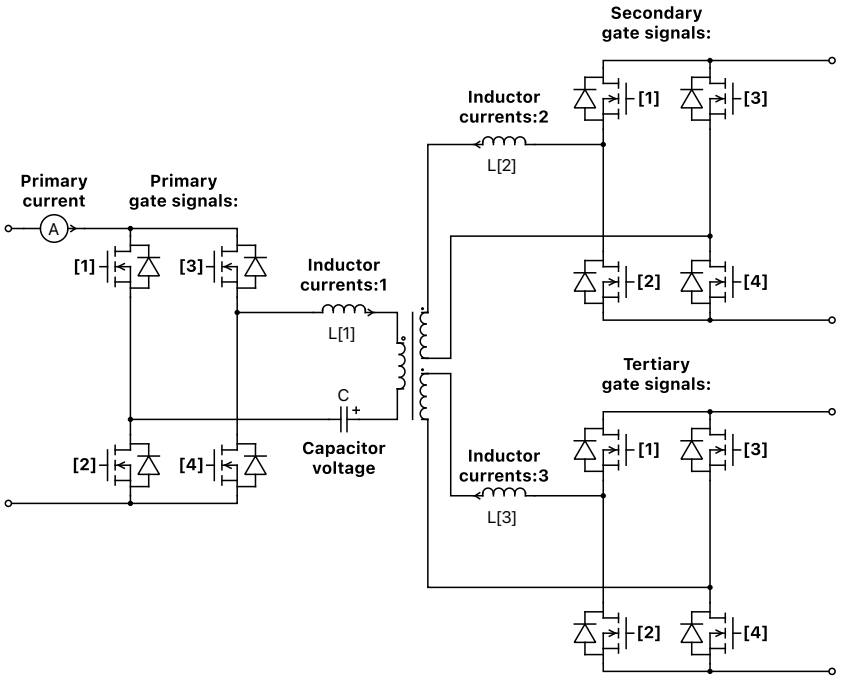


Fig. 16.134: Probes for the Triple Active Bridge

Triple Switch

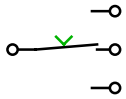
Purpose

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.

Parameters

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`.

Switch model (CPU code generation)

Select the switch model in the generated code.

Fundamental grid frequency

Frequency where non-ideal switches in the open state have increased series impedance, in hertz (Hz). The default is `0`.

Probe Signals

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.

Turn-on Delay

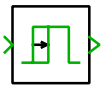
Purpose

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.

Example Model

- See the example model **"Turn-on Delay"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

Parameters

Dead time source

Specifies whether the dead time is determined by the **Dead time** parameter (`internal`) or by an external input signal (`external`).

↪ *Added in PLECS 4.3.*

Dead time

Time by which the turn-on event is delayed, in seconds (s). 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

$$i = \frac{\mathrm{d}}{\mathrm{d}t} C \cdot v + C \cdot \frac{\mathrm{d}}{\mathrm{d}t} v$$

Since v is the state variable the equation above must be solved for $\frac{\mathrm{d}v}{\mathrm{d}t}$. The control signal must provide the values of both C and $\frac{\mathrm{d}}{\mathrm{d}t}C$ in the following form: $\left[C_1\ C_2 \ldots C_n\ \frac{\mathrm{d}}{\mathrm{d}t}C_1\ \frac{\mathrm{d}}{\mathrm{d}t}C_2 \ldots \frac{\mathrm{d}}{\mathrm{d}t}C_n\right]$. 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: $\left[C_{11}\ C_{12}\ C_{21}\ C_{22}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{11}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{12}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{21}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{22}\right]$. 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 $C_{\mathrm{tot}}(v) = Q/v$ and the *differential* capacitance $C_{\mathrm{diff}}(v) = \mathrm{d}Q/\mathrm{d}v$.

With the *total* capacitance $C_{\mathrm{tot}}(v) = Q/v$ you have

$$\begin{aligned} i &= \frac{\mathrm{d}Q}{\mathrm{d}t} \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\left(C_{\mathrm{tot}} \cdot v\right) \\ &= C_{\mathrm{tot}} \cdot \frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\mathrm{d}C_{\mathrm{tot}}}{\mathrm{d}t} \cdot v \\ &= C_{\mathrm{tot}} \cdot \frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\mathrm{d}C_{\mathrm{tot}}}{\mathrm{d}v} \cdot \frac{\mathrm{d}v}{\mathrm{d}t} \cdot v \\ &= \left(C_{\mathrm{tot}} + \frac{\mathrm{d}C_{\mathrm{tot}}}{\mathrm{d}v} \cdot v\right) \cdot \frac{\mathrm{d}v}{\mathrm{d}t} \end{aligned}$$

which can be implemented as in Fig. 16.135.

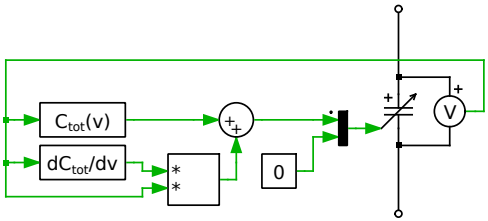


Fig. 16.135: Schematic for first equation

With the *differential* capacitance $C_{\mathrm{diff}}(v) = \mathrm{d}Q/\mathrm{d}v$ you have

$$\begin{aligned} i &= \frac{\mathrm{d}Q}{\mathrm{d}t} \\ &= \frac{\mathrm{d}Q}{\mathrm{d}v} \cdot \frac{\mathrm{d}v}{\mathrm{d}t} \\ &= C_{\mathrm{diff}} \cdot \frac{\mathrm{d}v}{\mathrm{d}t} \end{aligned}$$

which can be implemented as in Fig. 16.136:

Note that in both cases the $\frac{\mathrm{d}}{\mathrm{d}t}C$-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: $C = C(x)$.

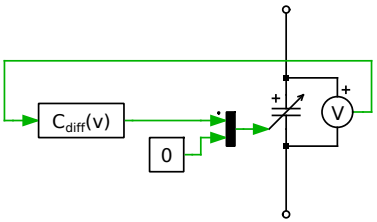


Fig. 16.136: Schematic for second equation

Therefore you have

$$\begin{aligned} i &= C \cdot \frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\mathrm{d}C}{\mathrm{d}t} \cdot v \\ &= C \cdot \frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\mathrm{d}C}{\mathrm{d}x} \cdot \frac{\mathrm{d}x}{\mathrm{d}t} \cdot v \end{aligned}$$

which can be implemented as in Fig. 16.137.

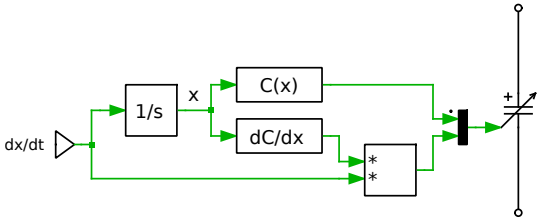


Fig. 16.137: Schematic for third equation

Note that x is preferably calculated as the integral of $\mathrm{d}x/\mathrm{d}t$ rather than calculating $\mathrm{d}x/\mathrm{d}t$ 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 Frequency PWM

Purpose

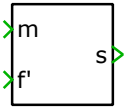
Generate PWM signals with variable frequency

Library

Control / Modulators

Description

The Variable Frequency PWM block generates one or several PWM signals with a variable switching frequency.



The block offers different regular sampling methods for the modulation index if the parameter **Carrier type** is set to `Symmetrical (carrier counts up/down)`. If double edge sampling is used, the modulation index is updated at the minimum and maximum of the carrier. The regular sampling methods are visualized in the Symmetrical PWM block (p. 988). If the **Carrier type** parameter is set to `Sawtooth (carrier counts up)`, the modulation index is updated when the carrier reaches the minimum.

The f' input is sampled when the carrier reaches the minimum, regardless of the **Carrier type**. Fig. 16.138 illustrates the f' input sampling and the sampling instant is indicated by a solid black dot.

Inputs and Outputs

The input m is the modulation index, f' is a frequency scaling factor, 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 input f' has to be a scalar value. The output s is in on-state if the modulation index m is greater than the carrier.

m

A scalar or vector representing the modulation index.

f'

A scalar signal that represents a scaling factor for the nominal carrier frequency. The resulting carrier frequency is calculated as $f_{\mathrm{PWM}} = f' \cdot f_{\mathrm{nominal}}$, where f_{nominal} is the value of the **Nominal carrier frequency** parameter. The f' input must be in the range $[0.001, 1000]$.

Parameters

Carrier type

Specifies the carrier type. `Sawtooth (carrier counts up)` and `Symmetrical (carrier counts up/down)` counter modes are supported.

Modulation index sampling

The Regular Sampling method lets you choose among single update at either min or max, and double update at both min and max of the carrier. This parameter is visible if the **Carrier type** parameter is set to `Symmetrical (carrier counts up/down)`.

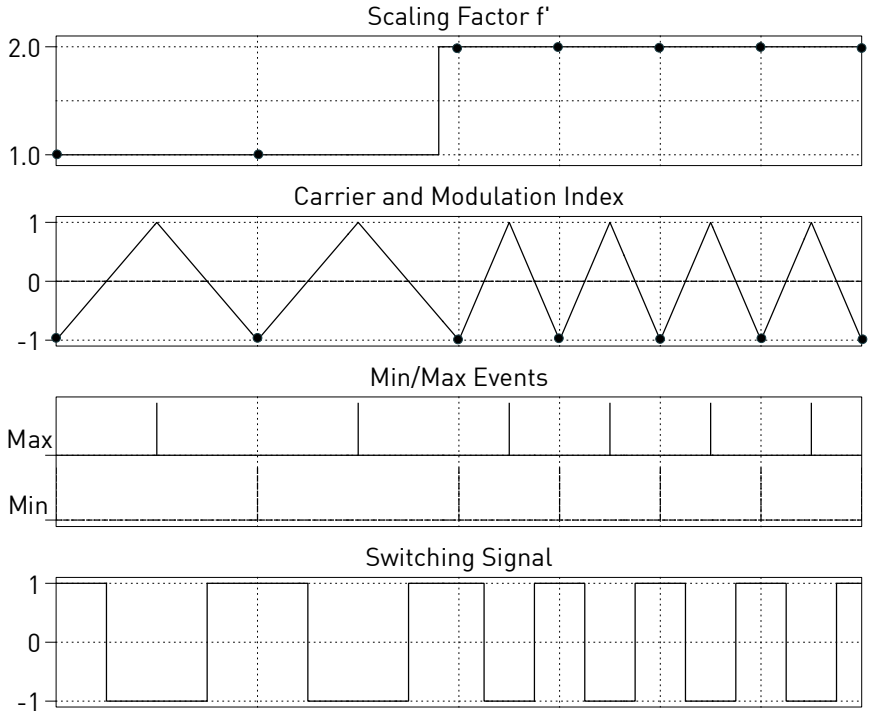


Fig. 16.138: Variable frequency PWM sampling diagram

Nominal carrier frequency
The nominal frequency f_{nominal} of the carrier signal when the f' scaling factor input is equal to 1, in hertz (Hz).

Carrier limits [min max]
The range of the carrier. The default is `[-1 1]`.

Output values [off on]
Values of the switching function in off-state and on-state. The default is `[-1 1]`.

Probe Signals

Max event
Outputs a rising edge if the carrier reaches the maximum value.

Min event
Outputs a rising edge if the carrier reaches the minimum value.

Carrier
Outputs the carrier signal for each channel.

↪ *Added in PLECS 4.7.*

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:

$$v = L \cdot \frac{\mathrm{d}i}{\mathrm{d}t} + \frac{\mathrm{d}L}{\mathrm{d}t} \cdot i$$

Since i is the state variable the equation above must be solved for $\frac{\mathrm{d}i}{\mathrm{d}t}$. The control signal must provide the values of both L and $\frac{\mathrm{d}}{\mathrm{d}t}L$ in the following form: $\left[L_1\ L_2 \ldots L_n\ \frac{\mathrm{d}}{\mathrm{d}t}L_1\ \frac{\mathrm{d}}{\mathrm{d}t}L_2 \ldots \frac{\mathrm{d}}{\mathrm{d}t}L_n\right]$. 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: $\left[L_{11}\ L_{12}\ L_{21}\ L_{22}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{11}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{12}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{21}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{22}\right]$. 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 (p. 982).

Saturable Inductor Modeling

When specifying the characteristic of a saturable inductor, you need to distinguish carefully between the *total* inductivity $L_{\mathrm{tot}}(i) = \Psi/i$ and the *differential* inductivity $L_{\mathrm{diff}}(i) = \mathrm{d}\Psi/\mathrm{d}i$. See also the piece-wise linear Saturable Inductor (p. 931).

With the *total* inductivity $L_{\mathrm{tot}}(i) = \Psi/i$ you have

$$\begin{aligned} v &= \frac{\mathrm{d}\Psi}{\mathrm{d}t} \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\left(L_{\mathrm{tot}} \cdot i\right) \\ &= L_{\mathrm{tot}} \cdot \frac{\mathrm{d}i}{\mathrm{d}t} + \frac{dL_{\mathrm{tot}}}{\mathrm{d}t} \cdot i \\ &= L_{\mathrm{tot}} \cdot \frac{\mathrm{d}i}{\mathrm{d}t} + \frac{dL_{\mathrm{tot}}}{\mathrm{d}i} \cdot \frac{\mathrm{d}i}{\mathrm{d}t} \cdot i \\ &= \left(L_{\mathrm{tot}} + \frac{\mathrm{d}L_{\mathrm{tot}}}{\mathrm{d}i} \cdot i\right) \cdot \frac{\mathrm{d}i}{\mathrm{d}t} \quad , \end{aligned}$$

which can be implemented as in Fig. 16.139.

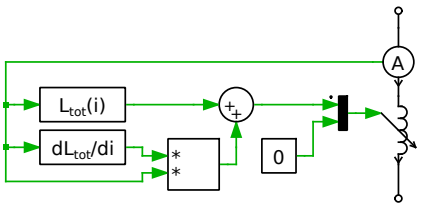


Fig. 16.139: Schematic for first equation

With the *differential* inductivity $L_{\mathrm{diff}}(i) = \mathrm{d}\Psi/\mathrm{d}i$ you have

$$\begin{aligned} v &= \frac{\mathrm{d}\Psi}{\mathrm{d}t} \\ &= \frac{\mathrm{d}\Psi}{\mathrm{d}i} \cdot \frac{\mathrm{d}i}{\mathrm{d}t} \\ &= L_{\mathrm{diff}} \cdot \frac{\mathrm{d}i}{\mathrm{d}t} \quad , \end{aligned}$$

which can be implemented as in Fig. 16.140.

Note that in both cases the $\frac{\mathrm{d}}{\mathrm{d}t}L$-input of the Variable Inductor is zero!

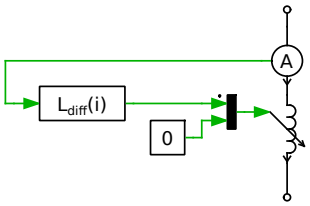


Fig. 16.140: Schematic for second equation

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

$$\begin{aligned} v &= L \cdot \frac{\mathrm{d}i}{\mathrm{d}t} + \frac{\mathrm{d}L}{\mathrm{d}t} \cdot i \\ &= L \cdot \frac{\mathrm{d}i}{\mathrm{d}t} + \frac{\mathrm{d}L}{\mathrm{d}x} \cdot \frac{\mathrm{d}x}{\mathrm{d}t} \cdot i \end{aligned} \quad ,$$

which can be implemented as in Fig. 16.141.

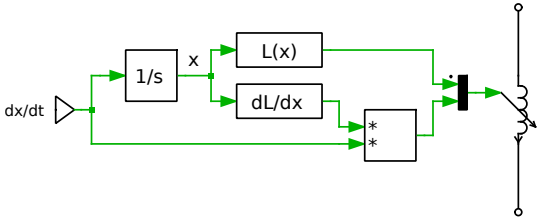


Fig. 16.141: Schematic for third equation

Note that x is preferably calculated as the integral of $\mathrm{d}x/\mathrm{d}t$ rather than calculating $\mathrm{d}x/\mathrm{d}t$ as the derivative of x.

Example Model

- See the example model **"Variable Inductor"**.
- Find it in PLECS under **Help > PLECS Documentation >** List of Example Models.

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

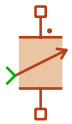
Purpose

Variable permeance controlled by external signal

Library

Magnetic / Components

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:

$$\dot{\Phi} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\mathcal{P}\cdot\mathcal{F}\right) = \mathcal{P}\cdot\frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t} + \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{P}\cdot\mathcal{F}$$

Since $\mathcal{F}$ is the state variable the equation above must be solved for $\frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t}$. The control signal must provide the values of $\mathcal{P}(t)$, $\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{P}(t)$ and Φ as a vector. It is the responsibility of the user to provide the appropriate signals.

Modeling non-linear material properties

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

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

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

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

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

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

$$\begin{aligned} \dot{\Phi} &= \frac{\mathrm{d}\Phi}{\mathrm{d}t} \\ &= \frac{\mathrm{d}\Phi}{\mathrm{d}\mathcal{F}} \cdot \frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t} \\ &= \mathcal{P}_{\mathrm{diff}} \cdot \frac{\mathrm{d}\mathcal{F}}{\mathrm{d}t} \end{aligned}$$

the control signal is:

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

Parameters

Initial MMF

Magneto-motive force at simulation start, in ampere-turns (A).

Probe Signals

MMF

The magneto-motive force measured from the marked to the unmarked terminal, in ampere-turns (A).

Flux

The magnetic flux flowing through the component, in webers (Wb). A flux entering at the marked terminal is counted as positive.

Variable Phase PWM

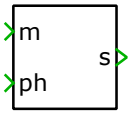
Purpose

Generate PWM signals with variable phase shift

Library

Control / Modulators

Description



The Variable Phase PWM block can generate phase-shifted PWM signals.

The block offers different regular sampling methods for the modulation index if the parameter **Carrier type** is set to `Symmetrical (carrier counts up/down)`. If double edge sampling is used, the modulation index is updated at the minimum and maximum of the carrier. The regular sampling methods are visualized in the Symmetrical PWM block (p. 988). For a `Sawtooth (carrier counts up)` **Carrier type**, the modulation index is updated when the carrier reaches the minimum.

The ph input is only sampled when the carrier of the internal master channel reaches the minimum, regardless of the **Carrier type**. The figure below illustrates the sampling of the ph input. The sampling instant is indicated by a solid black dot. The phase shift of the channels are always relative to the internal master channel.

Inputs and Outputs

The input m is the modulation index, ph the phase shift in per unit $(\mathrm{p.u.})$, and the output s is the switching function. If the modulation index is a vector, the switching function and the ph input are also a vector of the same width. The output s of a channel goes to the on-state if the modulation index m of the respective channel is greater than the carrier.

m

A scalar or vector representing the modulation index. If the modulation is a scalar signal and the ph input a vector, the output is a vector of the same width as the ph input. In this case, all output channels will have an identical modulation index.

ph

A vector representing the phase shift of each output in respect to the internal master channel, in per unit $(\mathrm{p.u.})$.

Parameters

Carrier type

Specifies the carrier type. `Sawtooth (carrier counts up)` and `Symmetrical (carrier counts up/down)` counter modes are supported.

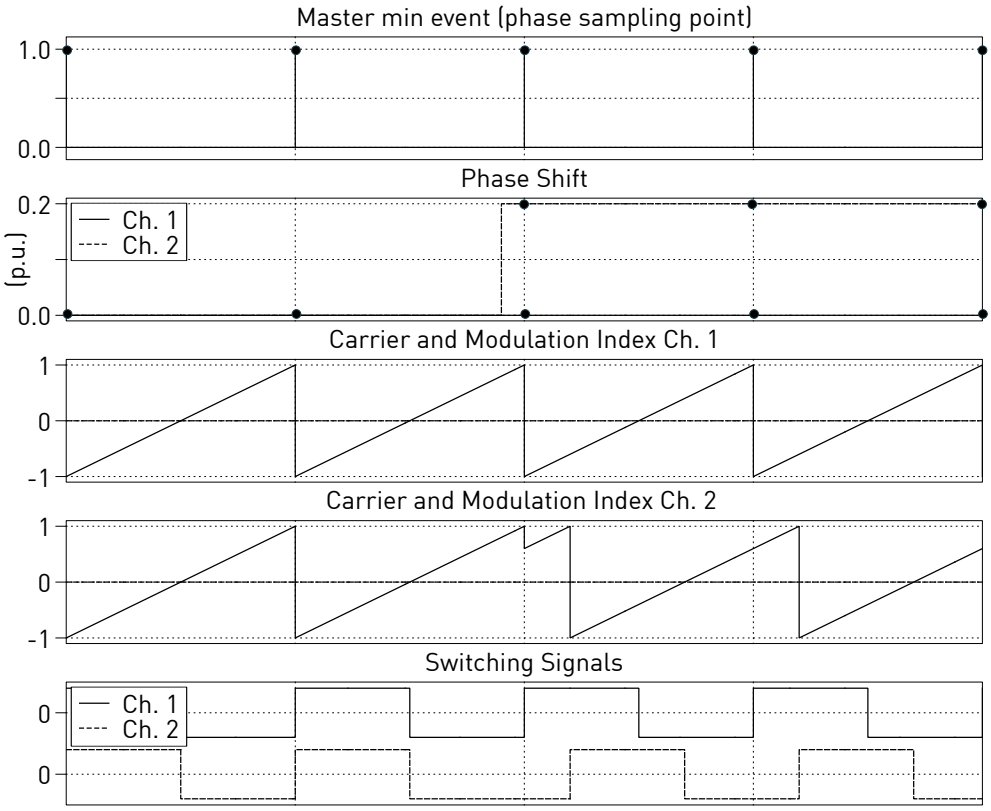


Fig. 16.142: Variable phase PWM with sawtooth carrier

Modulation index sampling
The Regular Sampling method of the modulation index lets you choose among single update at either min or max, and double update at both min and max of the carrier. This parameter is made visible if the **Carrier type** parameter is set to `Symmetrical (carrier counts up/down)`.

Carrier frequency
The frequency of the carrier signal, in hertz (Hz).

Carrier limits [min max]
The range of the carrier. The default is `[-1 1]`.

Output values [off on]
Values of the switching function in off-state and on-state. The default is

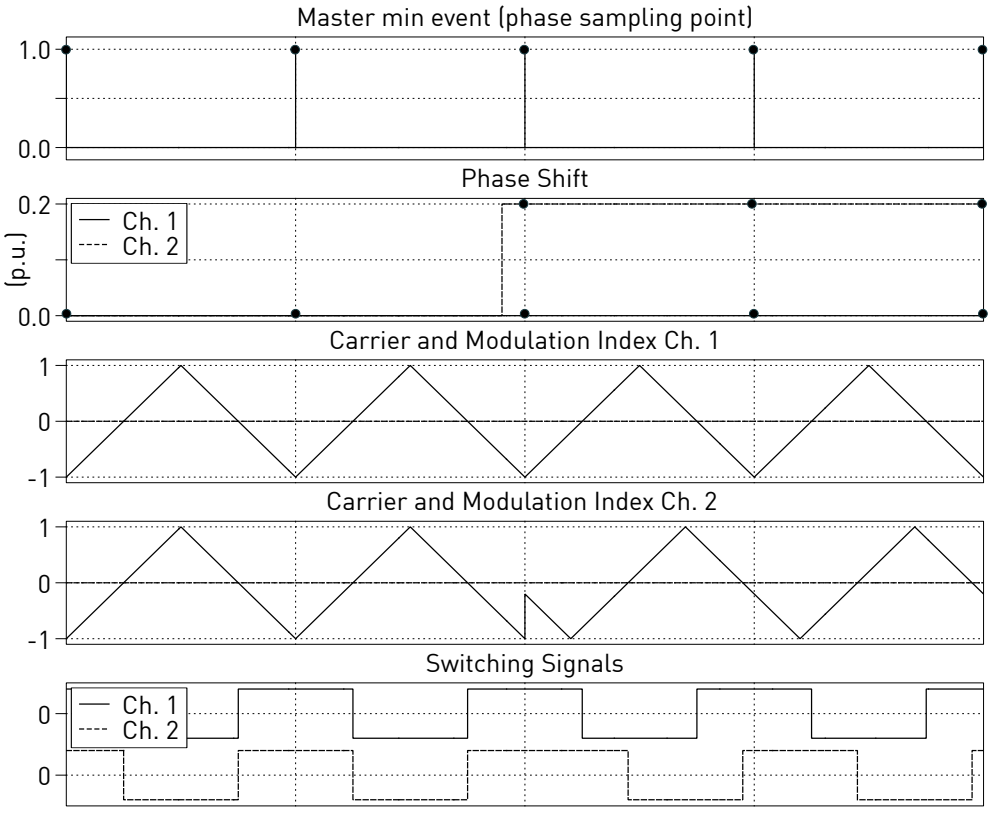


Fig. 16.143: Variable phase PWM with symmetrical carrier

[-1 1].

Probe Signals

Master max event
Outputs a rising edge if the internal master carrier reaches the maximum value.

Master min event (phase sampling point)
Outputs a rising edge if the internal master carrier reaches the minimum value. This instant also describes the point in time when a new phase value of each channel is sampled.

Master carrier
Outputs the carrier signal for each channel.

Max event
Outputs a rising edge if the carrier reaches the maximum value.

Min event
Outputs a rising edge if the carrier reaches the minimum value.

Carrier
Outputs the carrier signal for each channel.

↪ *Added in PLECS 4.7.*

Variable Resistor

Purpose

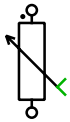
Resistance controlled by a signal

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” (p. 36) 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.

↪ *Added in PLECS 4.1.*

Variable Resistor with Constant Capacitor

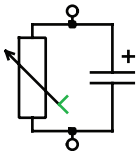
Purpose

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 $[C_1\ C_2 \ldots C_n]$. The length n of the vector determines the width of the component.

Initial voltage

The initial voltage of the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the width of the component. The positive pole is marked with a "+". The initial voltage default is `0`.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with "+" is greater than the potential at the unmarked terminal.

Variable Resistor with Constant Inductor

Purpose

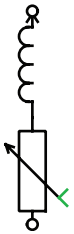
Controlled resistance in series with constant inductance

Library

Electrical / Passive Components

Description

This component models a variable resistor with a constant inductor connected in series. The resistance is determined by the signal fed into the input of the component.



> **Note**
>
> In this component the resistor is implemented as a current-dependent voltage source. Without the series inductor, which fixes the momentary current, this would result in an algebraic loop. Therefore, the inductance may not be set to zero.

Parameters

Inductance

The inductance in henries (H). All finite positive and negative values are accepted, excluding `0`. The default is `1e-3`.

In a vectorized component, all internal inductors have the same inductance if the parameter is a scalar. To specify the inductances individually use a vector $[L_1\ L_2 \ldots L_n]$. The length n of the vector determines the width of the component.

Initial current

The initial current through the component at simulation start, in amperes (A). This parameter may either be a scalar or a vector corresponding to the width of the component. The direction of a positive initial current is indicated by a small arrow in the component symbol. The default of the initial current is `0`.

Probe Signals

Inductor current

The current flowing through the inductor, in amperes (A). The direction of a positive current is indicated with a small arrow in the component symbol.

Variable Resistor with Variable Capacitor

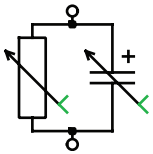
Purpose

Controlled resistance in parallel with controlled capacitance

Library

Electrical / Passive Components

Description



This component models a variable resistor with a variable capacitor connected in parallel. The resistance and capacitance are determined by the signals fed into the inputs of the component. The current through this component is determined by the equation:

$$i = \left(\frac{1}{R} + \frac{\mathrm{d}}{\mathrm{d}t}C\right) \cdot v + C \cdot \frac{\mathrm{d}}{\mathrm{d}t}v$$

The control signal for the capacitor must provide the values of both C and $\frac{\mathrm{d}}{\mathrm{d}t}C$ in the following form: $\left[C_1\ C_2 \ldots C_n\ \frac{\mathrm{d}}{\mathrm{d}t}C_1\ \frac{\mathrm{d}}{\mathrm{d}t}C_2 \ldots \frac{\mathrm{d}}{\mathrm{d}t}C_n\right]$. It is the responsibility of the user to provide the appropriate signals for a particular purpose. For detailed information see the Variable Capacitor (p. 1092)

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: $\left[C_{11}\ C_{12}\ C_{21}\ C_{22}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{11}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{12}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{21}\ \frac{\mathrm{d}}{\mathrm{d}t}C_{22}\right]$. The control signal thus has a width of $2 \cdot n^2$, n being the number of phases.

> **Note**
>
> The momentary capacitance and the resistance may not be set to zero. In case of coupled capacitors, the capacitance matrix may not be singular.

Parameters

Capacitive coupling

Specifies whether the phases should be coupled capacitively. This parameter determines how the elements of the control signal are interpreted. The default is `off`.

Initial voltage

The initial voltage of the capacitor at simulation start, in volts (V). This parameter may either be a scalar or a vector corresponding to the implicit width of the component. The positive pole is marked with a "+". The initial voltage default is `0`.

Probe Signals

Capacitor voltage

The voltage measured across the capacitor, in volts (V). A positive voltage is measured when the potential at the terminal marked with "+" is greater than the potential at the unmarked terminal.

Variable Resistor with Variable Inductor

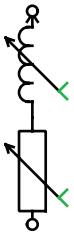
Purpose

Controlled resistance in series with controlled inductance

Library

Electrical / Passive Components

Description



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:

$$v = \left(R + \frac{\mathrm{d}}{\mathrm{d}t}L\right) \cdot i + L \cdot \frac{\mathrm{d}}{\mathrm{d}t}i$$

The control signal for the inductor must provide the values of both L and $\frac{\mathrm{d}}{\mathrm{d}t}L$ in the following form: $\left[L_1\ L_2 \ldots L_n\ \frac{\mathrm{d}}{\mathrm{d}t}L_1\ \frac{\mathrm{d}}{\mathrm{d}t}L_2 \ldots \frac{\mathrm{d}}{\mathrm{d}t}L_n\right]$. It is the responsibility of the user to provide the appropriate signals for a particular purpose. For detailed information see the Variable Inductor (p. 1099).

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: $\left[L_{11}\ L_{12}\ L_{21}\ L_{22}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{11}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{12}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{21}\ \frac{\mathrm{d}}{\mathrm{d}t}L_{22}\right]$. 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 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.

Vienna Rectifier

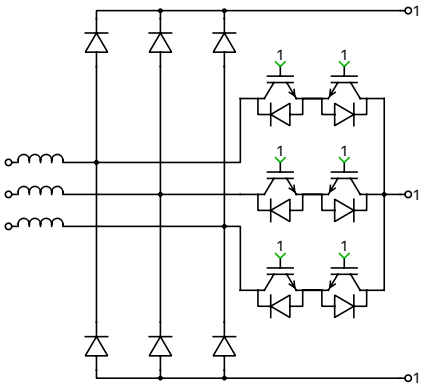
Purpose

3-phase Vienna rectifier with filter inductors, optimized for "Real-Time Simulations" (p. 360) on the RT Box (p. 361).

Library

Electrical / Nanostep

Description



This component implements a 3-phase Vienna rectifier. It includes filter inductors on the AC side.

Both sides of the converter have current source behavior. The DC side must be connected directly to a capacitor or a voltage source.

The component offers two configurations:

`Nanostep / Switched`

The power semiconductors are modelled as ideal switches (p. 344) controlled by instantaneous logical gate signals. A switch is turned on when the corresponding gate signal is `true`.

In `Nanostep / Switched` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on a Nanostep (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

The Nanostep implementation requires the DC and AC sides to be galvanically isolated. Any electrical connection between the two sides outside the component would therefore not reflect the actual current and voltage situation and should be avoided.

`Sub-cycle average`
Each gate input is controlled by a floating-point number with values between 0 and 1, representing the relative on-time of the corresponding semiconductor over one simulation step, as described in Sub-cycle Averaging (p. 489).

In `Sub-cycle average` configuration, C code generated (p. 341) from this component is suitable for real-time simulations (p. 360) on the RT Box (p. 361), where it is executed on the CPU (p. 362) or FlexArray (p. 363) solver. In this case, connect the gate inputs to PWM Capture blocks from the RT Box component library.

> **Note**
>
> This component has a Nanostep (p. 363) solver weight of 3, indicating that it occupies one entire Nanostep solver for Real-time Simulations (p. 360) on the RT Box (p. 361).

> **Note**
>
> DC voltage clamping through diodes is not permitted. Therefore, ensure that DC voltages across diodes are not negative.

Parameters

Configuration
Selects the component configuration. See the descriptions of `Nanostep / Switched` and `Sub-cycle average` above.

Switch model (CPU code generation)
If the **Configuration** is set to `Sub-cycle average`, this parameter selects the switch model (p. 344) when CPU (p. 362) code is generated.

Semiconductor symbol
Use this setting to choose between `IGBT` and `MOSFET` for the symbol of the active semiconductor switches. This setting only affects the schematic representation and does not change the electrical behavior of the component.

Inductance
A non-zero scalar or 3-element vector specifying the inductances L, in henries (H).

Winding resistance
A non-zero scalar or 3-element vector specifying the winding resistances R_L, in ohms (Ω).

Nanostep Probe Signals

See Fig. 16.144 for the probe signal positions.

Inductor currents

A vector containing the currents through the inductors L, in amperes (A), defined positive in the direction of the arrow.

DC link currents

A vector containing the currents at the DC-link side converter terminals, in amperes (A), defined positive in the direction of the arrow.

Gate signals

A vector for each phase containing the gate signals of the semiconductor switches.

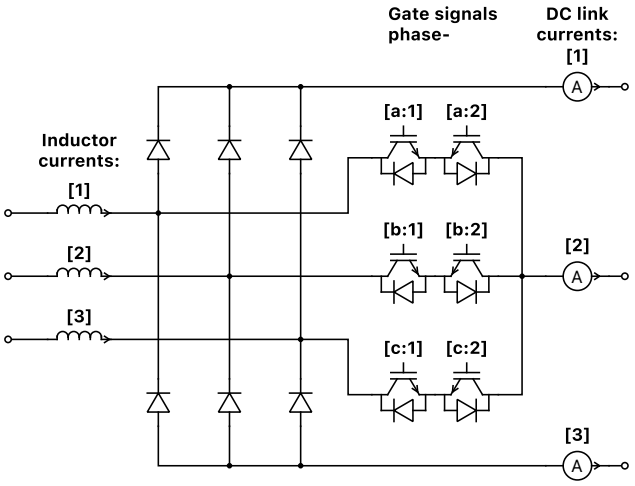


Fig. 16.144: Probes for the Vienna Rectifier

↪ *Added in PLECS 4.9.*

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" (p. 39).

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 ω, in $\left(\frac{\mathrm{rad}}{\mathrm{s}}\right)$. The default is `2*pi*50` which corresponds to $50\,\mathrm{Hz}$.

Phase
The phase shift φ, 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)

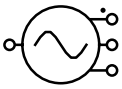
Purpose

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

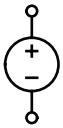
Generate constant voltage

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.

Parameters

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

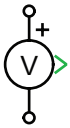
Purpose

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 Signal Outport block (p. 952) 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.

Probe Signals

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. Fig. 16.145 illustrates the distribution for three different sets of parameters.

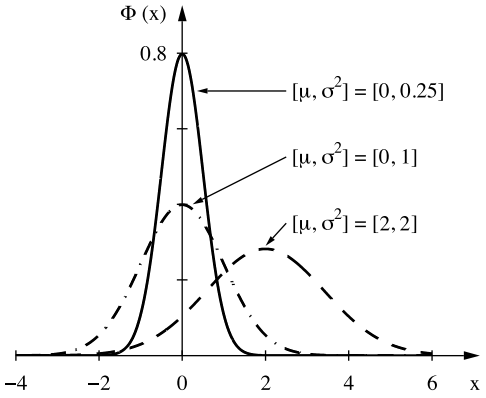


Fig. 16.145: Gaussian distribution

The seed of the generator initializes the algorithm at the simulation 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 μ
The mean value of the distribution.

Standard deviation σ
The standard deviation of the distribution.

Seed
The seed used to initialize the White Noise generator.

Sample time

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s), used for generating random output values. See also "Sample Times" (p. 43).

References

- 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 / Sources

Description



The Winding forms the interface between the electrical and the magnetic domain. A winding of N turns is described with the equations:

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

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

where v and i are voltage and current at the electrical terminals. $\mathcal{F}$ 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 Fig. 16.146 for the two polarities of the winding. Notice the positions of the black dot and the brown arrow in the winding symbol.

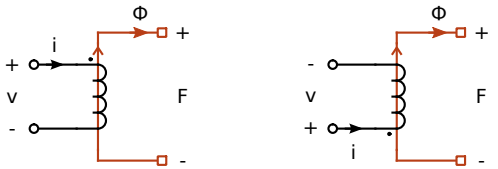


Fig. 16.146: Sign conventions used for electrical and magnetic quantities

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

Electrical / Connectivity

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.



Parameters

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.

↪ *Added in PLECS 4.6.*

Wire Selector

Purpose

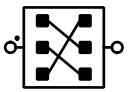
Select or reorder elements from wire bus

Library

Electrical / Connectivity

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.

↪ *Added in PLECS 4.6.*

XY Plot

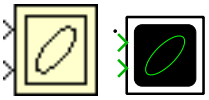
Purpose

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” (p. 129).

Parameters

Title

The name which is displayed above the plot.

Sample time

A scalar specifying the sampling period or a two-element vector specifying the sampling period and offset, in seconds (s), 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 also section “Sample Times” (p. 43).

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 (p. 641), 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 (Ω), 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`.

Forward resistance

The resistance R_f of the forward conducting device, in ohms (Ω). 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

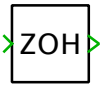
Sample and hold input signal periodically

Library

Control / Discrete

Description

The Zero Order Hold samples the input signal and holds this value at its output for a specified sample time.



Example Model

- See the example model **"Zero Order Hold"**.
- Find it in PLECS under **Help** > **PLECS Documentation** > List of Example Models.

Parameters

Sample time

A scalar specifying the length of the hold time or a two-element vector specifying the hold time length and offset, in seconds (s). See also the sample time type in section "Sample Times" (p. 43).

Probe Signals

Input

The input signal.

Output

The output signal.

17

Simulink Blocks

This chapter lists the contents of the **PLECS Extras** library for PLECS Blockset.

Adapted Blocks

The following blocks are duplicates of PLECS Standalone blocks, adapted to the Simulink framework. Please refer to their documentation:

Discrete Analyses

- Fourier Transform (p. 691)
- Discrete Mean Value (p. 653)
- RMS Value (p. 906)
- Total Harmonic Distortion (p. 1036)

Modulators

- 2-Pulse Generator (p. 510)
- 3-Phase Overmodulation (p. 535)
- 6-Pulse Generator (p. 553)
- Blanking Time (p. 571)
- Blanking Time (3-Level) (p. 572)
- Sawtooth PWM (p. 937)
- Sawtooth PWM (3-Level) (p. 939)
- Symmetrical PWM (p. 988)
- Symmetrical PWM (3-Level) (p. 992)
- Peak Current Controller (p. 859)

Transformations

- Transformation 3ph->RRF (p. 1040)
- Transformation 3ph->SRF (p. 1041)
- Transformation RRF->3ph (p. 1042)
- Transformation RRF->SRF (p. 1043)
- Transformation SRF->3ph (p. 1044)
- Transformation SRF->RRF (p. 1045)

Simulink-Only Blocks

The following blocks are available in PLECS Blockset only.

- AC Sweep
- Impulse Response Analysis (p. 1141)
- Loop Gain Analysis (AC Sweep) (p. 1143)
- Loop Gain Analysis (Multitone) (p. 1144)
- Multitone Analysis (p. 1145)
- Steady-State Analysis (p. 1147)
- Timer (p. 1149)

AC Sweep

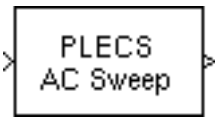
Purpose

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 Sweep" (p. 231).

Parameters

System period length
The period length of the unperturbed system, in seconds (s).

Simulation start time
The simulation start time for the ac sweep, in seconds (s).

Frequency sweep range
A vector containing the lowest and highest perturbation frequency, in hertz (Hz).

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`).

Impulse Response Analysis

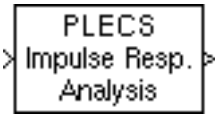
Purpose

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" (p. 232)

Parameters

System period length
The period length of the unperturbed system, in seconds (s).

Simulation start time
The simulation start time for the impulse response analysis, in seconds (s).

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 “Impulse Response Analysis” (p. 262) for a short explanation and example.

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 `plimpulseresponse` 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)

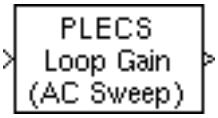
Purpose

Determine loop gain of closed control loop

Library

PLECS Extras / Analysis Tools

Description



The Loop Gain Analysis block enables you to determine the loop 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 Sweep” (p. 231).

> **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 (p. 1138).

Loop Gain Analysis (Multitone)

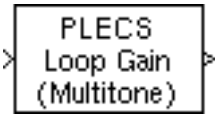
Purpose

Determine loop gain of closed control loop

Library

PLECS Extras / Analysis Tools

Description



The Loop Gain Analysis block enables you to determine the loop 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” (p. 233).

> **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.

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" (p. 233).

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

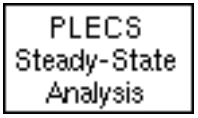
Purpose

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 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” (p. 228).

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; in seconds (s).

Simulation start time

The simulation start time for the steady-state analysis, in seconds (s).

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

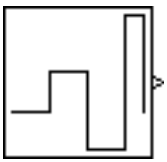
Generate piece-wise constant signal

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, in seconds (s). 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.

plexim

electrical engineering software



Plexim GmbH info@plexim.com www.plexim.com