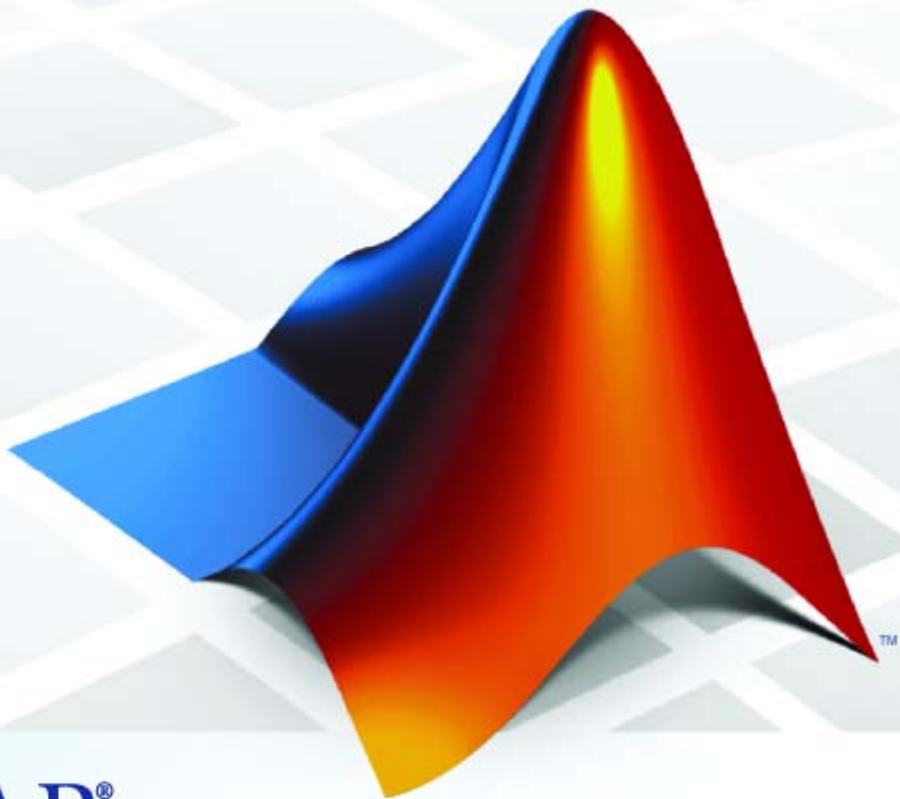


Simulink® Design Verifier™ 1

User's Guide



MATLAB®
& **SIMULINK®**

How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Simulink® Design Verifier™ User's Guide

© COPYRIGHT 2007–2008 by The MathWorks, Inc.

The software described in this document 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 The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

May 2007	Online only	New for Version 1.0 (Release 2007a+)
September 2007	Online only	Revised for Version 1.1 (Release 2007b)
March 2008	Online only	Revised for Version 1.2 (Release 2008a)

Acknowledgment

The Simulink® Design Verifier™ software uses Prover Plug-In® products from Prover® Technology to generate test cases and prove model properties.



Acknowledgment

Acknowledgment

Getting Started

1

Product Overview	1-2
Before You Begin	1-3
What You Need to Know	1-3
Required Products	1-3
Starting the Simulink® Design Verifier™ Software	1-4
Running a Demo Model	1-6
About This Demo	1-6
Opening the Model	1-6
Generating Test Cases	1-7
Exploring the Test Harness	1-9
Interpreting the Simulink® Design Verifier™ Report	1-12
Basic Workflow for Using the Simulink® Design Verifier™ Software	1-17
Learning More	1-18
Next Step	1-18
Product Help	1-18
The MathWorks Online	1-19

Ensuring Compatibility with the Simulink® Design Verifier™ Software

2

Unsupported Simulink® Software Features	2-3
List of Unsupported Simulink® Software Features	2-3
Limitations of Simulink® Block Support	2-3
Unsupported Stateflow® Software Features	2-5
Limitations of Support for the Embedded MATLAB™	
Subset	2-7
List of Unsupported Embedded MATLAB™ Subset	
Features	2-7
Limitations of Embedded MATLAB™ Library Function	
Support	2-8
Limitations of Fixed-Point Support	2-9
Checking Model Compatibility	2-10

Working with Block Replacements

3

About Block Replacements	3-2
Built-In Block Replacements	3-3
Template for Block Replacement Rules	3-6
Creating Custom Block Replacements	3-7
About Custom Block Replacements	3-7
Constructing Replacement Blocks	3-7
Writing Block Replacement Rules	3-10
Executing Block Replacements	3-15

Configuring Block Replacements	3-15
Replacing Blocks in a Model	3-16

Specifying Parameter Configurations

4

About Parameter Configurations	4-2
Template for Parameter Configurations	4-3
Defining Parameter Configurations	4-4
Parameter Configuration Example	4-7
About This Example	4-7
Constructing the Example Model	4-8
Parameterizing the Constant Block	4-11
Specifying a Parameter Configuration	4-12
Analyzing the Example Model	4-13
Simulating the Test Cases	4-16

Configuring Simulink® Design Verifier™ Options

5

Viewing Simulink® Design Verifier™ Options	5-2
Configuring Simulink® Design Verifier™ Options	5-5
Design Verifier Pane	5-5
Block Replacements Pane	5-6
Parameters Pane	5-8
Test Generation Pane	5-9
Property Proving Pane	5-11
Results Pane	5-12
Report Pane	5-14

Generating Test Cases

6

About Test Case Generation	6-2
Basic Workflow for Generating Test Cases	6-3
Generating Test Cases Example	6-4
About This Example	6-4
Constructing the Example Model	6-5
Checking Compatibility of the Example Model	6-6
Configuring Test Generation Options	6-10
Analyzing the Example Model	6-13
Customizing Test Generation	6-21
Reanalyzing the Example Model	6-25

Proving Properties of a Model

7

About Property Proofs	7-2
Basic Workflow for Proving Model Properties	7-3
Proving Model Properties Example	7-4
About This Example	7-4
Constructing the Example Model	7-5
Checking Compatibility of the Example Model	7-6
Instrumenting the Example Model	7-10
Configuring Property Proving Options	7-13
Analyzing the Example Model	7-15
Customizing the Example Proof	7-23
Reanalyzing the Example Model	7-25

Exploring Test Harness Models	8-2
About Test Harness Models	8-2
Anatomy of a Test Harness	8-2
Simulating the Test Harness	8-6
Understanding Simulink® Design Verifier™ Reports ..	8-8
About Simulink® Design Verifier™ Reports	8-8
Front Matter	8-8
Summary Chapter	8-9
Block Replacements Summary Chapter	8-14
Test/Proof Objectives Chapter	8-14
Test Cases / Counterexamples Chapter	8-19
Approximations Chapter	8-22
Examining Simulink® Design Verifier™ Data Files	8-23
About Simulink® Design Verifier™ Data Files	8-23
Anatomy of the sldvData Structure	8-23
Simulating Models with Simulink® Design Verifier™ Data Files	8-28

Analyzing Large Models and Improving Performance

How the Simulink® Design Verifier™ Software Works	A-2
Sources of Model Complexity	A-5
Handling Models with Large Numbers of Inputs	A-6
Reducing Complexity from Floating-Point Operations and Nonlinear Arithmetic	A-7

Partitioning Inputs and Generating Tests	
Incrementally	A-9
Handling Models with Large State Spaces	A-11
Handling Problems with Counters and Timers	A-12
Strategies for Proving Properties of Large Models	A-13

Function Reference

9

Block Reference

10

Configuration Parameters

11

Design Verifier Pane	11-2
Design Verifier Pane Overview	11-3
Mode	11-4
Maximum analysis time	11-5
Display unsatisfiable test objectives	11-6
Output directory	11-7
Make output file names unique by adding a suffix	11-8
Design Verifier Pane: Block Replacements	11-9
Block Replacements Pane Overview	11-10
Apply block replacements	11-11
List of block replacement rules	11-12
File path of the output model	11-13
Design Verifier Pane: Parameters	11-14

Parameters Pane Overview	11-15
Apply parameters	11-16
Parameter configuration file	11-17
Design Verifier Pane: Test Generation	11-18
Test Generation Pane Overview	11-19
Model coverage objectives	11-20
Test conditions	11-21
Test objectives	11-22
Maximum test case steps	11-23
Test suite optimization	11-24
Design Verifier Pane: Property Proving	11-26
Property Proving Pane Overview	11-27
Assertion blocks	11-28
Proof assumptions	11-29
Strategy	11-30
Maximum violation steps	11-31
Design Verifier Pane: Results	11-32
Results Pane Overview	11-33
Save test harness as model	11-34
Harness model file name	11-35
Save test data to file	11-36
Data file name	11-37
Include expected output values	11-38
Randomize data that does not affect outcome	11-40
Design Verifier Pane: Report	11-42
Report Pane Overview	11-43
Generate report of the results	11-44
Report file name	11-45
Include screen shots and plots	11-46
Display report	11-47
Parameter Command-Line Information Summary	11-48

Simulink® Block Support

12

Embedded MATLAB™ Subset Support

13

Glossary

Examples

B

Working with Block Replacements	B-2
Specifying Parameter Configurations	B-2
Generating Test Cases	B-2
Proving Properties of a Model	B-2

Index

Getting Started

Product Overview (p. 1-2)

Overview of the Simulink® Design Verifier™ software

Before You Begin (p. 1-3)

Other products you need or might want to use with the Simulink Design Verifier software

Starting the Simulink® Design Verifier™ Software (p. 1-4)

Accessing the Simulink Design Verifier library

Running a Demo Model (p. 1-6)

Analyzing a simple demo model with the Simulink Design Verifier software

Basic Workflow for Using the Simulink® Design Verifier™ Software (p. 1-17)

Overview of the basic workflow

Learning More (p. 1-18)

Where to find more information

Product Overview

The Simulink® Design Verifier™ software extends the Simulink® product by performing exhaustive formal analyses of your models to confirm that they behave correctly.

The Simulink Design Verifier software allows you to perform the following tasks:

- Generate test cases that achieve model coverage and custom objectives you specify in a model.
- Prove properties that you specify in a model, and identify examples of any property violations.
- Detect unreachable design elements in a model, such as inaccessible subsystems, illegal switch conditions, and unachievable states.
- Produce detailed reports regarding test case generation and property proofs.

Before You Begin

In this section...
“What You Need to Know” on page 1-3
“Required Products” on page 1-3

What You Need to Know

Getting started with the Simulink® Design Verifier™ software requires that you have some experience using model coverage, as well as building and running Simulink® models.

To learn more about these topics, see the following:

- “Using Model Coverage” in the *Simulink® Verification and Validation™ User’s Guide*
- *Simulink Getting Started Guide* and *Using Simulink*

Required Products

You must have the following products installed to use the Simulink Design Verifier software:

- MATLAB®
- Simulink
- Simulink Verification and Validation

If you want to use the Simulink Design Verifier software with Stateflow® charts, you must have the following software product:

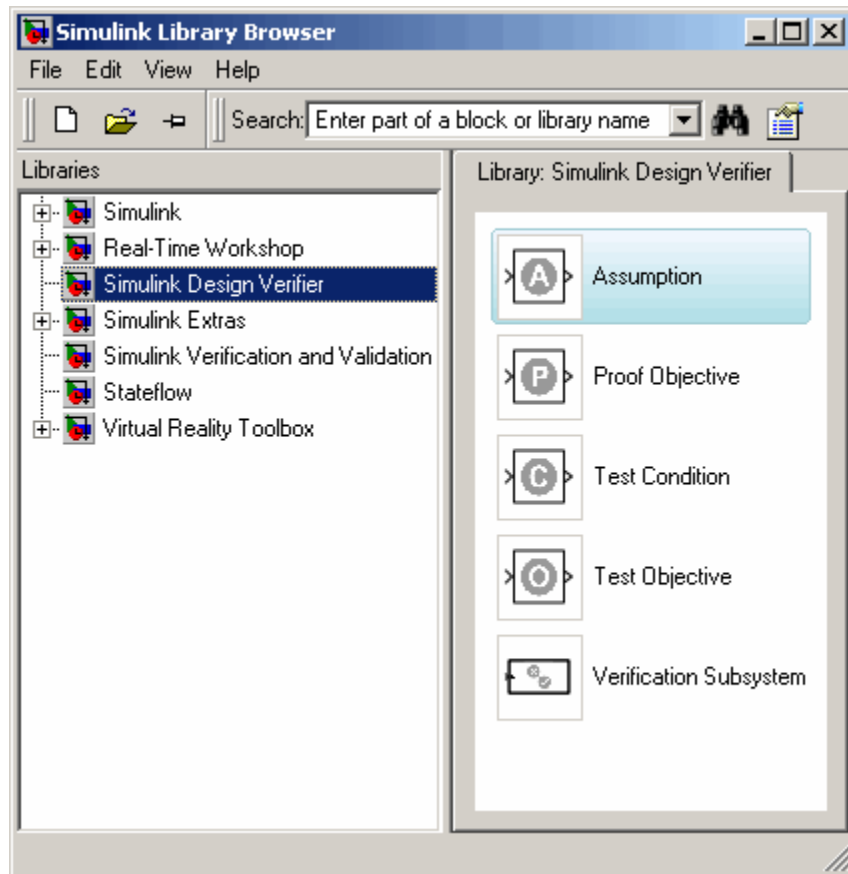
- Stateflow

Starting the Simulink® Design Verifier™ Software

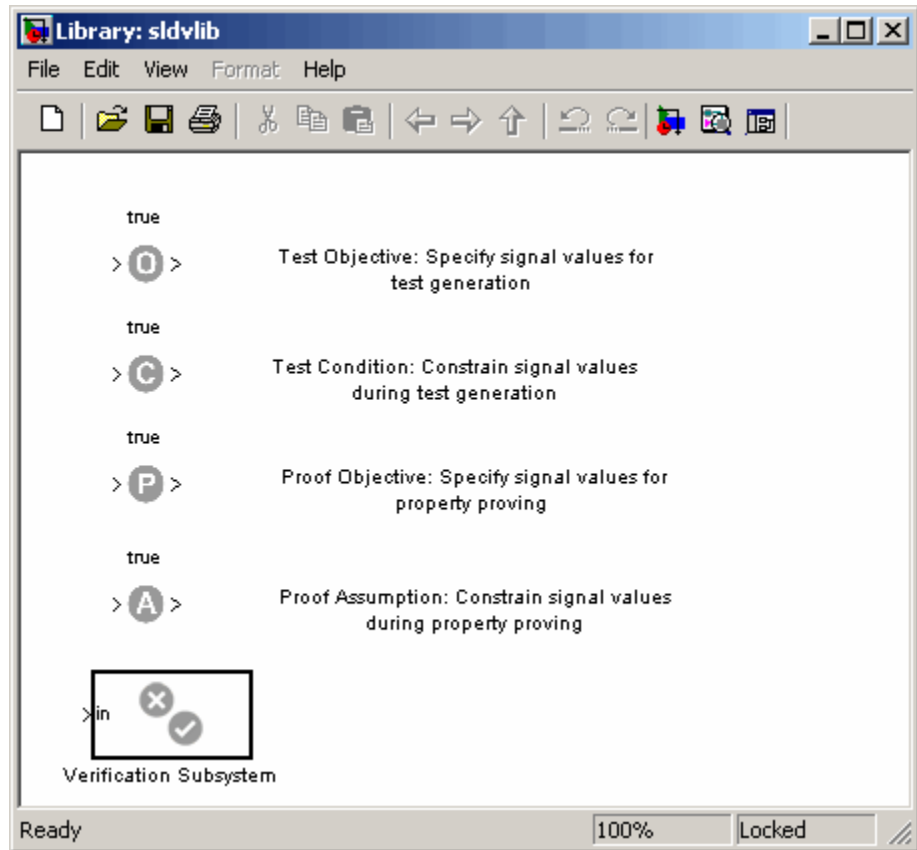
The Simulink® Design Verifier™ software is part of your MATLAB® installation.

To open the Simulink Design Verifier block library:

- Type `simulink` at the MATLAB prompt to display the Simulink® Library Browser, and then select the **Simulink Design Verifier** entry in the contents tree.



- Alternatively, type `sldvlib` at the MATLAB prompt to display the Simulink Design Verifier library.



Running a Demo Model

In this section...
“About This Demo” on page 1-6
“Opening the Model” on page 1-6
“Generating Test Cases” on page 1-7
“Exploring the Test Harness” on page 1-9
“Interpreting the Simulink® Design Verifier™ Report” on page 1-12

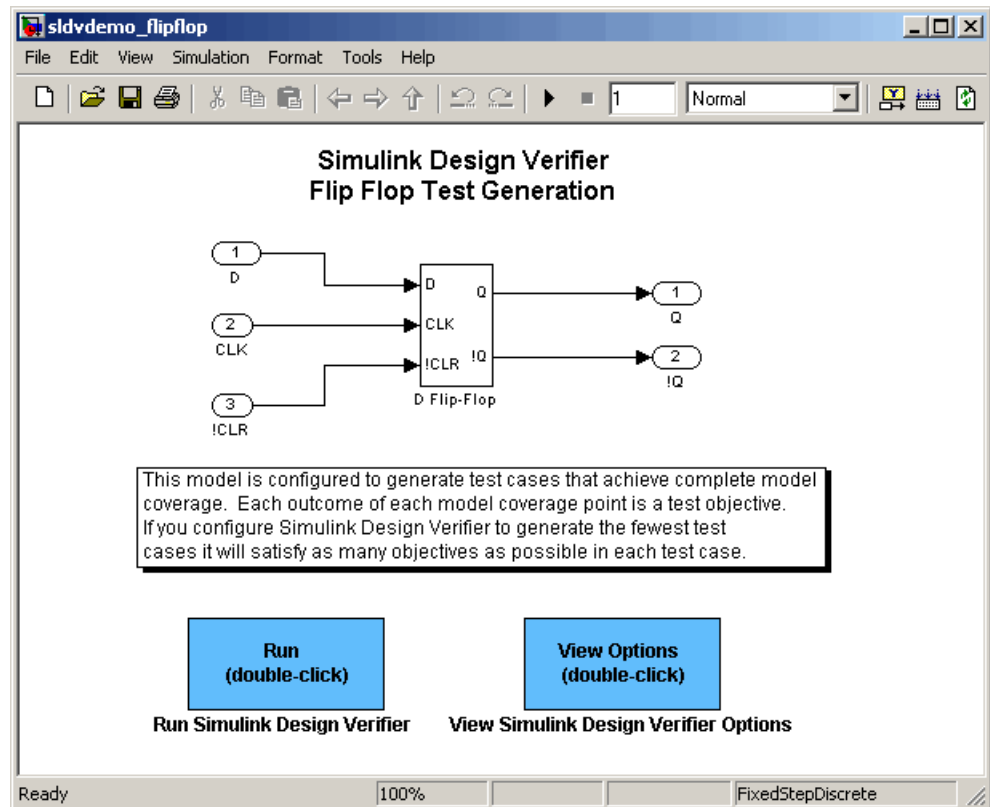
About This Demo

The sections that follow describe a demo model, Flip Flop Test Generation, which illustrates how the Simulink® Design Verifier™ software can be used to generate test cases that achieve complete model coverage. This demo will help you understand how to analyze models with the Simulink Design Verifier software and interpret the results.

Opening the Model

To open the Flip Flop Test Generation model, enter `sldvdemo_flipflop` at the MATLAB® prompt.

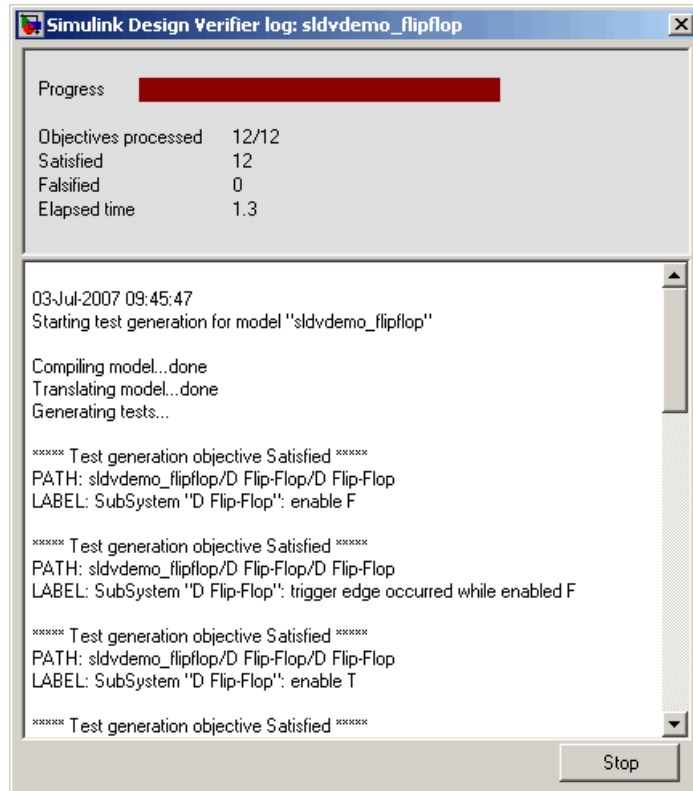
The Flip Flop Test Generation model appears.



Generating Test Cases

To generate test cases for the Flip Flop Test Generation model, in the model window double-click the block labeled **Run**.

The Simulink Design Verifier software begins analyzing the model to generate test cases. During its analysis, the software displays the following log window:



The log window updates you on the progress of the Simulink Design Verifier software as it analyzes the model. Also, the log window includes a **Stop** button that you can click to terminate the analysis at anytime.

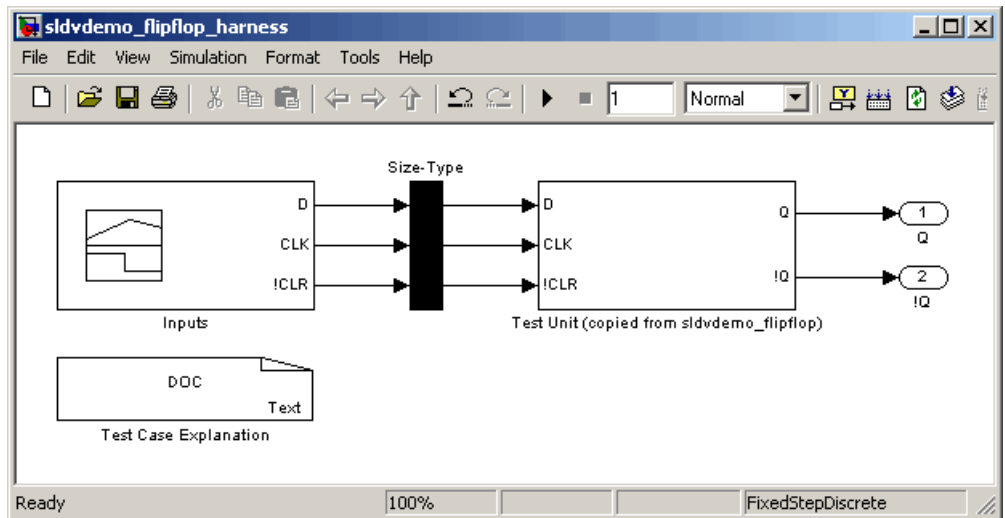
When the Simulink Design Verifier software completes its analysis, it displays the following items:

- Test harness model named `sldvdemo_flipflop_harness.mdl`
- Report named `sldvdemo_flipflop_report.html`

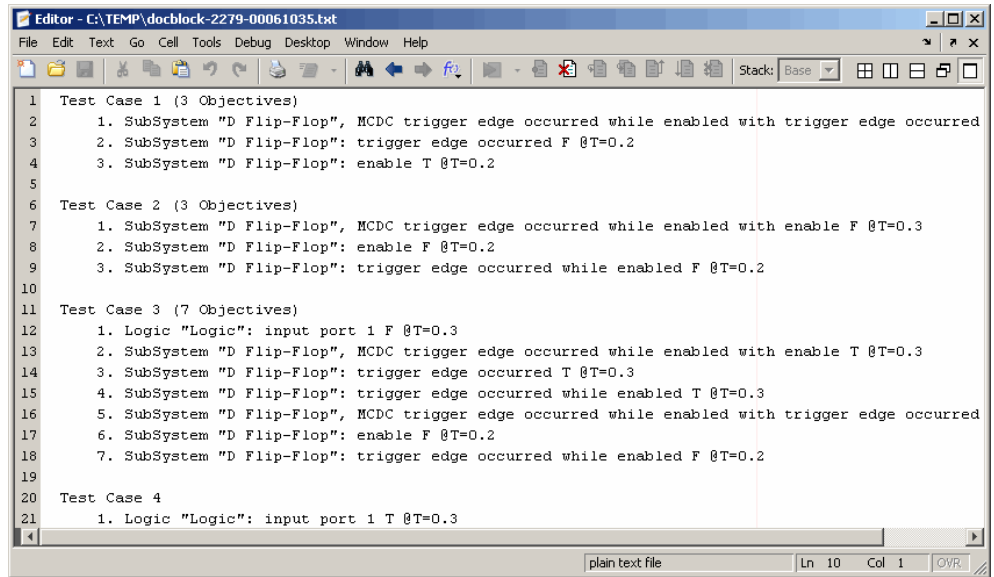
The sections that follow describe each of these items.

Exploring the Test Harness

The Simulink Design Verifier software creates a test harness model when it completes its analysis. The test harness for the Flip Flop Test Generation model appears as follows:



- 1 The block labeled Test Case Explanation is a DocBlock that documents the test cases the Simulink Design Verifier software generated. Double-click the Test Case Explanation block to view a description of each test case in terms of the objectives that it satisfies.



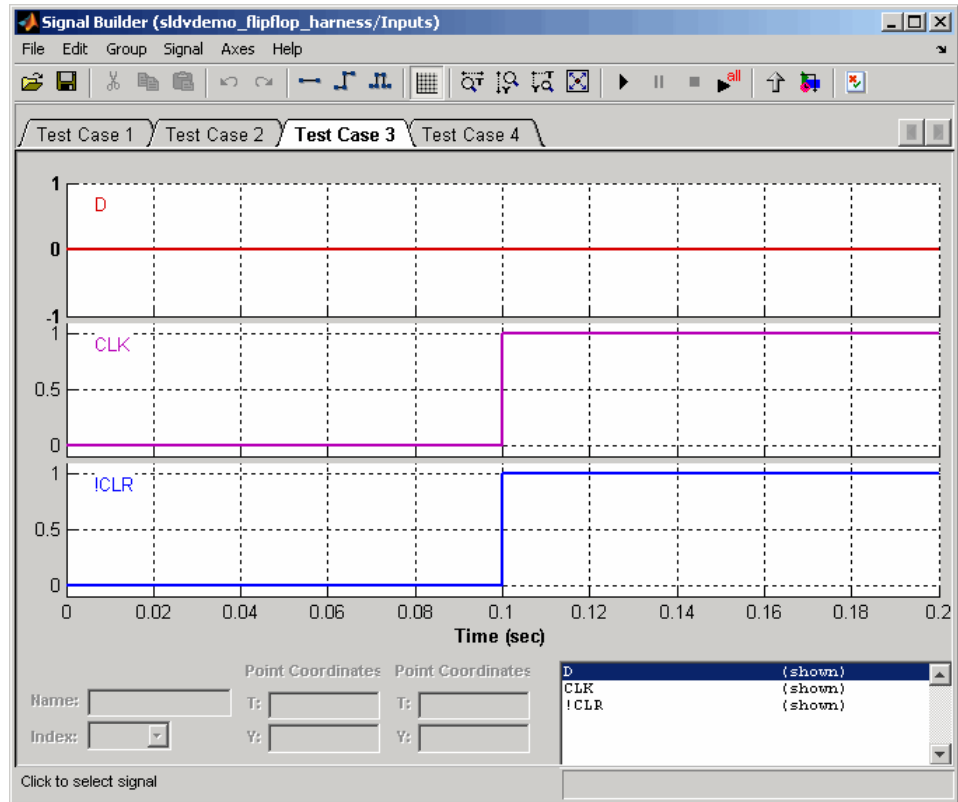
The screenshot shows a text editor window titled "Editor - C:\TEMP\docblock-2279-00061035.txt". The window contains a list of test case objectives for a "D Flip-Flop" subsystem. The objectives are organized into four test cases:

```
1 Test Case 1 (3 Objectives)
2   1. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with trigger edge occurred
3   2. SubSystem "D Flip-Flop": trigger edge occurred F @T=0.2
4   3. SubSystem "D Flip-Flop": enable T @T=0.2
5
6 Test Case 2 (3 Objectives)
7   1. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with enable F @T=0.3
8   2. SubSystem "D Flip-Flop": enable F @T=0.2
9   3. SubSystem "D Flip-Flop": trigger edge occurred while enabled F @T=0.2
10
11 Test Case 3 (7 Objectives)
12   1. Logic "Logic": input port 1 F @T=0.3
13   2. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with enable T @T=0.3
14   3. SubSystem "D Flip-Flop": trigger edge occurred T @T=0.3
15   4. SubSystem "D Flip-Flop": trigger edge occurred while enabled T @T=0.3
16   5. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with trigger edge occurred
17   6. SubSystem "D Flip-Flop": enable F @T=0.2
18   7. SubSystem "D Flip-Flop": trigger edge occurred while enabled F @T=0.2
19
20 Test Case 4
21   1. Logic "Logic": input port 1 T @T=0.3
```

The status bar at the bottom of the window indicates "plain text file", "Ln 10", "Col 1", and "OVR".

- 2** The block labeled Test Unit is a Subsystem block that contains a copy of the original model the Simulink Design Verifier software analyzed. Double-click the Test Unit block to view its contents and confirm that it is a copy of the Flip Flop Test Generation model.
- 3** The block labeled Inputs is a Signal Builder block that contains the test case signals the Simulink Design Verifier software generated. Double-click the Inputs block to open the Signal Builder dialog box and view the test case signals.
- 4** Look at the signal values for a particular test case. In the Signal Builder dialog box, select the tab associated with a test case. For example, select Test Case 3.


The Signal Builder dialog box displays the signal values for Test Case 3.



In Test Case 3 at 0.1 seconds,

- The D signal remains 0.
- The CLK signal transitions from 0 to 1.
- The !CLR signal transitions from 0 to 1.

This group of signals achieves the test objectives described in the Test Case Explanation block.

- 5 To confirm that the Simulink Design Verifier software achieved complete model coverage, simulate the test harness using all the test cases. In the Signal Builder dialog box, click the **Run all** button .

The report displays its Summary chapter, as shown here.

Chapter 1. Summary

Input Model

File:	C:\matlab\toolbox\sldv\sldvdemos\sldvdemo_flipflop.mdl
Version:	1.14
Time Stamp:	Fri Jun 29 18:44:32 2007
Author:	

Analysis Information

Design Verifier Version:	1.1
Total Analysis Time:	1.26 secs
Status:	Completed normally
Approximations:	0
Objectives Satisfied:	12
Objectives Satisfied - No Test Case:	0
Objectives Proven Unsatisfiable:	0
Objectives Undecided:	0
Objectives Producing Errors:	0

Output Files

Harness model:	C:\sldv_output\sldvdemo_flipflop\sldvdemo_flipflop_harness.mdl
Data file:	C:\sldv_output\sldvdemo_flipflop\sldvdemo_flipflop_sldvdata.mat
Report:	C:\sldv_output\sldvdemo_flipflop\sldvdemo_flipflop_report.html

The Summary chapter provides an overview of the Simulink Design Verifier analysis. For instance, the chapter includes information about the model it analyzed, the results it obtained, the files it generated, and the options it used.

2 Under **Analysis Information**, click **Objectives Satisfied**.

The report displays the following table under its Test Objectives chapter:

Chapter 2. Test Objectives

Table of Contents

[Status](#)
[sldvdemo_flipflop](#)
[D Flip-Flop](#)

Status

Table 2.1. Objectives Satisfied

#:	Type	Model Item	Description
1	Condition	D Flip-Flop	SubSystem "D Flip-Flop": enable T
2	Condition	D Flip-Flop	SubSystem "D Flip-Flop": enable F
3	Condition	D Flip-Flop	SubSystem "D Flip-Flop": trigger edge occurred T
4	Condition	D Flip-Flop	SubSystem "D Flip-Flop": trigger edge occurred F
5	Decision	D Flip-Flop	SubSystem "D Flip-Flop": trigger edge occurred while enabled F
6	Decision	D Flip-Flop	SubSystem "D Flip-Flop": trigger edge occurred while enabled T
7	Mcdc	D Flip-Flop	SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with enable T
8	Mcdc	D Flip-Flop	SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with enable F
9	Mcdc	D Flip-Flop	SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with trigger edge occurred T
10	Mcdc	D Flip-Flop	SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with trigger edge occurred F
11	Condition	Logic	Logic "Logic": input port 1 T
12	Condition	Logic	Logic "Logic": input port 1 F

The **Objectives Satisfied** table lists model coverage objectives that the Simulink Design Verifier software satisfied. That is, the software generated test cases that achieve each of the model coverage objectives shown here.

- 3 Under the # column of the **Objectives Satisfied** table, click objective 5.

The report displays the following table under its Test Objectives chapter:

sldvdemo_flipflop

Objectives of: D Flip-Flop

#:	Status	Test Cases	Description
1	Satisfied	TC 1	T
2	Satisfied	TC 3	F
3	Satisfied	TC 3	T
4	Satisfied	TC 1	F
5	Satisfied	TC 3	F
6	Satisfied	TC 3	T
7	Satisfied	TC 3	trigger edge occurred while enabled with enable T
8	Satisfied	TC 2	trigger edge occurred while enabled with enable F
9	Satisfied	TC 3	trigger edge occurred while enabled with trigger edge occurred T
10	Satisfied	TC 1	trigger edge occurred while enabled with trigger edge occurred F

This table lists all the model coverage objectives associated with the D Flip-Flop subsystem in the demo model. It displays a description and status for each objective, as well as the test case that achieves the objective. Objective 5 applies only to the D Flip-Flop subsystem, so it is listed here.

- 4 Under the **Test Cases** column of the table, click [TC 3](#).

The report displays its Test Case 3 section under the Test Cases chapter, as shown here.

Test Case 3

Summary

Length: 0.2 Seconds (3 sample periods)

Objective Count: 7

Objectives Reached At:

Step	Time	Objectives
1	0	2 5
2	0.1	3 6 7 9 12

Generated Input Data.

Time 0	0.1
Step 1	2
D 0	-
CLK 0	1
ICLR 0	1

This section provides details about a test case that the Simulink Design Verifier software generated. For example, Test Case 3 satisfies seven model coverage objectives. In this test case, the following signal values achieve objectives 2, 3, 5, 6, 7, 9, and 12:

- The D signal’s initial value is 0 at 0 seconds.
- The CLK signal transitions from 0 to 1 at 0.1 seconds.
- The !CLR signal transitions from 0 to 1 at 0.1 seconds.

This information matches what you see in the test harness model. Specifically, the Inputs block depicts identical signal values for Test Case 3, and the Test Case Explanation block lists seven objectives that Test Case 3 achieves (see “Exploring the Test Harness” on page 1-9).

Basic Workflow for Using the Simulink® Design Verifier™ Software

The Simulink® Design Verifier™ User’s Guide on page 1 is organized on the basis of workflow that you follow when generating tests for your model or proving its properties. This workflow is described in the following steps, which cite locations in the documentation that you can refer to for more information:

Step	Action	See...
1	Check the compatibility of your model.	Chapter 2, “Ensuring Compatibility with the Simulink® Design Verifier™ Software”
2	Optionally, prepare your model for analysis.	Chapter 3, “Working with Block Replacements” Chapter 4, “Specifying Parameter Configurations”
3	Set Simulink® Design Verifier™ options.	Chapter 5, “Configuring Simulink® Design Verifier™ Options”
4	Generate test cases for your model or prove its properties.	Chapter 6, “Generating Test Cases” Chapter 7, “Proving Properties of a Model”
5	Interpret the results.	Chapter 8, “Reviewing the Results”

Learning More

In this section...
“Next Step” on page 1-18
“Product Help” on page 1-18
“The MathWorks Online” on page 1-19


Next Step

To begin learning how to use the Simulink® Design Verifier™ software, see Chapter 2, “Ensuring Compatibility with the Simulink® Design Verifier™ Software”. Also see the following topics to continue your exploration of the software:

For...	See...
Exercise that walks you through the process of generating test cases for a model	“Generating Test Cases Example” on page 6-4
Exercise that walks you through the process of proving a model property	“Proving Model Properties Example” on page 7-4

Product Help

More information is available with your product installation. In the

MATLAB® desktop, click  for help, and then click the product name in the **Contents** pane.

For...	See...
List of blocks	Blocks — Alphabetical List
Tutorials	Examples in Documentation
More product demonstrations	Simulink Design Verifier Demos
What’s new in this product	Release Notes

The MathWorks Online

Point your internet browser to the MathWorks Web site for additional information and support at

<http://www.mathworks.com/products/slidesignverifier/>

Ensuring Compatibility with the Simulink® Design Verifier™ Software

The Simulink® Design Verifier™ software supports a broad range of Simulink® and Stateflow® software features. However, there are features that the product does not support. Therefore, you must avoid using particular features in models that you plan to analyze with the Simulink Design Verifier software. The following sections identify the unsupported features and describe how to check whether your model is compatible for use with the Simulink Design Verifier software.

Unsupported Simulink® Software
Features (p. 2-3)

Lists Simulink software features that the Simulink Design Verifier software does not support.

Unsupported Stateflow® Software
Features (p. 2-5)

Lists the Stateflow software features that the Simulink Design Verifier software does not support.

Limitations of Support for the
Embedded MATLAB™ Subset
(p. 2-7)

Lists limitations associated with Simulink Design Verifier software support for the Embedded MATLAB™ subset.

Limitations of Fixed-Point Support
(p. 2-9)

Lists limitations associated with Simulink Design Verifier software support for fixed-point data types.

Checking Model Compatibility
(p. 2-10)

Describes how to check whether your model is compatible with the Simulink Design Verifier software.

Unsupported Simulink® Software Features

In this section...

“List of Unsupported Simulink® Software Features” on page 2-3

“Limitations of Simulink® Block Support” on page 2-3

List of Unsupported Simulink® Software Features

The Simulink® Design Verifier™ software does not support the following Simulink® software features. Avoid using these unsupported features in models that you analyze with the Simulink Design Verifier software.

Feature Not Supported	Remarks
Variable-step solvers	The Simulink Design Verifier software supports only fixed-step solvers (see “Choosing a Fixed-Step Solver” in <i>Using Simulink</i>).
Complex signals	The Simulink Design Verifier software supports only real signals (for contrast, see “Complex Signals” in <i>Using Simulink</i>).
Nonvirtual buses	The Simulink Design Verifier software supports only virtual buses (see “Virtual and Nonvirtual Buses” in <i>Using Simulink</i>).
Nonzero start times	Although Simulink allows you to specify a nonzero simulation start time (see “Specifying a Simulation Start and Stop Time” in <i>Using Simulink</i>), the Simulink Design Verifier software generates signal data that begins only at zero. If your model specifies a nonzero start time, the Simulink Design Verifier software ignores it and uses zero instead.

Limitations of Simulink® Block Support

The Simulink Design Verifier software provides various levels of support for Simulink blocks. That is, the software either fully or partially supports particular blocks, while it does not support others. Refrain from using

unsupported Simulink blocks in models that you analyze with the Simulink Design Verifier software. Similarly, specify only the block parameters that the Simulink Design Verifier software recognizes for blocks that it partially supports. See Chapter 12, “Simulink® Block Support” for a list of Simulink blocks and details regarding whether the Simulink Design Verifier software provides support.

Unsupported Stateflow® Software Features

The Simulink® Design Verifier™ software does not support the following Stateflow® software features. Avoid using these unsupported features in models that you analyze with the Simulink Design Verifier software.

Feature Not Supported	Remarks
m1 namespace operator, m1 function, m1 expressions	The Simulink Design Verifier software does not support calls to MATLAB® functions or access to MATLAB workspace variables, which the Stateflow software allows (see “Using MATLAB Functions and Data in Actions” in the <i>Stateflow and Stateflow® Coder™ User’s Guide</i>).
C math functions	The Simulink Design Verifier software supports calls to the following C math functions: <code>abs</code> , <code>ceil</code> , <code>fabs</code> , <code>floor</code> , <code>fmod</code> , <code>labs</code> , <code>ldexp</code> , and <code>pow</code> (only for an integer exponent). However, the Simulink Design Verifier software does not support calls to other C math functions that the Stateflow software allows (see “Calling C Functions in Actions” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).
Recursion	The Simulink Design Verifier software does not support recursive functions, which the Stateflow software allows you to implement using graphical functions (see “Using Graphical Functions to Extend Actions” in the <i>Stateflow and Stateflow Coder User’s Guide</i>). Also, the Simulink Design Verifier software does not support recursion that the Stateflow software allows you to implement using a combination of event broadcasts and function calls.
Custom C or C++ code	The Simulink Design Verifier software does not support custom C or C++ code, which the Stateflow software allows (see “Building Targets” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).

Feature Not Supported	Remarks
Machine-parented data and events	The Simulink Design Verifier software does not support machine-parented data and events (i.e., defined at the level of the Stateflow machine in the Stateflow hierarchy), which the Stateflow software allows (see “Defining Data” and “Defining Events” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).
Stateflow structures	The Simulink Design Verifier software does not support Stateflow structures, which the Stateflow software allows for implementing bus signals (see “Working with Structures and Bus Signals in Stateflow Charts” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).
Absolute-time temporal logic	The Simulink Design Verifier software does not support absolute-time temporal logic, which the Stateflow software allows (see “Operators for Absolute-Time Temporal Logic” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).

Limitations of Support for the Embedded MATLAB™ Subset

In this section...
“List of Unsupported Embedded MATLAB™ Subset Features” on page 2-7
“Limitations of Embedded MATLAB™ Library Function Support” on page 2-8

List of Unsupported Embedded MATLAB™ Subset Features

The Simulink® Design Verifier™ software does not support the following features of the Embedded MATLAB™ Function block in the Simulink® software and Embedded MATLAB functions in the Stateflow® software. Avoid using these unsupported features in models that you analyze with the Simulink Design Verifier software.

Feature Not Supported	Remarks
Complex numbers	The Simulink Design Verifier software supports only real numbers. However, the Embedded MATLAB subset also supports complex numbers (see “Working with Complex Numbers” in the <i>Embedded MATLAB™ User’s Guide</i>).
Structures	The Simulink Design Verifier software does not support structures, which the Embedded MATLAB subset allows (see “Using Structures” in the <i>Embedded MATLAB™ User’s Guide</i>).
Characters	The Simulink Design Verifier software does not support characters, which the Embedded MATLAB subset allows (see “Working with Characters” in the <i>Embedded MATLAB™ User’s Guide</i>).

Feature Not Supported	Remarks
C functions	The Simulink Design Verifier software does not support calls to external C functions, which the Embedded MATLAB subset allows (see “Calling C Functions from the Embedded MATLAB Subset” in the <i>Embedded MATLAB™ User’s Guide</i>).
Extrinsic functions	The Simulink Design Verifier software supports extrinsic functions only when they do not affect the output of an Embedded MATLAB function. See “Calling MATLAB® Functions” in the <i>Embedded MATLAB™ User’s Guide</i> for more information.

Limitations of Embedded MATLAB™ Library Function Support

The Simulink Design Verifier software provides various levels of support for Embedded MATLAB library functions. That is, the software either fully or partially supports particular functions, while it does not support others. Refrain from using unsupported Embedded MATLAB library functions in models that you analyze with the Simulink Design Verifier software. See Chapter 13, “Embedded MATLAB™ Subset Support” for a list of the Embedded MATLAB library functions for which the Simulink Design Verifier software provides no support or limited support.

Limitations of Fixed-Point Support

The Simulink® Design Verifier™ software supports fixed-point data types in models that it analyzes. However, the following limitations apply:

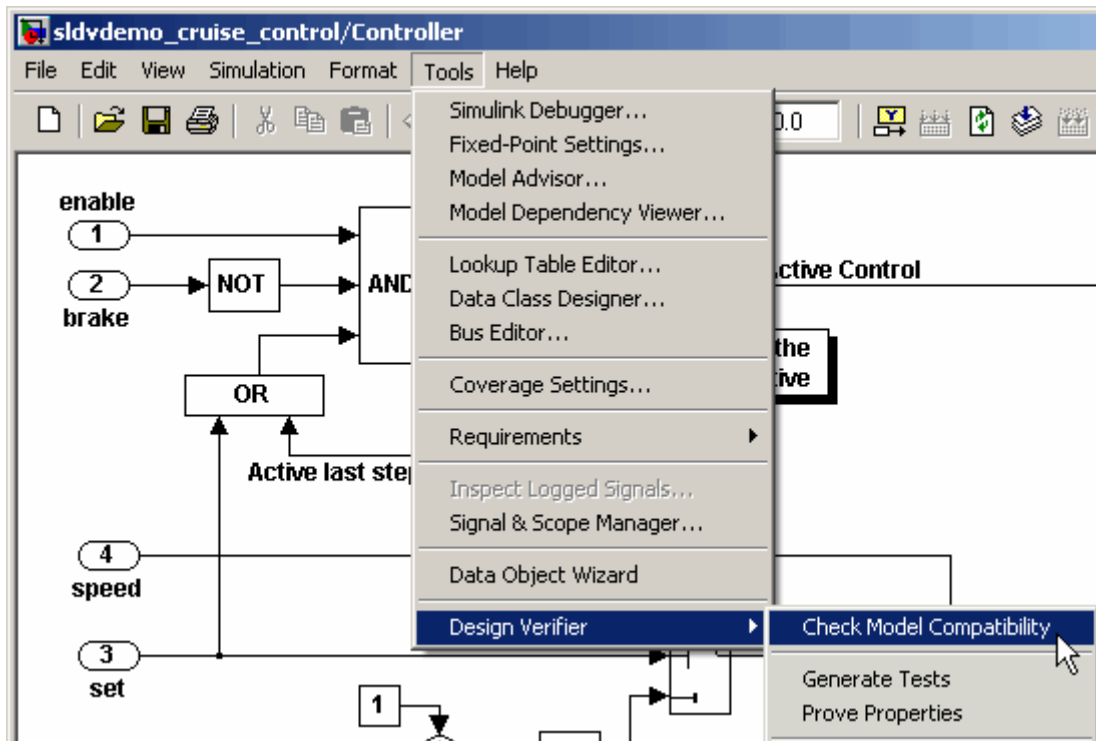
- The Simulink Design Verifier software supports fixed-point data types whose word size is 32 bits or less.
- The following Simulink Design Verifier blocks cannot accept or emit fixed-point signals, nor can you specify fixed-point data types for their block parameters:
 - Proof Assumption
 - Proof Objective
 - Test Condition
 - Test Objective

Likewise, the corresponding Simulink Design Verifier functions for use in Stateflow® charts do not support fixed-point data types:

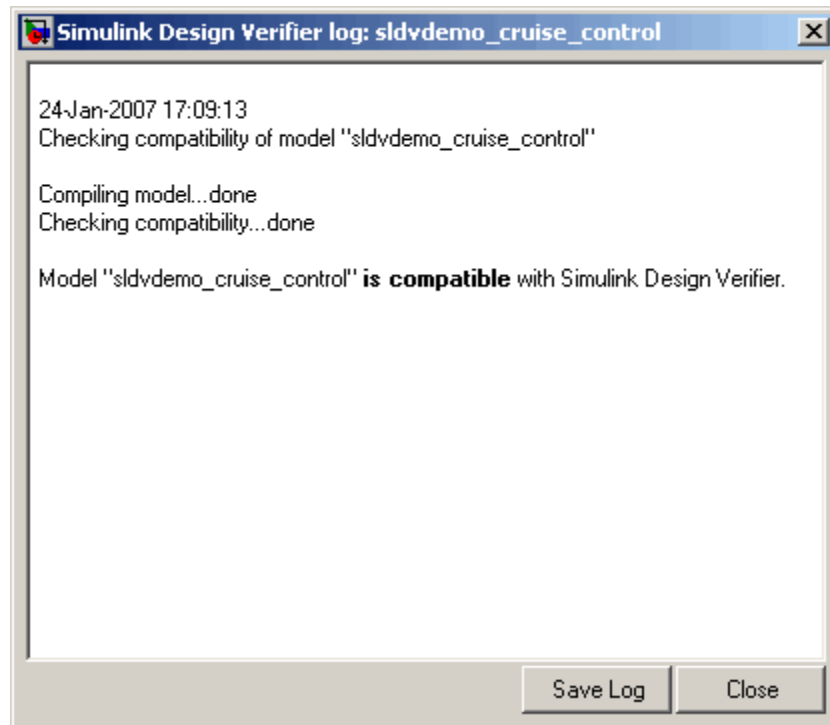
- `dv.assume()`
- `dv.prove()`
- `dv.condition()`
- `dv.test()`
- Parameter configurations do not support fixed-point data types (see Chapter 4, “Specifying Parameter Configurations”).

Checking Model Compatibility

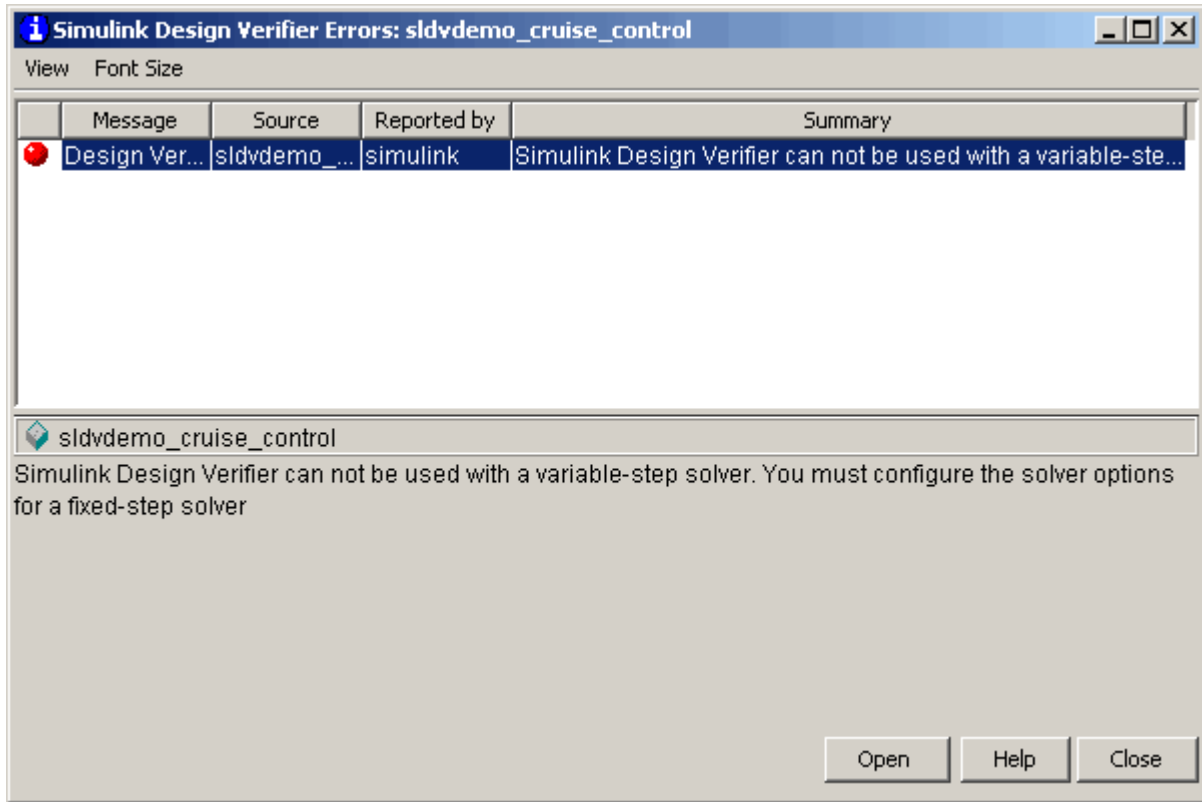
The Simulink® Design Verifier™ software provides a mechanism that checks whether your model is compatible for analysis. To check the compatibility of your model, from the **Tools** menu of your Simulink® model, select **Design Verifier > Check Model Compatibility**.



The Simulink Design Verifier software displays a log window that confirms whether your model is compatible for analysis.



Otherwise, the Simulink Design Verifier software alerts you to any incompatibilities that it identifies in your model. For example, suppose that the preceding model specifies the use of an incompatible feature, such as a variable-step solver. When checking the compatibility of your model in this case, the Simulink Design Verifier software displays incompatibility errors in the Simulation Diagnostics Viewer (see “Simulation Diagnostics Viewer” in *Using Simulink*).



Using the information that the Simulation Diagnostics Viewer displays, you can determine the cause of an incompatibility and correct it.

Note The Simulink Design Verifier software checks the compatibility of a model incrementally. When it detects an incompatibility, it displays an error message and stops the check without completing all the steps. If you receive an error, correct the problem and then recheck whether your model is compatible.

Alternatively, you can use the `sldvcompat` function to run the compatibility checker programmatically at the command line or in an M-file program. See `sldvcompat` in the Chapter 9, “Function Reference” for more information.

Working with Block Replacements

The Simulink® Design Verifier™ software allows you to define rules that replace blocks automatically in your model. For example, you can work around an incompatibility by creating a rule that replaces an unsupported Simulink® block in your model with a supported block that is functionally equivalent. Or you can customize blocks for analysis by creating a rule that adds constraints or objectives to particular blocks in your model. The following sections introduce block replacements and illustrate a process for writing block replacement rules.

About Block Replacements (p. 3-2)	Brief overview of block replacements.
Built-In Block Replacements (p. 3-3)	Describes the factory default block replacement rules and library.
Template for Block Replacement Rules (p. 3-6)	Introduces a template for creating custom block replacement rules.
Creating Custom Block Replacements (p. 3-7)	Outlines a process for creating custom block replacements.
Executing Block Replacements (p. 3-15)	Describes how to execute block replacements.

About Block Replacements

The Simulink® Design Verifier™ software can perform block replacements automatically in a model. That is, it can replace instances of a particular block in your model with an entirely different block. When performing block replacements, the software copies your model and replaces blocks in the copy, leaving your original model unaltered. In this way, you can easily customize a model for analysis with the Simulink Design Verifier software.

The Simulink Design Verifier software replaces blocks automatically in a model using

- Libraries of replacement blocks
- Rules that define which blocks to replace and under what conditions

Block replacements are extensible, allowing you to define your own libraries of replacement blocks and custom block replacement rules. This capability is beneficial if you need to

- Work around an incompatibility, such as the presence of unsupported blocks in your model.
- Customize a block for analysis, such as adding constraints to its input signals or objectives to its output signals.

Built-In Block Replacements

The Simulink® Design Verifier™ software provides a set of block replacement rules and a corresponding library of replacement blocks. These built-in block replacements are useful when analyzing models with the Simulink Design Verifier software. Moreover, they serve as examples that you can examine to learn how to create your own block replacements.

The following table lists the factory default block replacement rules, available in the `matlabroot\toolbox\sldv\sldv\private` directory.

File Name	Description
blkrep_rule_lookup_normal.m blkrep_rule_lookup_configss.m	A rule that replaces Lookup Table blocks with an implementation that includes test objectives for each breakpoint and interval specified by the Vector of input values parameter.
blkrep_rule_lookup2D_normal.m blkrep_rule_lookup2D_configss.m	A rule that adds Test Condition/Proof Assumption blocks to the input ports of Lookup Table (2-D) blocks. Each Test Condition/Proof Assumption block constrains signal values to the interval specified by the corresponding breakpoint vector.
blkrep_rule_mpswitch2_normal.m blkrep_rule_mpswitch2_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter specifies 2. The Test Condition/Proof Assumption block constrains signal values to the interval [1, 2] (or [0, 1] if the block uses zero-based indexing).
blkrep_rule_mpswitch3_normal.m blkrep_rule_mpswitch3_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter specifies 3. The Test Condition/Proof Assumption block constrains signal values to the interval [1, 3] (or [0, 2] if the block uses zero-based indexing).

File Name	Description
blkrep_rule_mpswitch4_normal.m blkrep_rule_mpswitch4_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter specifies 4. The Test Condition/Proof Assumption block constrains signal values to the interval [1, 4] (or [0, 3] if the block uses zero-based indexing).
blkrep_rule_mpswitch5_normal.m blkrep_rule_mpswitch5_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter specifies 5. The Test Condition/Proof Assumption block constrains signal values to the interval [1, 5] (or [0, 4] if the block uses zero-based indexing).
blkrep_rule_switch_normal.m blkrep_rule_switch_configss.m	A rule that replaces Switch blocks with an implementation that includes test objectives, requiring each switch position to be exercised when the values of the first and third input ports differ.
blkrep_rule_selector IndexVecPort_normal.m blkrep_rule_selector IndexVecPort_configss.m	A rule that adds a Test Condition/Proof Assumption block to the index port of Selector blocks whose Index Option parameter specifies Index vector (port). The Test Condition/Proof Assumption block constrains signal values to an interval whose endpoints are derived from the values of the Selector block's Input port size and Index mode parameters.
blkrep_rule_selector StartingIdxPort_normal.m blkrep_rule_selector StartingIdxPort_configss.m	A rule that adds a Test Condition/Proof Assumption block to the index port of Selector blocks whose Index Option parameter specifies Starting index (port). The Test Condition/Proof Assumption block constrains signal values to an interval whose endpoints are derived from the values of the Selector block's Input port size, Output size, and Index mode parameters.

The library of replacement blocks that corresponds to the factory default rules resides at

```
matlabroot/toolbox/sldv/sldv/sldvblockreplacementlib.mdl
```

Note The Simulink Design Verifier software provides two implementations of each factory default block replacement rule. Rules whose file names end with `_normal.m` replace blocks with Subsystem blocks. Rules whose file names end with `_configss.m` replace blocks with Configurable Subsystem blocks. See “Writing Block Replacement Rules” on page 3-10 for more information.

Template for Block Replacement Rules

To help you create block replacement rules, the Simulink® Design Verifier™ software provides an annotated M-file template containing a skeleton implementation of the requisite callbacks. The template resides at

```
matlabroot/toolbox/sldv/sldv/sldvblockreplacetemplate.m
```

To create a block replacement rule, make a copy of the template and edit the copy as necessary to reflect the desired behavior of the rule you are creating. The comments in the template help to explain how to implement your rule. See “Writing Block Replacement Rules” on page 3-10 for information about using the template to write custom block replacement rules.

Creating Custom Block Replacements

In this section...
“About Custom Block Replacements” on page 3-7
“Constructing Replacement Blocks” on page 3-7
“Writing Block Replacement Rules” on page 3-10

About Custom Block Replacements

The process of creating custom block replacements in the Simulink® Design Verifier™ software consists of the following tasks:

- “Constructing Replacement Blocks” on page 3-7
- “Writing Block Replacement Rules” on page 3-10

The Simulink Design Verifier software imposes several restrictions on replacement blocks. Replacement blocks must

- Use a masked Subsystem block that contains other Simulink® blocks.
- Reside in a block library that is available on your MATLAB® search path.
- Contain Inport and Outport blocks that have default names (e.g., In1 and Out1).

Note Be sure that you have read “Creating Block Masks” in *Using Simulink* before constructing a replacement block.

Constructing Replacement Blocks

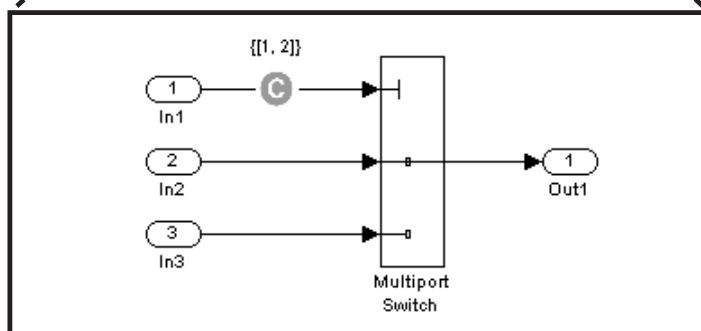
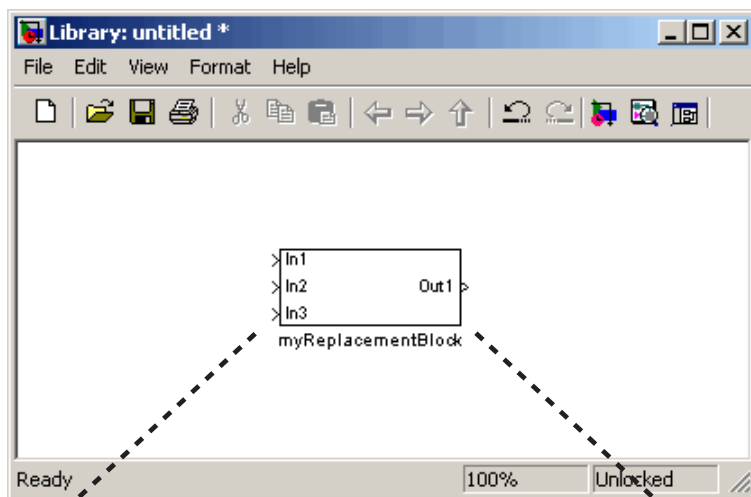
To create a replacement block:

- 1 Create a block library for your replacement block (see “Creating a Library” in *Using Simulink*). For example, from the **File** menu of the Simulink library window, select **New > Library**.

- 2 In your library, create a subsystem that represents your replacement block (see “Creating Subsystems” in *Using Simulink*).

This example uses a subsystem named `myReplacementBlock`, which contains a

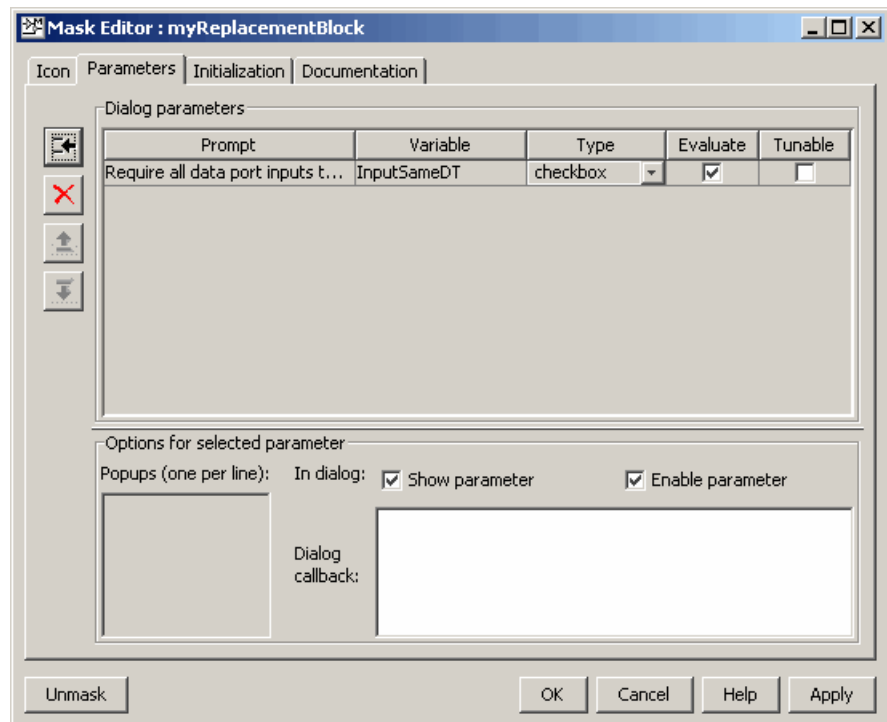
- Multiport Switch block whose **Number of inputs** parameter specifies 2
- Test Condition block whose **Values** parameter specifies `{[1, 2]}`



- 3 Create a mask for your subsystem (see “Masking a Subsystem” in *Using Simulink*).

In this example, the mask dialog box of the subsystem displays a mask parameter that controls the **Require all data port inputs to have the same data type** parameter of the underlying Multiport Switch block. The masked subsystem includes the following specifications in its Mask Editor:

- The **Parameters** pane defines a mask parameter named InputSameDT, which replicates the behavior of the **Require all data port inputs to have the same data type** parameter of the underlying Multiport Switch block.



Note When creating mask parameters that control the behavior of parameters associated with their underlying blocks, specify actual parameter names as dialog variables in the Mask Editor. For instance, `InputSameDT` is the actual parameter name that controls the **Require all data port inputs to have the same data type** parameter of the Multiport Switch block; hence, it specifies the name of the dialog variable in this example.

- The **Initialization** pane defines the following commands in the **Initialization commands** field:

```
maskInputSameDT = get_param(gcf,'InputSameDT');
blkName = sprintf('/Multiport\nSwitch');
targetBlock = [gcb, blkName];
set_param(targetBlock,'InputSameDT',maskInputSameDT);
```

- 4 Save your block library, e.g., as `custom_rule.mdl`, in a directory that is available on your MATLAB search path (see “Search Path” in the MATLAB documentation).

After constructing your replacement block, you are ready to write a custom block replacement rule, which the next section describes.

Writing Block Replacement Rules

The Simulink Design Verifier software imposes the following restrictions on block replacement rules:

- The M-file that represents a block replacement rule must include particular callbacks. The MathWorks recommends that you use the block replacement rule template as a starting point for writing a custom rule (see “Template for Block Replacement Rules” on page 3-6).
- The M-file that represents a block replacement rule must be available on the MATLAB search path.
- You cannot create a rule that replaces Inport, Outport, or Subsystem blocks in your model.

To write a rule for the replacement block you created in the previous section (see “Constructing Replacement Blocks” on page 3-7):

1 Make a copy of the block replacement rule template

```
matlabroot/toolbox/sldv/sldv/sldvblockreplacetemplate.m
```

saving it with an appropriate file name, e.g., `custom_rule_switch.m`.

Note In the remaining steps, you edit the copy of the template that you saved.

2 Rename the function, as defined on the first line of the M-file. The function name should be the same as its file name, without the `.m` extension. Optionally, you can edit the comments that follow the function declaration to create your own M-file help for this rule.

In this example, the first few lines of `custom_rule_switch.m` declare the function and its M-file help, which appear as follows:

```
function rule = custom_rule_switch
%CUSTOM_RULE_SWITCH Custom block replacement rule for
%the Simulink Design Verifier software
%
% This block replacement rule identifies Multiport
% Switch blocks whose "Number of inputs" parameter
% specifies '2' and "Use zero-based indexing" parameter
% specifies 'off'. It replaces such blocks with an
% implementation that includes a Test Condition block
% on the control input signal.
```

3 Identify the type of block that you wish to replace in your model by specifying its `BlockType` parameter as the `rule.blockType` object. Consider using the `get_param` function to obtain the value of the `BlockType` parameter for the block you intend to replace. Alternatively, you can determine this value by referring to “Block-Specific Parameters” in the *Simulink Reference*.

This example replaces Multiport Switch blocks, so the `rule.blockType` object specifies the appropriate `BlockType` parameter:

```
%% Target Block Type
%
rule.blockType = 'MultiPortSwitch';
```

- 4** Identify the replacement block by specifying its full block path name as the `rule.replacementPath` object. Consider using the `gcb` function to get the full block path name.

This example replaces Multiport Switch blocks with the replacement block developed in “Constructing Replacement Blocks” on page 3-7, so the `rule.replacementPath` object specifies the full block path name:

```
%% Replacement Library
%
rule.replacementPath = sprintf('custom_rule/myReplacementBlock');
```

- 5** Identify the type of subsystem that the Simulink Design Verifier software uses when replacing blocks by specifying a value for the `rule.replacementMode` object. Valid values include:
- **Normal** — When using this rule, the Simulink Design Verifier software replaces blocks with a copy of the subsystem specified by the `rule.replacementPath` object.
 - **ConfigurableSubSystem** — When using this rule, the Simulink Design Verifier software replaces blocks with a Configurable Subsystem block (see Configurable Subsystem in the *Simulink Reference*). The Configurable Subsystem block allows you to choose whether it represents the subsystem specified by the `rule.replacementPath` object, or the original block before its replacement.

This example replaces Multiport Switch blocks with an ordinary Subsystem block:

```
%% Replacement Mode
%
rule.replacementMode = 'Normal';
```


- 6** Identify parameter values that the replacement blocks inherit from the blocks being replaced. You achieve inheritance by mapping the parameter names in a structure. Each field of the structure represents a parameter that the replacement block inherits. Specify the value of each field using the token `$original.parameter$`, where *parameter* is the name of the parameter that belongs to the original block. You can determine block parameter names by referring to “Model and Block Parameters” in the *Simulink Reference*.

The following example defines a structure named `parameter` that maps the `InputSameDT` parameter from the original Multiport Switch blocks to their replacement blocks:

```
%% Parameter Handling
%
parameter.InputSameDT = '$original.InputSameDT$';

% Register the parameter mapping for the rule
rule.parameterMap = parameter;
```

- 7** Customize the subfunction named `replacementTestFunction` by specifying conditions under which the Simulink Design Verifier software replaces blocks in your model.

The following example instructs the Simulink Design Verifier software to replace only the Multiport Switch blocks whose `NumInputPorts` and `zeroIdx` parameters specify 2 and off, respectively:

```
function out = replacementTestFunction(blockH)
% Specify the logic that determines when the Simulink Design
% Verifier software replaces a block in your model. For example,
% restrict replacements to only the blocks whose parameters
% specify particular values.
out = false;
numInputPorts = eval(get_param(blockH,'NumInputPorts'));
zeroIdx = eval(get_param(blockH,'zeroIdx'));
if numInputPorts==2 && zeroIdx=='off',
    out = true;
end
```

After constructing a replacement block and writing its corresponding block replacement rule, you are ready to execute your custom block replacement (see “Executing Block Replacements” on page 3-15).

Executing Block Replacements

In this section...
“Configuring Block Replacements” on page 3-15
“Replacing Blocks in a Model” on page 3-16

Configuring Block Replacements

You must configure block replacement options before executing block replacements in your model. To specify block replacement options using the Simulink® GUI:

- 1 From the **Tools** menu of your Simulink model, select **Design Verifier > Options**.

The Configuration Parameters dialog box displays the Simulink® Design Verifier™ options.

- 2 In the **Select** tree of the Configuration Parameters dialog box, click the **Block Replacements** category.

The Configuration Parameters dialog box displays the **Block replacements** pane.

- 3 Enable block replacements by selecting the **Apply block replacements** option.

Enabling this option provides access to the **List of block replacement rules** and **File path of the output model** options.

- 4 In the **List of block replacement rules** box, enter file names of the block replacement rules that you wish to execute. The default value, <FactoryDefaultRules>, executes all the factory default rules (see “Built-In Block Replacements” on page 3-3).

You can specify multiple rules as a list delimited by spaces, commas, or carriage returns. The Simulink Design Verifier software executes the rules in the order that you list them. For example, to execute only a subset of the factory default rules followed by the custom block replacement

example from “Creating Custom Block Replacements” on page 3-7, enter the following file names:

```
blkrep_rule_mpswitch4_normal  
blkrep_rule_lookup_normal  
custom_rule_switch
```

Note The Simulink Design Verifier software replaces a block in your model only once. If multiple rules apply to the same block, the software replaces the block using the rule with the highest priority.

- 5 In the **File path of the output model** box, specify a directory to which the Simulink Design Verifier software saves the model that results after applying the block replacement rules.
- 6 Click the **OK** button to apply the changes and close the Configuration Parameters dialog box.

Alternatively, you can use the `sldvoptions` function at the command line to specify the block replacement options associated with a Simulink Design Verifier options object. See `sldvoptions` in Chapter 9, “Function Reference” for more information.

Replacing Blocks in a Model

After enabling the **Apply block replacements** option (see “Configuring Block Replacements” on page 3-15), you can execute block replacements in your model by starting a Simulink Design Verifier analysis. For example, to trigger block replacements from the Configuration Parameters dialog box, on the **Design Verifier** pane, click the **Analyze Model** button.

Note The Simulink Design Verifier software can execute block replacements only on models that have no unsaved changes.

When performing block replacements, the Simulink Design Verifier software copies your model and replaces blocks in the copy, leaving your original model

unaltered. Upon completing its analysis, the software generates a report that displays information about the block replacements it executed (see “Understanding Simulink® Design Verifier™ Reports” on page 8-8).

Alternatively, you can use the `sldvblockreplacement` function to execute block replacements from the command line or an M-file program. The syntax of the function is

```
status = sldvblockreplacement('system')
```

where *system* is the name of the model whose blocks you aim to replace. See `sldvblockreplacement` for more information.

If you execute block replacements programmatically, the Simulink Design Verifier software displays in the MATLAB® Command Window a table that lists available block replacement rules:

Configuration of available block replacement rules:

Type:	Registration M-File name:	Block types:	Priority:	Active:
Built-in	blkrep_rule_mpswitch2_normal.m	MultiPortSwitch	5	0
Built-in	blkrep_rule_mpswitch2_configss.m	MultiPortSwitch	4	0
Built-in	blkrep_rule_mpswitch3_normal.m	MultiPortSwitch	3	0
Built-in	blkrep_rule_mpswitch3_configss.m	MultiPortSwitch	6	0
Built-in	blkrep_rule_mpswitch4_normal.m	MultiPortSwitch	1	1
Built-in	blkrep_rule_mpswitch4_configss.m	MultiPortSwitch	7	0
Built-in	blkrep_rule_mpswitch5_normal.m	MultiPortSwitch	2	0
Built-in	blkrep_rule_mpswitch5_configss.m	MultiPortSwitch	8	0
Built-in	blkrep_rule_lookup_normal.m	Lookup	1	1
Built-in	blkrep_rule_lookup_configss.m	Lookup	2	0
Built-in	blkrep_rule_switch_normal.m	Switch	1	0
Built-in	blkrep_rule_switch_configss.m	Switch	2	0
Built-in	blkrep_rule_lookup2D_normal.m	Lookup2D	1	0
Built-in	blkrep_rule_lookup2D_configss.m	Lookup2D	2	0
Built-in	blkrep_rule_selectorIndexVecPort_normal.m	Selector	1	0
Built-in	blkrep_rule_selectorIndexVecPort_configss.m	Selector	2	0
Built-in	blkrep_rule_selectorStartingIdxPort_normal.m	Selector	3	0
Built-in	blkrep_rule_selectorStartingIdxPort_configss.m	Selector	4	0
Custom	custom_rule_switch.m	MultiPortSwitch	2	1

The list of available block replacement rules includes all built-in rules and any custom rules that you specified using the **List of block replacement rules** option (see “Configuring Block Replacements” on page 3-15). The columns of the preceding table identify the following information:

- Type — the type of rule, either built-in or custom
- Registration M-File name — the name of the M-file that expresses the rule
- Block types — the BlockType parameter value of the block that the rule replaces
- Priority — the priority of execution when multiple rules target the same type of block for replacement
- Active — a flag that indicates whether the rule is active (1) or ignored (0)

Also, the Simulink Design Verifier software displays information about the block replacements that it performed. For example, the following message indicates that the software used the `custom_rule_switch.m` rule to replace a Multiport Switch block (of the same name) at the top level of the model:

```
Performed block replacements:
```

```
Replacement rule M-file name:  Replaced block:  
custom_rule_switch.m         ./Multiport Switch
```

Specifying Parameter Configurations

The Simulink® Design Verifier™ software allows you to treat block parameters in your model as variables in its analysis. The following sections introduce parameter configurations and illustrate a process for specifying constraints on block parameters.

About Parameter Configurations
(p. 4-2)

Brief overview of parameter configurations.

Template for Parameter Configurations (p. 4-3)

Introduces the template for creating a parameter configuration file.

Defining Parameter Configurations
(p. 4-4)

Describes how to define parameter configurations.

Parameter Configuration Example
(p. 4-7)

Provides an example that walks you through the process of specifying a parameter configuration.

About Parameter Configurations

The Simulink® Design Verifier™ software can treat block parameters in your model as variables during its analysis. For example, suppose you specify a variable that is defined in the MATLAB® workspace as the value of a block parameter in your model. You can instruct the Simulink Design Verifier software to treat that parameter as another input variable in its analysis. This allows you to

- Extend the results of a proof to consider the impact of additional parameter values.
- Generate comprehensive test cases for situations in which parameter values must vary to achieve more complete coverage results (for an example, see “Parameter Configuration Example” on page 4-7).

Template for Parameter Configurations

To help you create a parameter configuration file, the Simulink® Design Verifier™ software provides an annotated M-file template. The template resides at

```
matlabroot/toolbox/sldv/sldv/sldv_params_template.m
```

Alternatively, you can access the template from the **Parameters** pane in the Simulink Design Verifier options (see “Parameters Pane” on page 5-8).

To create a parameter configuration file, make a copy of the template and edit the copy. The comments in the template explain the syntax for defining parameter configurations. For more information about defining parameter configurations, see “Defining Parameter Configurations” on page 4-4.

Defining Parameter Configurations

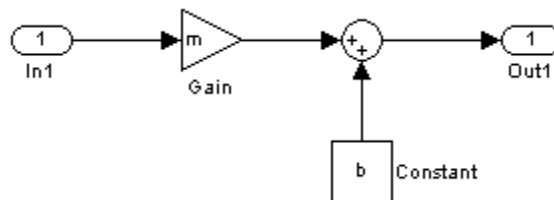
This section describes how to define parameter configurations and outlines the required syntax for their definition.

1 Define parameter configurations in an M-file function.

The Simulink® Design Verifier™ software provides an annotated template for an M-file function that you can use as a starting point (see “Template for Parameter Configurations” on page 4-3).

2 Specify parameter configurations using a structure whose fields share the same names as the parameters that you treat as input variables.

For example, suppose you wish to constrain the **Gain** and **Constant value** parameters, *m* and *b*, which appear in the following model:



In your parameter configuration file, use the following names for the fields of the structure:

```

params.m
params.b
  
```

3 Constrain parameters by assigning values to the fields of the structure.

Specify points using the `Sldv.Point` constructor, which accepts a single value as its argument. Specify intervals using the `Sldv.Interval` constructor, which requires two input arguments, i.e., a lower bound and an upper bound for the interval. Optionally, you can provide one of the following strings as a third input argument that specifies inclusion or exclusion of the interval endpoints:

- '()' — Defines an open interval.

- '[' — Defines a closed interval.
- '(' — Defines a left-open interval.
- '[' — Defines a right-open interval.

Note By default, the Simulink Design Verifier software considers an interval to be closed if you omit its two-character string.

The following example constrains m to 3 and b to any value in the closed interval $[0, 10]$:

```
params.m = Sldv.Point(3);
params.b = Sldv.Interval(0, 10);
```

If the parameters are scalar, you can omit the constructors and instead specify single values or two-element vectors. For instance, you can alternatively specify the previous example as:

```
params.m = 3;
params.b = [0 10];
```

4 Use cell arrays to specify multiple constraints for a single parameter.

You can specify multiple constraints for a single parameter by using a cell array. In this case, the Simulink Design Verifier software combines the constraints using a logical OR operation during its analysis.

The following example constrains m to either 3 or 5, and it constrains b to any value in the closed interval $[0, 10]$:

```
params.m = {3, 5};
params.b = [0 10];
```

5 Use a 1-by- n structure to specify n sets of parameters.

You can specify several sets of parameters by expanding the size of your structure.

For instance, the following example uses a 1-by-2 structure to define two sets of parameters:

```
params(1).m = {3, 5};  
params(1).b = [0 10];  
  
params(2).m = {12, 15, Sldv.Interval(50, 60, '()')};  
params(2).b = 5;
```

The first parameter set constrains m to either 3 or 5, and it constrains b to any value in the closed interval $[0, 10]$. The second parameter set constrains m to either 12, 15, or any value in the open interval $(50, 60)$, and it constrains b to 5.

Parameter Configuration Example

In this section...
“About This Example” on page 4-7
“Constructing the Example Model” on page 4-8
“Parameterizing the Constant Block” on page 4-11
“Specifying a Parameter Configuration” on page 4-12
“Analyzing the Example Model” on page 4-13
“Simulating the Test Cases” on page 4-16

About This Example

The sections that follow describe a simple Simulink® model, for which you generate test cases that achieve decision coverage. However, in this example, achieving complete decision coverage is possible only when the Simulink® Design Verifier™ software treats a particular block parameter as a variable during its analysis. Toward that end, this example will help you understand how to specify parameter configurations for use with the Simulink Design Verifier software.

The following workflow guides you through the process of completing this example:

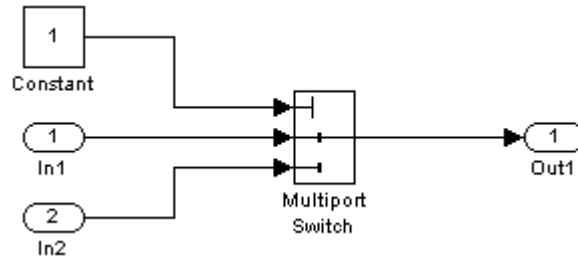
Task	Description	See...
1	Construct the example model.	“Constructing the Example Model” on page 4-8
2	Specify a variable as the value of a Constant block parameter.	“Parameterizing the Constant Block” on page 4-11
3	Constrain the value of the variable that the Constant block specifies.	“Specifying a Parameter Configuration” on page 4-12

Task	Description	See...
4	Generate test cases for your model and interpret the results.	“Analyzing the Example Model” on page 4-13
5	Simulate the test cases and measure the resulting decision coverage.	“Simulating the Test Cases” on page 4-16

Constructing the Example Model

This section presents Task 1 of the process that describes how to specify parameter configurations in the Simulink Design Verifier software. In this task, you construct a simple Simulink model that you use throughout the remaining tasks. To complete this task, perform the following steps:

- 1 Create an empty Simulink model (see “Creating an Empty Model” in *Using Simulink* for help with this step).
- 2 Copy the following blocks into your empty model window (see “Adding Blocks to Your Model” in the Simulink documentation for help with this step):
 - Two Inport blocks to initiate the input signals, from the Sources library
 - A Multiport Switch block to provide simple logic, from the Signal Routing library
 - A Constant block to control the switch, from the Sources library
 - An Outport block to receive the output signal, from the Sinks library
- 3 In your model window, double-click the Multiport Switch block to access its dialog box and specify its **Number of inputs** option as 2.
- 4 In your model window, connect the blocks so that your model looks like this (see “Connecting Blocks” in *Using Simulink* for help with this step):

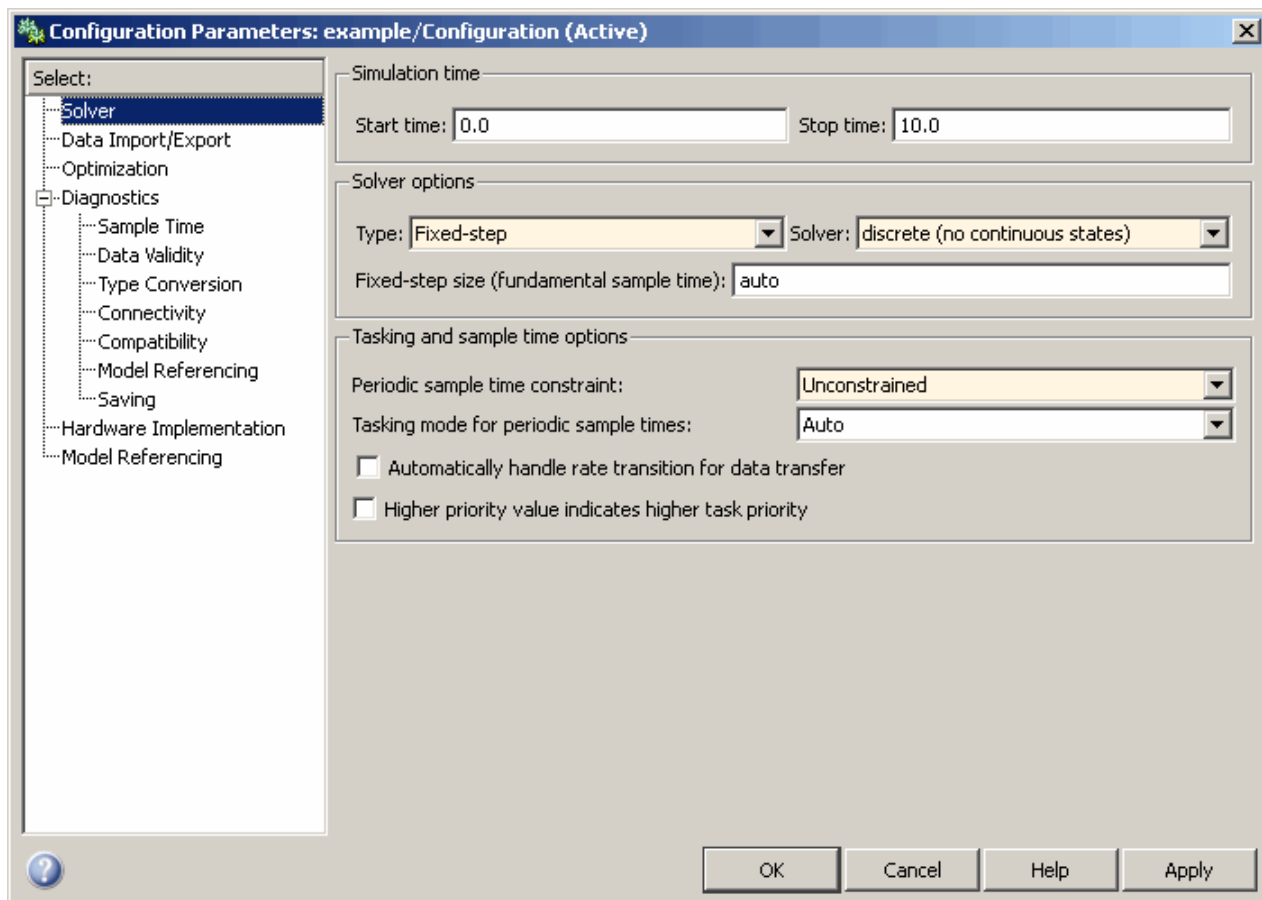


- 5 In your model window, select **Simulation > Configuration Parameters**.

The Configuration Parameters dialog box appears.

- 6 In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Solver** category (if not already selected). Under **Solver options** on the right side, set the **Type** option to Fixed-step, and then set the **Solver** option to discrete (no continuous states).

The Configuration Parameters dialog box appears as follows:



7 Click the **OK** button to apply your changes and close the Configuration Parameters dialog box.

8 Save your model as `param_example.mdl` (see “Saving a Model” in *Using Simulink* for help with this step).

What to do next: Now you are ready to begin Task 2 of this example, “Parameterizing the Constant Block” on page 4-11.

Parameterizing the Constant Block

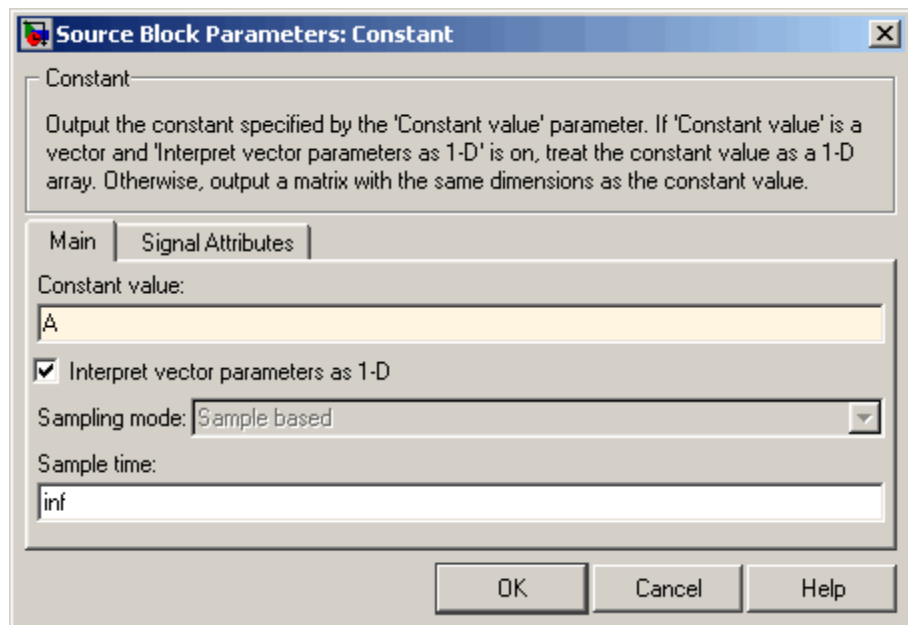
This section presents Task 2 of the process that describes how to specify parameter configurations in the Simulink Design Verifier software. In this task, you parameterize the Constant block in the model that you created in the previous task (see “Constructing the Example Model” on page 4-8). In particular, you specify a variable as the value of the Constant block’s **Constant value** parameter. To complete this task, perform the following steps:

- 1 In your model window, double-click the Constant block.

The Constant block parameter dialog box appears.

- 2 In the **Constant value** box, enter A.

The Constant block parameter dialog box appears as follows:



- 3 Click the **OK** button to apply your change and close the Constant block parameter dialog box.

- 4 In the MATLAB® Command Window, enter

```
A = 1;
```

This command defines in the MATLAB workspace a variable named A whose value is 1. The Simulink software resolves the **Constant value** parameter to this variable, initializing its value for simulation.

What to do next: Now you are ready to begin Task 3 of this example, “Specifying a Parameter Configuration” on page 4-12.

Specifying a Parameter Configuration

This section presents Task 3 of the process that describes how to specify parameter configurations in the Simulink Design Verifier software. In this task, you customize the parameter configuration file template so that it constrains the variable that you specified in the previous task (see “Parameterizing the Constant Block” on page 4-11). To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Options**.

The Simulink Design Verifier software displays its options in the Configuration Parameters dialog box.

- 2 In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Design Verifier > Parameters** category. In the **Parameters** pane on the right side, ensure that the **Apply parameters** option is enabled.

Enabling the **Apply parameters** option provides access to the **Parameter configuration file** option.

- 3 Click the **Edit** button next to the **Parameter configuration file** option.

The Simulink Design Verifier software opens `sldv_params_template.m` in an editor.

- 4 Edit the template’s text so that it appears as follows:

```
function params = param_example_function
```

```
% This function defines a parameter configuration for the  
% example model that the documentation discusses.
```

```
params.A = [1 2];
```

The preceding code renames the function as `params_example_function` and constrains parameter A to the closed interval [1 2].

- 5 Save your changes to the template as `params_example_function.m` in the same directory as the example model.
- 6 In the Configuration Parameters dialog box, click the **Browse** button next to the **Parameter configuration file** option, and then select your parameter configuration file, `params_example_function.m`.
- 7 Click the **OK** button to apply your change and close the Configuration Parameters dialog box.

What to do next: Now you are ready to begin Task 4 of this example, “Analyzing the Example Model” on page 4-13.

Analyzing the Example Model

This section presents Task 4 of the process that describes how to specify parameter configurations in the Simulink Design Verifier software. In this task, you execute the Simulink Design Verifier analysis, which uses the parameter configuration file you customized in the previous task (see “Specifying a Parameter Configuration” on page 4-12). The software generates test cases and produces results for you to interpret. To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Generate Tests**.

The Simulink Design Verifier software begins analyzing your model to generate test cases. When the software completes its analysis, it generates the following items:

- Simulink Design Verifier report — The Simulink Design Verifier software displays an HTML report named `param_example_report.html`.

- Test harness — The Simulink Design Verifier software displays a harness model named `param_example_harness.mdl`.

2 In the Simulink Design Verifier report **Table of Contents**, click Test Case 1.

The report displays its Test Case 1 section, which appears as follows:

Test Case 1

Summary

Length: 0.2 Seconds (2 sample periods)

Objective Count: 1

Parameters:

Parameter	Value
A	1

Objectives Reached At:

Step	Time	Objectives
1	0	<u>2</u>

Generated Input Data.

Time 0
Step 1
In1 -
In2 -

This section provides details about Test Case 1 that the Simulink Design Verifier software generated to satisfy a coverage objective in the model. In this test case, a value of 1 for parameter A satisfies the objective.

3 Go to the Test Case 2 section in the **Test Cases** chapter.

The Test Case 2 section of the report appears as follows:

Test Case 2

Summary

Length: 0.2 Seconds (2 sample periods)

Objective Count: 1

Parameters:

Parameter	Value
A	2

Objectives Reached At:

Step	Time	Objectives
1	0	<u>1</u>

Generated Input Data.

Time 0	
Step 1	
In1	-
In2	-

This section provides details about Test Case 2, which satisfies another coverage objective in the model. In this test case, a value of 2 for parameter A satisfies the objective.

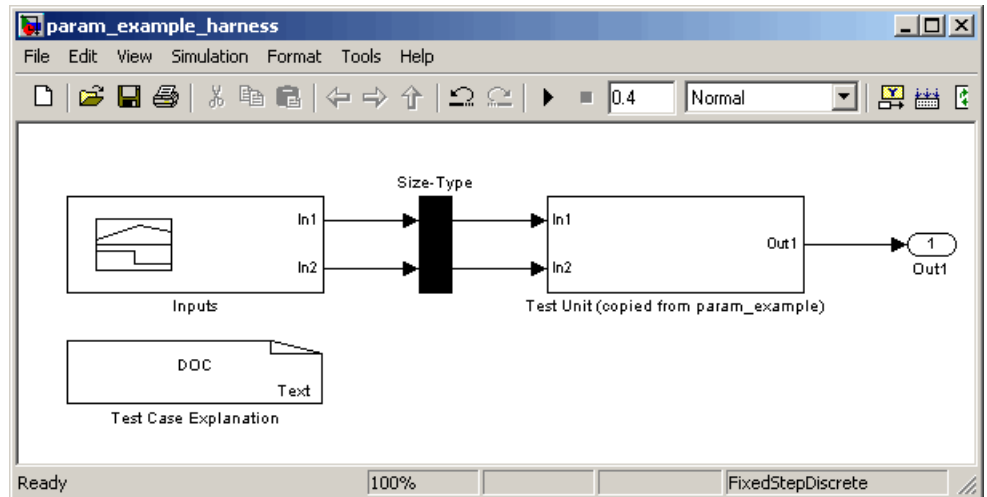
What to do next: Now you are ready to begin Task 5 of this example, “Simulating the Test Cases” on page 4-16.

Simulating the Test Cases

This section presents Task 5 of the process that describes how to specify parameter configurations in the Simulink Design Verifier software. In this final task, you simulate the test cases that the Simulink Design Verifier software generated in the previous task (see “Simulating the Test Cases” on page 4-16). Also, you review the coverage report that results from the simulation. To complete this task, perform the following steps:

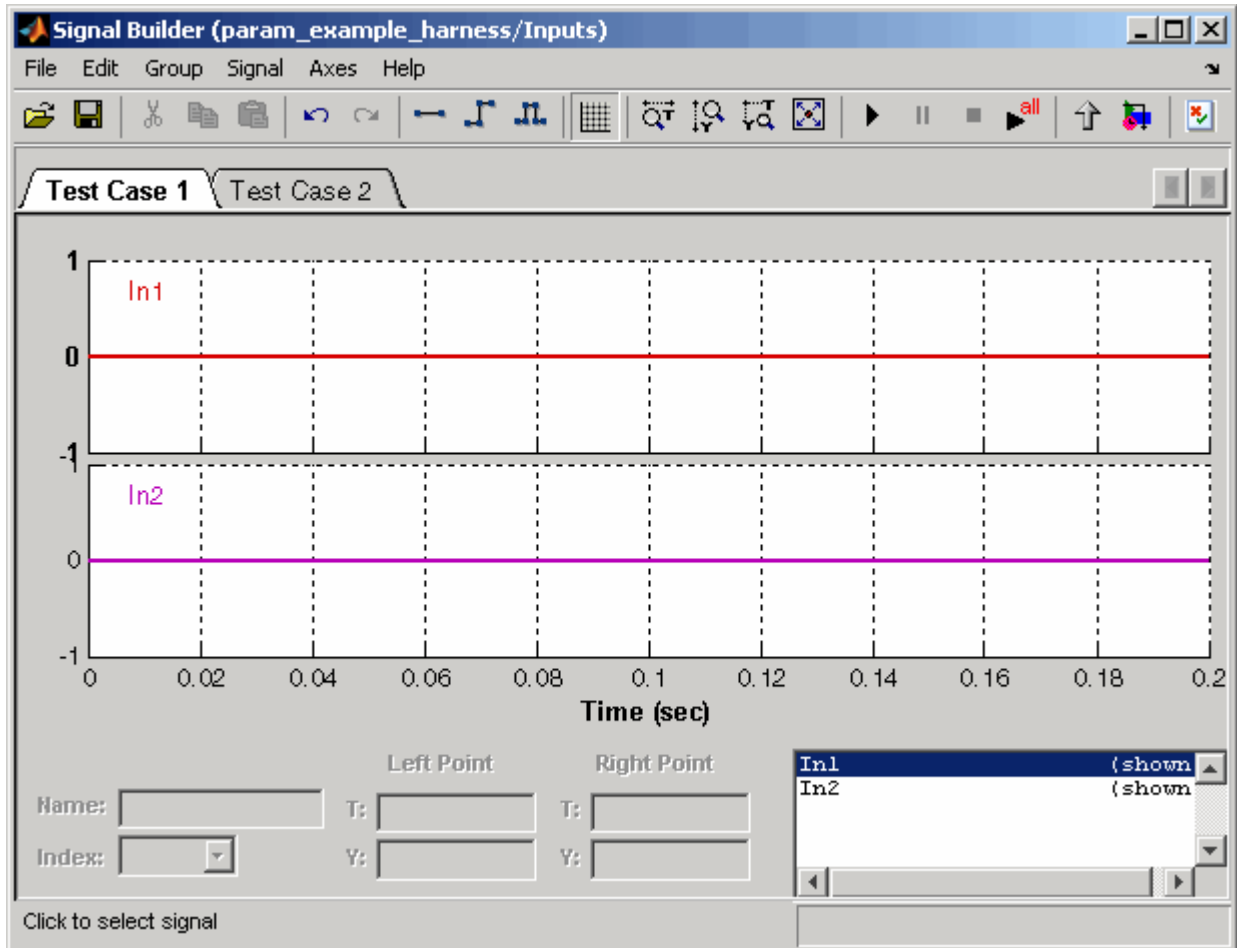
- 1 Open the test harness model named `param_example_harness.mdl` (if it is not already open).


The test harness model appears as follows:



- 2 The block labeled "Inputs" in the test harness model is a Signal Builder block that contains the test case signals. Double-click the "Inputs" block to view the test case signals.

The Signal Builder dialog box appears as follows:





- 3** In the Signal Builder dialog box, click the **Run all** button .

The Simulink software simulates each of the test cases in succession, collects coverage data for each simulation, and displays a report of the combined coverage results at the end of the last simulation.

- 4** In the model coverage report, review the Summary section:

Summary

Model Hierarchy/Complexity:		Test 1
		D1
1. param_example_harness	2	100% 
2. . . . Test Unit (copied from param_example)	1	100% 

This section summarizes the coverage results for the harness model and its Test Unit subsystem. Observe that the subsystem achieves 100% decision coverage.

5 In the **Summary**, click the Test Unit subsystem.

The report displays detailed coverage results for the Test Unit subsystem.

2. Subsystem "[Test Unit \(copied from param_example\)](#)"

Parent: [/param_example_harness](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	1
Decision (D1)	NA	100% (2/2) decision outcomes

Mpswitch block "[Multiport Switch](#)"

Parent: [param_example_harness/Test Unit \(copied from param_example\)](#)

Metric	Coverage
Cyclomatic Complexity	1
Decision (D1)	100% (2/2) decision outcomes

Decisions analyzed:

truncated input value	100%
= 1 (output is from input port 2)	51/102
= 2 (output is from input port 3)	51/102

This section reveals that the Multiport Switch block achieves complete decision coverage because the test cases exercise each of its switch pathways.

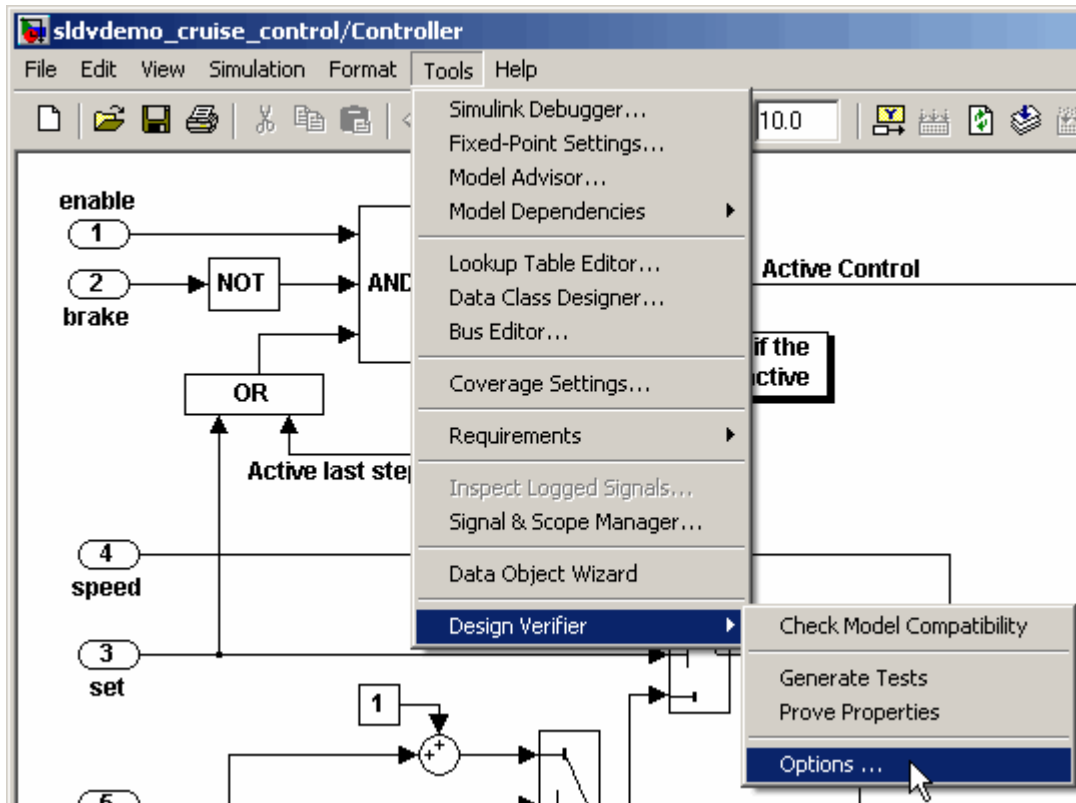
Configuring Simulink® Design Verifier™ Options

This chapter provides an overview of the Simulink® Design Verifier™ options that you specify typically with the Configuration Parameters dialog box. The following sections step you through the Simulink Design Verifier dialog panes and describe its options.

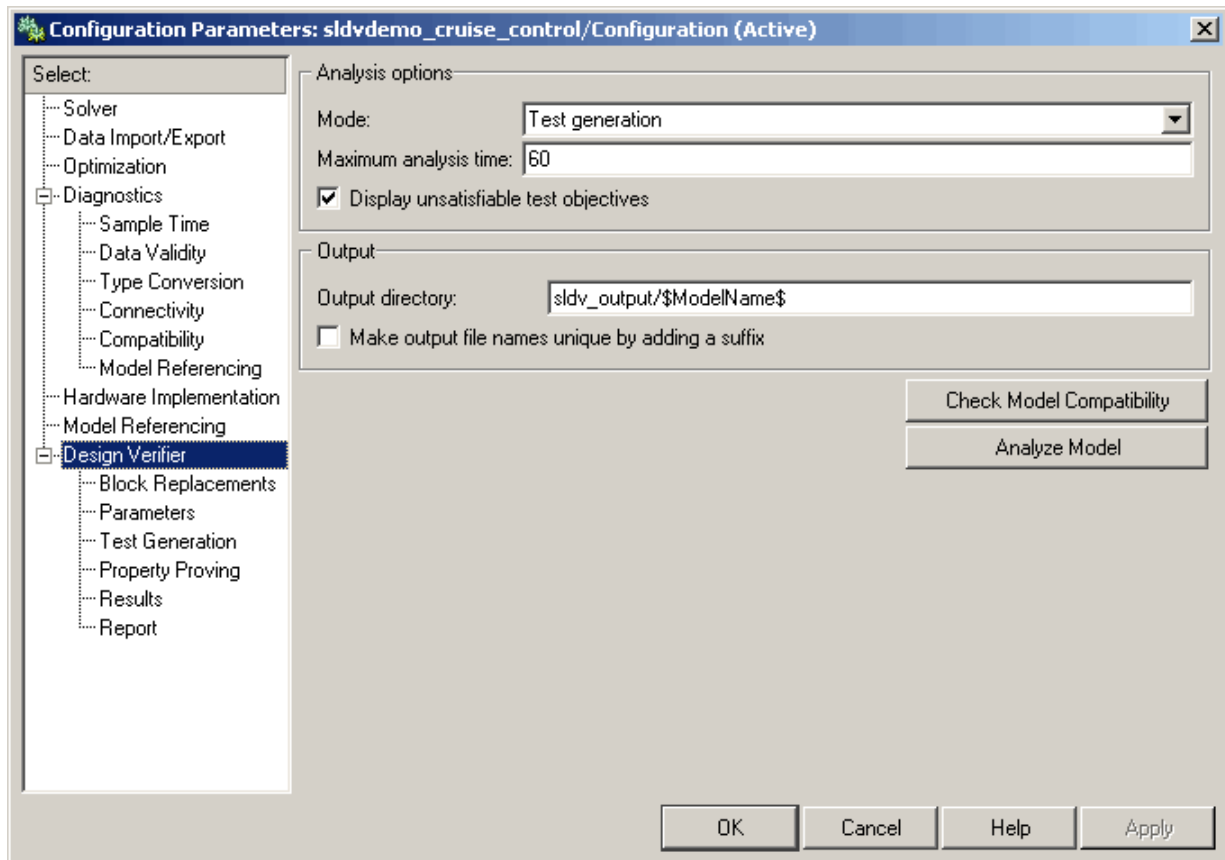
Viewing Simulink® Design Verifier™ Options (p. 5-2)	Explains how to view options that control the Simulink Design Verifier software.
Configuring Simulink® Design Verifier™ Options (p. 5-5)	Describes the options that control the Simulink Design Verifier software.
Saving Simulink® Design Verifier™ Options (p. 5-16)	Discusses how the Simulink Design Verifier software saves its options.

Viewing Simulink® Design Verifier™ Options

The Simulink® Design Verifier™ software provides numerous options that control its behavior when analyzing models. To view its options, from the **Tools** menu of your Simulink® model, select **Design Verifier > Options**.



The Simulink Design Verifier software displays its options in the Configuration Parameters dialog box.



Typically, you specify values for these options using the Configuration Parameters dialog box. See “Configuration Parameters Dialog Box” in *Simulink Graphical User Interface* for more information about working with this interface.

Note By default, Simulink Design Verifier options do not appear in a model's Configuration Parameters dialog box. If you select **Design Verifier > Options** from a model's **Tools** menu, the Simulink Design Verifier software associates its options with that model. Afterward, you can access those options directly from the Configuration Parameters dialog box or Model Explorer (see “The Model Explorer” in *Using Simulink*).

Alternatively, you can use the `sldvoptions` function to view Simulink Design Verifier options at the command line. Use the following syntax to access and view programmatically the options associated with the Simulink model *system*:

```
opts = sldvoptions('system');  
get(opts)
```

See `sldvoptions` in Chapter 9, “Function Reference” for more information.

Configuring Simulink® Design Verifier™ Options

In this section...

- “Design Verifier Pane” on page 5-5
- “Block Replacements Pane” on page 5-6
- “Parameters Pane” on page 5-8
- “Test Generation Pane” on page 5-9
- “Property Proving Pane” on page 5-11
- “Results Pane” on page 5-12
- “Report Pane” on page 5-14

Design Verifier Pane

The **Design Verifier** pane allows you to specify analysis options and configure Simulink® Design Verifier™ output.

The screenshot shows the Design Verifier configuration window. It is divided into two main sections: "Analysis options" and "Output".

Analysis options:

- Mode: Test generation (dropdown menu)
- Maximum analysis time: 600 (text input)
- Display unsatisfiable test objectives

Output:

- Output directory: sldv_output/\$ModelName\$ (text input)
- Make output file names unique by adding a suffix

At the bottom right, there are two buttons: "Check Model Compatibility" and "Analyze Model".

The **Design Verifier** pane contains the following groups of options:

- “Analysis options” on page 5-6
- “Output” on page 5-6

Analysis options

This group contains controls that enable you to specify how the Simulink Design Verifier software analyzes Simulink® models. It contains the following controls.

Mode. Specifies the mode in which the Simulink Design Verifier software operates, either Test generation (the default) or Property proving.

Maximum analysis time. Specifies the maximum time (in seconds) that the Simulink Design Verifier software spends analyzing the model.

Display unsatisfiable test objectives. If selected, this option causes the Simulink Design Verifier software to display a warning message in the Simulation Diagnostics Viewer when it is unable to satisfy a test objective. See “Simulation Diagnostics Viewer” in *Using Simulink* for more information.

Output

This group contains controls that enable you to configure Simulink Design Verifier output. It contains the following controls.

Output directory. Specifies a directory to which the Simulink Design Verifier software writes its output. Enter a path that is either absolute or relative to the current directory.

The default value is `sldv_output/$modelName$`, where `$modelName$` is a token that represents the model name.

Make output file names unique by adding a suffix. If selected, this option causes the Simulink Design Verifier software to append an incremental numeric suffix to output file names. Selecting this option prevents the software from overwriting existing files that have the same name.

Block Replacements Pane

The **Block Replacements** pane allows you to specify options that control how the Simulink Design Verifier software preprocesses the models it analyzes.

The image shows a dialog box titled "Block replacements" with a light gray background. At the top left, the title "Block replacements" is displayed. Below it is a checkbox labeled "Apply block replacements" which is currently unchecked. Underneath the checkbox is the text "List of block replacement rules (in order of priority):" followed by a large, empty rectangular text area. At the bottom of the dialog box, there is a section titled "Output model" containing a text input field with the label "File path of the output model:" and a cursor inside the field.

Block replacements

This group contains controls that enable you to specify block replacement options. It contains the following controls.

Apply block replacements. If selected, this option causes the Simulink Design Verifier software to replace blocks in the model before its analysis (see Chapter 3, “Working with Block Replacements”). By default, this option is disabled. Enabling this option provides access to the **List of block replacement rules** and **File path of the output model** options.

List of block replacement rules. Specifies a list of block replacement rules that the Simulink Design Verifier software processes before analyzing the model. This option is accessible only if **Apply block replacements** is selected. The software processes the block replacement rules in the order that you list them.

Specify block replacement rules as a list delimited by spaces, commas, or carriage returns (see “Configuring Block Replacements” on page 3-15).

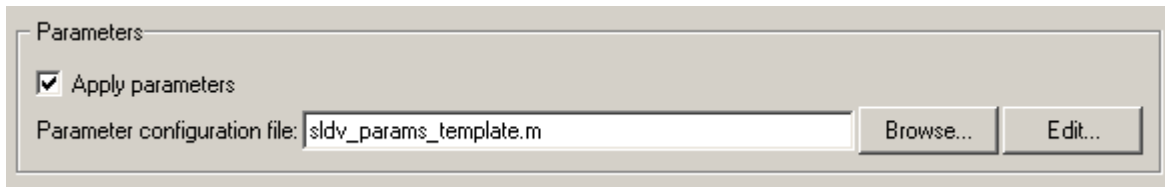
The default value is <FactoryDefaultRules>. If you specify the default value, the Simulink Design Verifier software uses its factory default block replacement rules (see “Built-In Block Replacements” on page 3-3).

File path of the output model. Specifies a directory to which the Simulink Design Verifier software saves the model that results after applying the block replacement rules. Enter a pathname that is either absolute or relative to the pathname specified as the **Output directory**. This option is accessible only if **Apply block replacements** is selected.

The default value is \$ModelName\$_replacement, where \$ModelName\$ is a token that represents the model name.

Parameters Pane

The **Parameters** pane allows you to specify options that control how the Simulink Design Verifier software uses parameter configurations when analyzing models.



Parameters

This group contains controls that enable you to specify parameter configurations. It contains the following controls.

Apply parameters. If selected (the default), this option causes the Simulink Design Verifier software to use parameter configurations when analyzing a model (see Chapter 4, “Specifying Parameter Configurations”). Enabling this option provides access to the **Parameter configuration file** option.

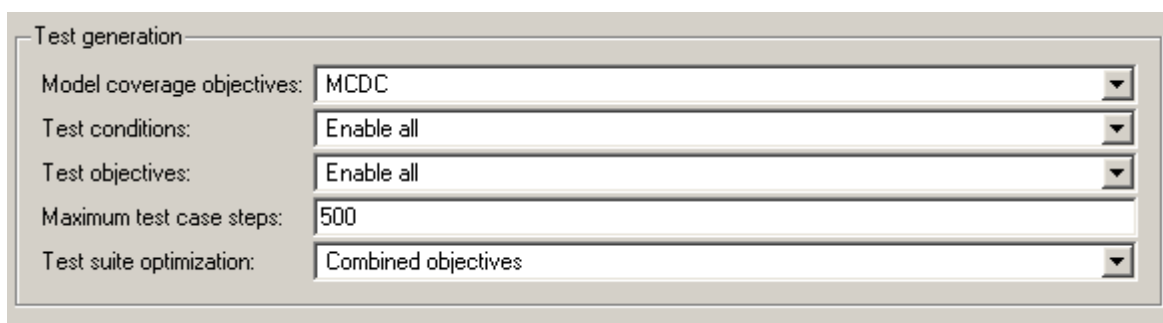
Parameter configuration file. Specifies an M-file function that defines parameter configurations for a model. Click the **Browse** button to select an existing M-file function using a file chooser dialog box. Click the **Edit** button to open the specified M-file function in an editor.

The default value is `sldv_params_template.m`, a template that you can edit and save. The comments in the template explain the syntax you use to specify parameter configurations.

Tip See the Parameter Identification Example demo for an illustration of how to use parameter configurations when generating tests cases for a Simulink model.

Test Generation Pane

The **Test Generation** pane allows you to specify options that control how the Simulink Design Verifier software generates tests for the models it analyzes.



The screenshot shows a dialog box titled "Test generation" with the following settings:

Model coverage objectives:	MCDC
Test conditions:	Enable all
Test objectives:	Enable all
Maximum test case steps:	500
Test suite optimization:	Combined objectives

Test generation

This group contains controls that enable you to specify test generation options. It contains the following controls.

Model coverage objectives. Specifies the type of model coverage that the Simulink Design Verifier software attempts to achieve. Select either Decision, Condition Decision, MCDC, or None.

Test conditions. This option allows you to enable or disable Test Condition blocks in the current model either globally or locally. Select one of the following options:

- **Use local settings** — Enables or disables Test Condition blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.
- **Enable all** — Enables all Test Condition blocks in the model regardless of the settings of their **Enable** parameters.
- **Disable all** — Disables all Test Condition blocks in the model regardless of the settings of their **Enable** parameters.

Test objectives. This option allows you to enable or disable Test Objective blocks in the current model either globally or locally. Select one of the following options:

- **Use local settings** — Enables or disables Test Objective blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.
- **Enable all** — Enables all Test Objective blocks in the model regardless of the settings of their **Enable** parameters.
- **Disable all** — Disables all Test Objective blocks in the model regardless of the settings of their **Enable** parameters.

Maximum test case steps. Specifies the maximum number of simulation steps the Simulink Design Verifier software takes when attempting to satisfy a test objective.

Test suite optimization. This option allows you to specify the optimization strategy that the Simulink Design Verifier software uses when generating test cases. Select one of the following options:

- **Combined objectives** — Minimizes the number of test cases in a suite by generating cases that address more than one test objective. Each test case tends to be long, i.e., it includes many time steps.
- **Individual objectives** — Maximizes the number of test cases in a suite by generating cases that each address only one test objective. Each test case tends to be short, i.e., it includes only a few time steps.
- **Large model** — Minimizes the number of test cases in a suite by generating cases that address more than one test objective. This strategy is tailored for large models that contain nonlinearities and numerous test objectives;

consequently, it tends to use all the time that the **Maximum analysis time** option allots.

Property Proving Pane

The **Property Proving** pane allows you to specify options that control how the Simulink Design Verifier software proves properties for the models it analyzes.

The screenshot shows a dialog box titled "Property proving" with the following settings:

- Assertion blocks: Enable all
- Proof assumptions: Enable all
- Strategy: Find violation
- Maximum violation steps: 20

Property proving

This group contains controls that enable you to specify property proving options. It contains the following controls.

Assertion blocks. This option allows you to enable or disable Assertion blocks in the current model either globally or locally. Select one of the following options:

- Use local settings — Enables or disables Assertion blocks based on the value of the **Enable assertion** parameter of each block. If a block's **Enable assertion** parameter is selected, the block is enabled; otherwise, the block is disabled.
- Enable all — Enables all Assertion blocks in the model regardless of the settings of their **Enable assertion** parameters.
- Disable all — Disables all Assertion blocks in the model regardless of the settings of their **Enable assertion** parameters.

Proof assumptions. This option allows you to enable or disable Proof Assumption blocks in the current model either globally or locally. Select one of the following options:

- **Use local settings** — Enables or disables Proof Assumption blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.
- **Enable all** — Enables all Proof Assumption blocks in the model regardless of the settings of their **Enable** parameters.
- **Disable all** — Disables all Proof Assumption blocks in the model regardless of the settings of their **Enable** parameters.

Strategy. Specifies the strategy the Simulink Design Verifier software uses when proving properties. Select one of the following options:

- **Find violation** — If this strategy is selected, the Simulink Design Verifier software searches for property violations within the number of simulation steps specified by the **Maximum violation steps** option. Enabling this option provides access to the **Maximum violation steps** option.
- **Prove** — If this strategy is selected, the Simulink Design Verifier software performs property proofs.
- **Prove with violation detection** — This strategy combines the **Find violation** and **Prove** strategies. If selected, the Simulink Design Verifier software searches for property violations within the number of simulation steps specified by the **Maximum violation steps** option; then it attempts to prove properties for which it failed to detect a violation. Enabling this option provides access to the **Maximum violation steps** option.

See “Strategies for Proving Properties of Large Models” on page A-13 for more information.

Maximum violation steps. Specifies the maximum number of simulation steps over which the Simulink Design Verifier software searches for property violations. The software does not search beyond the maximum number of simulation steps that you specify; it does not identify violations that occur later in a simulation. This option is accessible only if **Strategy** specifies either **Find violation** or **Prove with violation detection**.

Results Pane

The **Results** pane allows you to specify options that control how the Simulink Design Verifier software handles the results that it generates.

The screenshot shows a dialog box with two main sections:

- Harness model options:**
 - Save test harness as model
 - Harness model file name:
- Data file options:**
 - Save test data to file
 - Data file name:
 - Include expected output values
 - Randomize data that do not affect the outcome

The **Results** pane contains the following groups of options:

- “Harness model options” on page 5-13
- “Data file options” on page 5-14

Harness model options

This group contains controls that enable you to specify how the Simulink Design Verifier software handles the test harness it produces. It contains the following controls.

Save test harness as model. If selected, this option causes the Simulink Design Verifier software to save the test harness it generates as a model file. Enabling this option provides access to the **Harness model file name** option.

Harness model file name. Specifies a file name with which the Simulink Design Verifier software saves the test harness it generates. Enter a pathname that is either absolute or relative to the pathname specified by **Output directory**. This option is accessible only if **Save test harness as model** is selected.

The default value is \$ModelName\$_harness, where \$ModelName\$ is a token that represents the model name.

Data file options

This group contains controls that enable you to specify how the Simulink Design Verifier software handles the MAT-file it produces. It contains the following controls.

Save test data to file. If selected, this option causes the Simulink Design Verifier software to save the test data it generates to a MAT-file. Enabling this option provides access to the **Data file name** option.

Data file name. Specifies a file name with which the Simulink Design Verifier software saves the MAT-file it generates. Enter a pathname that is either absolute or relative to the directory specified by **Output directory**. This option is accessible only if **Save test data to file** is selected.

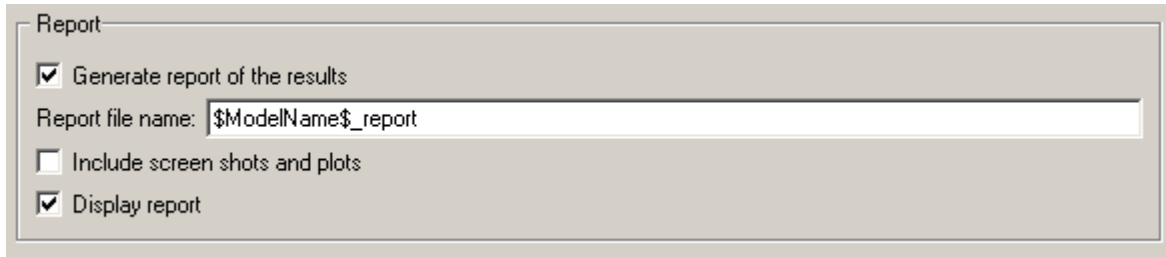
The default value is `$ModelName$_sldvdata`, where `$ModelName$` is a token that represents the model name.

Include expected output values. If selected, this option causes the Simulink Design Verifier software to simulate the model using the test case signals that it produces. For each test case, the software collects the simulation output values associated with Outputport blocks in the top-level system and includes those values in the MAT-file that it generates (see “TestCases Field” on page 8-26).

Randomize data that does not affect outcome. If selected, this option causes the Simulink Design Verifier software to assign random values instead of zeros to test case or counterexample signals that have no impact on test or proof objectives in a model. In the Simulink Design Verifier report, the Generated Input Data table always displays a dash (–) for such signals (see “Test Cases / Counterexamples Chapter” on page 8-19).

Report Pane

The **Report** pane allows you to specify options that control how the Simulink Design Verifier software reports its results.



Report

Generate report of the results

Report file name:

Include screen shots and plots

Display report

Report

This group contains controls that enable you to specify report options. It contains the following controls.

Generate report of the results. If selected, this option causes the Simulink Design Verifier software to save the HTML report it generates. If you select this option, you must also enable the **Save test harness as model** option (see “Harness model options” on page 5-13).

Enabling this option provides access to the **Report file name**, **Include screen shots and plots**, and **Display report** options.

Report file name. Specifies a file name with which the Simulink Design Verifier software saves the HTML report it generates. Enter a pathname that is either absolute or relative to the directory specified by **Output directory**. This option is accessible only if **Generate report of the results** is selected.

The default value is `$ModelName$_report`, where `$ModelName$` is a token that represents the model name.

Include screen shots and plots. If selected, this option causes the Simulink Design Verifier software to capture and include images in the HTML report it generates after completing its analysis. This option is disabled by default. It is accessible only if **Generate report of the results** is selected.

Display report. If selected, this option causes the Simulink Design Verifier software to display the HTML report it generates after completing its analysis. This option is enabled by default. It is accessible only if **Generate report of the results** is selected.

Saving Simulink® Design Verifier™ Options

The Simulink® Design Verifier™ software stores its options as a configuration set component attached to your model file (see “Configuration Sets” in *Using Simulink®*). To save the values of Simulink Design Verifier options that you specified for your model, simply save your model (see “Saving a Model” in *Using Simulink*).

Generating Test Cases

This chapter describes how you can use the Simulink® Design Verifier™ software to generate test cases for your model. The following sections introduce the notion of test case generation and present an example in which you generate test cases for a simple Simulink® model:

About Test Case Generation (p. 6-2)	Brief overview of test case generation with the Simulink Design Verifier software.
Basic Workflow for Generating Test Cases (p. 6-3)	Outlines a process for generating test cases for your model.
Generating Test Cases Example (p. 6-4)	Provides an example that walks you through the process of generating test cases for a model.

About Test Case Generation

The Simulink® Design Verifier™ software can generate test cases that satisfy your model's coverage objectives, including decision coverage, condition coverage, and modified condition/decision coverage (MC/DC). Test cases assist you in confirming that a model behaves correctly by demonstrating how its blocks execute in different modes. When generating test cases, the Simulink Design Verifier software performs a formal analysis of your model. After completing its analysis, the software produces a report that details its results and a test harness model that contains test cases. Simply review the report and simulate the test harness model to confirm that the test cases achieve your model's coverage objectives.

The Simulink Design Verifier software provides two blocks that allow you to customize test cases for your Simulink® models. Use the Test Objective block to define the values of a signal that a test case must satisfy. Use the Test Condition block to constrain the values of a signal during a Simulink Design Verifier analysis. For more information about these blocks, see Chapter 10, "Block Reference".

The Simulink Design Verifier software also provides two functions that extend the Stateflow® action language, allowing you to customize test cases for your Stateflow charts. These functions behave identically to the Test Objective and Test Condition blocks. Use the following syntax to invoke these functions in a Stateflow chart:

```
dv.test(expr, "{values}")  
dv.condition(expr, "{values}")
```

where `expr` represents the objective or condition, e.g., $x > 0$, and the optional argument `values` specifies the intervals that comprise the test objective or condition. For more information about the `values` argument, see "Specifying Test Objectives" on page 10-19 and "Specifying Test Conditions" on page 10-13.

Basic Workflow for Generating Test Cases

Here is the recommended workflow for generating test cases for your model:

- 1** Ensure that your model is compatible for use with the Simulink® Design Verifier™ software (for an example, see “Checking Compatibility of the Example Model” on page 6-6).
- 2** Optionally, instrument your model with blocks that specify test objectives and test conditions (for an example, see “Customizing Test Generation” on page 6-21).
- 3** Specify Simulink Design Verifier options that control how it generates test cases for your model (for an example, see “Configuring Test Generation Options” on page 6-10).
- 4** Execute the Simulink Design Verifier analysis and review its results (for examples, see “Analyzing the Example Model” on page 6-13 and “Reanalyzing the Example Model” on page 6-25).

See “Generating Test Cases Example” on page 6-4 for an exercise that demonstrates this workflow.

Generating Test Cases Example

In this section...
“About This Example” on page 6-4
“Constructing the Example Model” on page 6-5
“Checking Compatibility of the Example Model” on page 6-6
“Configuring Test Generation Options” on page 6-10
“Analyzing the Example Model” on page 6-13
“Customizing Test Generation” on page 6-21
“Reanalyzing the Example Model” on page 6-25

About This Example

The sections that follow describe a simple Simulink® model, for which you generate test cases that achieve decision coverage. This example will help you understand the test generation capabilities of the Simulink® Design Verifier™ software.

The following workflow guides you through the process of completing this example:

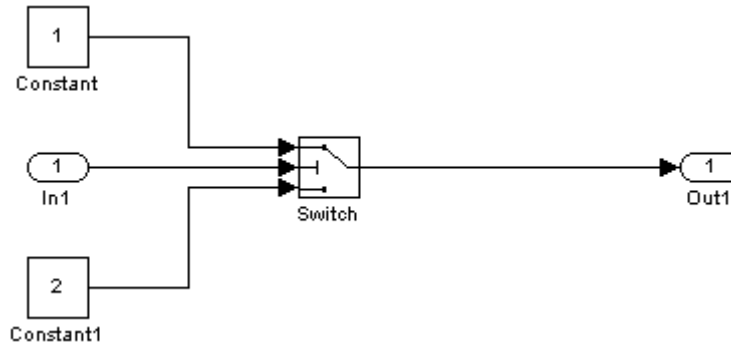
Task	Description	See...
1	Construct the example model.	“Constructing the Example Model” on page 6-5
2	Ensure your model’s compatibility with the Simulink Design Verifier software.	“Checking Compatibility of the Example Model” on page 6-6
3	Configure the Simulink Design Verifier software to generate tests.	“Configuring Test Generation Options” on page 6-10
4	Generate test cases for your model and interpret the results.	“Analyzing the Example Model” on page 6-13

Task	Description	See...
5	Add a Test Condition block to customize test generation.	“Customizing Test Generation” on page 6-21
6	Generate test cases for your modified model and interpret the results.	“Reanalyzing the Example Model” on page 6-25

Constructing the Example Model

This section presents Task 1 of the process that describes how to generate test cases with the Simulink Design Verifier software. In this task, you construct a simple Simulink model that you use throughout the remaining tasks. To complete this task, perform the following steps:

- 1 Create an empty Simulink model (see “Creating a New Model” in the Simulink documentation for help with this step).
- 2 Copy the following blocks into your empty model window (see “Adding Blocks to Your Model” in the Simulink documentation for help with this step):
 - An Inport block, from the Sources library, to initiate the input signal whose value the Simulink Design Verifier software controls
 - A Switch block to provide simple logic, from the Signal Routing library
 - Two Constant blocks to serve as Switch block data inputs, from the Sources library
 - An Outport block to receive the output signal, from the Sinks library
- 3 Double-click one of the Constant blocks in your model and specify its **Constant value** parameter as 2.
- 4 Connect the blocks such that your model appears similar to the following (see “Connecting Blocks in the Model Window” in the Simulink documentation for help with this step):



- 5 Save your model as `example.mdl` (see “Saving a Model” in the Simulink documentation for help with this step).

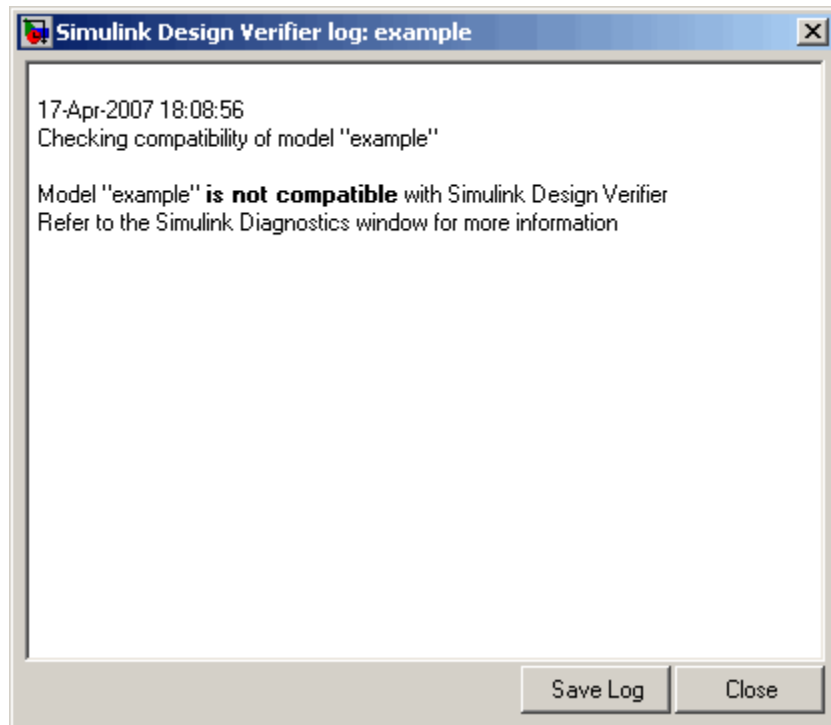
What to do next: Now you are ready to begin Task 2 of this example, “Checking Compatibility of the Example Model” on page 6-6.

Checking Compatibility of the Example Model

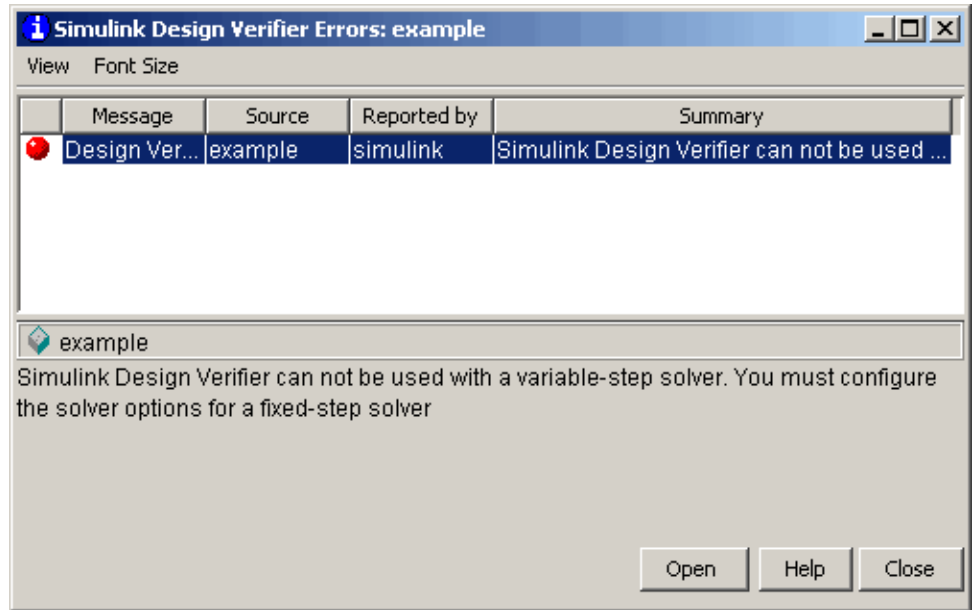
This section presents Task 2 of the process that describes how to generate test cases with the Simulink Design Verifier software. In this task, you ensure that a model is compatible for use with the Simulink Design Verifier software. Specifically, you check the compatibility of the simple Simulink model that you created in the previous task (see “Constructing the Example Model” on page 6-5). To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Check Model Compatibility**.

The Simulink Design Verifier software displays the following log window, which indicates that your model is incompatible:



It also displays the following incompatibility error in the Simulation Diagnostics Viewer:



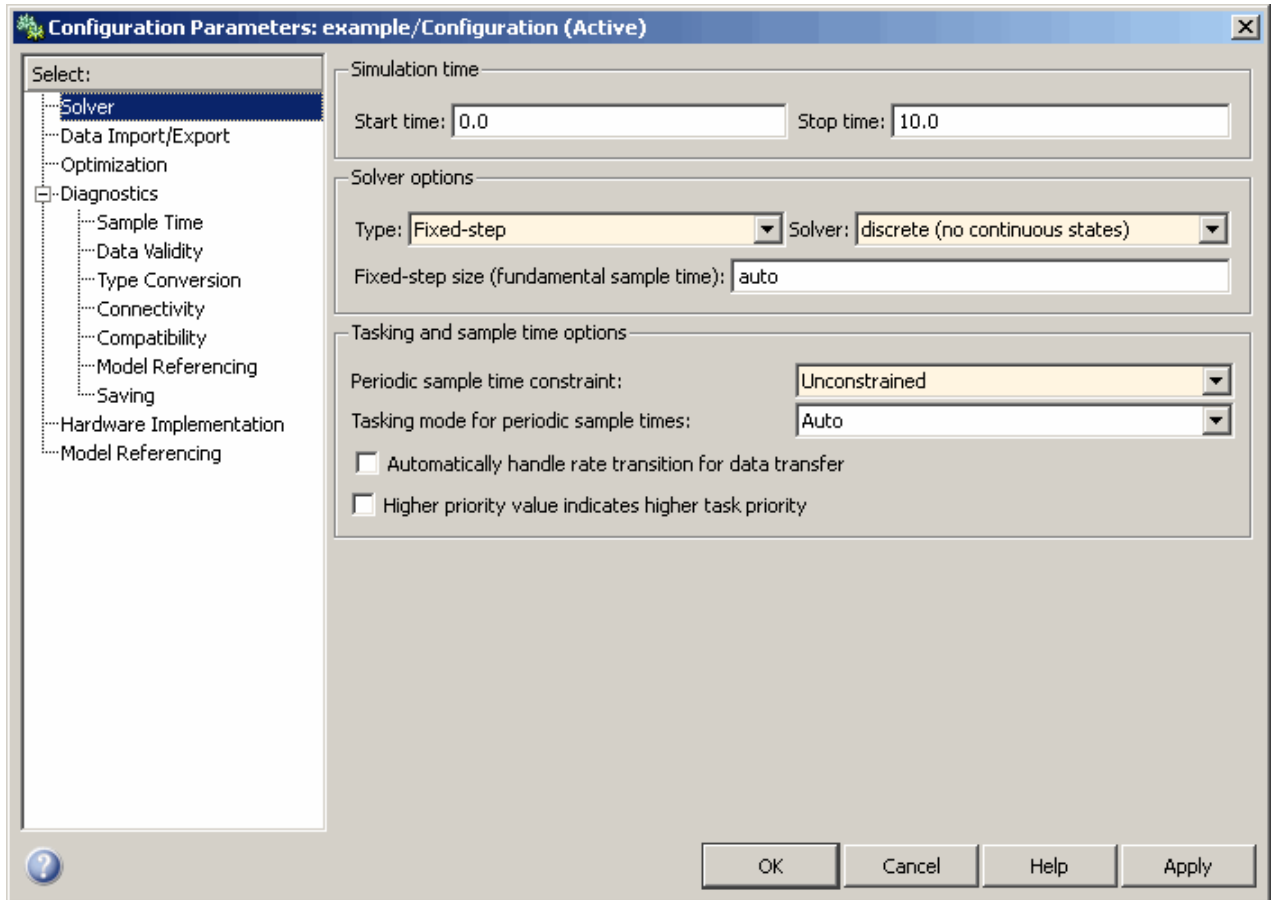
The error message informs you that the Simulink Design Verifier software does not support variable-step solvers. To work around this incompatibility, you must use a fixed-step solver.

- 2** In your Simulink model window, select **Simulation > Configuration Parameters**.

The Configuration Parameters dialog box appears.

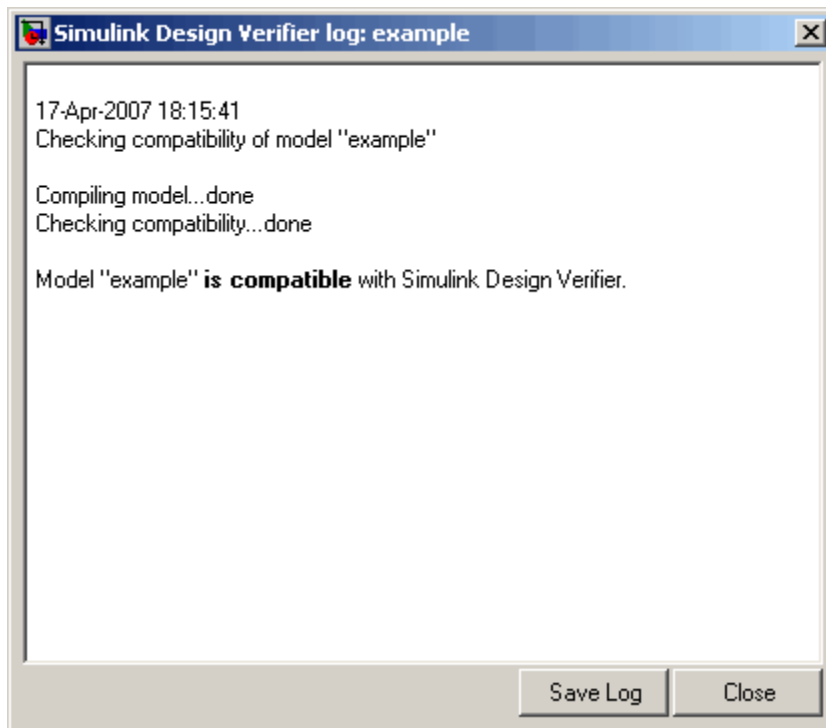
- 3** In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Solver** category (if not already selected). Under **Solver options** on the right side, set the **Type** option to Fixed-step, and then set the **Solver** option to discrete (no continuous states).

The Configuration Parameters dialog box appears as follows:



- 4 Click the **OK** button to apply your changes and close the Configuration Parameters dialog box.
- 5 Recheck the compatibility of your model. In your Simulink model window, select **Tools > Design Verifier > Check Model Compatibility**.

The Simulink Design Verifier software displays the following log window, which confirms that your model is compatible for analysis:



What to do next: Now you are ready to begin Task 3 of this example, “Configuring Test Generation Options” on page 6-10.

Configuring Test Generation Options

This section presents Task 3 of the process that describes how to generate test cases with the Simulink Design Verifier software. In this task, you configure the Simulink Design Verifier software to generate test cases that achieve complete decision coverage for the simple Simulink model that you created in a previous task (see “Constructing the Example Model” on page 6-5). To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Options** (see “Viewing Simulink® Design Verifier™ Options” on page 5-2 for help with this step).

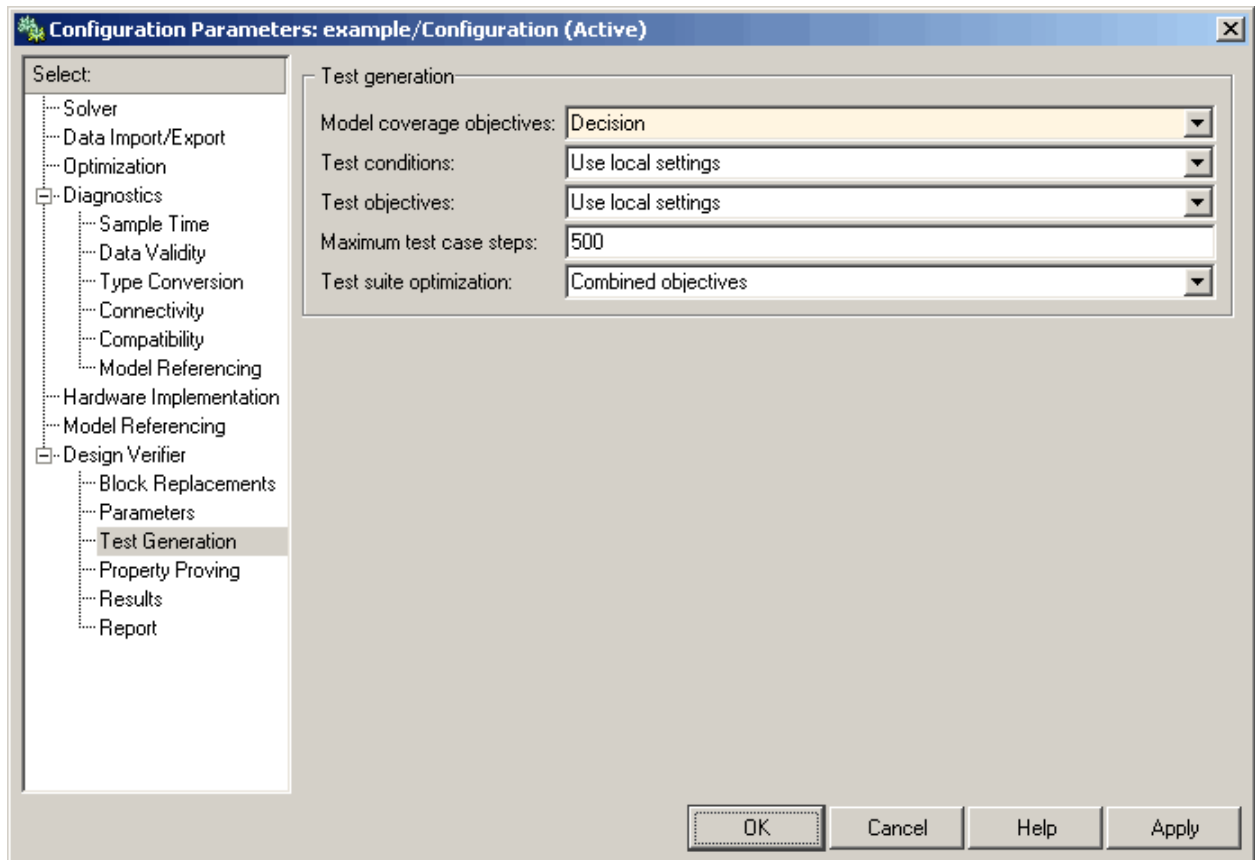
The Simulink Design Verifier software displays its options in the Configuration Parameters dialog box.

- 2** In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Design Verifier** category (if not already selected). Under **Analysis options** on the right side, ensure that the **Mode** option specifies Test generation.
- 3** In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Test Generation** category.

The Configuration Parameters dialog box displays the **Test Generation** pane.

- 4** On the **Test Generation** pane, specify the value of the **Model coverage objectives** parameter as Decision.

The Configuration Parameters dialog box appears as follows:



5 Click **OK** to apply your change and close the Configuration Parameters dialog box.

Note Using the **Test Generation** pane, you can optionally specify values for other parameters that control how the Simulink Design Verifier software generates test cases for your model. See “Test Generation Pane” on page 5-9 for more information.

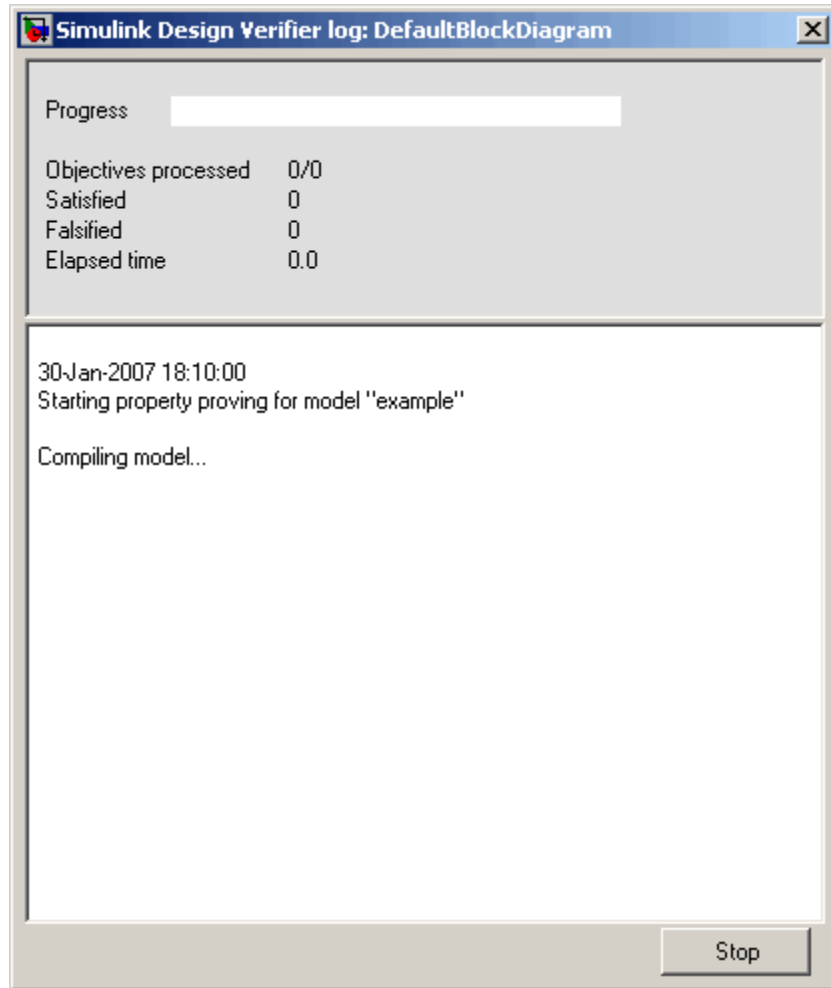
What to do next: Now you are ready to begin Task 4 of this example, “Analyzing the Example Model” on page 6-13.

Analyzing the Example Model

This section presents Task 4 of the process that describes how to generate test cases with the Simulink Design Verifier software. In this task, you execute the Simulink Design Verifier analysis, which you configured in the previous task (see “Configuring Test Generation Options” on page 6-10). The Simulink Design Verifier software generates test cases for your example model and produces results for you to interpret. To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Generate Tests**.

The Simulink Design Verifier software begins analyzing your model to generate test cases. During its analysis, the software displays a log window.



The Simulink Design Verifier log window updates you on the progress of the analysis, providing information such as the number of test objectives processed and how many of those objectives were satisfied. Also, this dialog box includes a **Stop** button that you can click to terminate the proof at anytime.

When the Simulink Design Verifier software completes its analysis, it displays the following items:

- Simulink Design Verifier report — The software displays an HTML report named `example_report.html`.
- Test harness — The software displays a harness model named `example_harness.mdl`.

The remaining steps in this section help you interpret the results that you obtained.

- 2 Review the Simulink Design Verifier report. The report includes the following **Table of Contents** whose items you can click to navigate to particular chapters and sections:

Table of Contents

- [1. Summary](#)
- [2. Test Objectives](#)
 - [Status](#)
 - [example](#)
- [3. Test Cases](#)
 - [Test Case 1](#)
 - [Test Case 2](#)
- [4. Approximations](#)

List of Tables

- 2.1. [Objectives Satisfied](#)

- a In the **Table of Contents**, click Summary.

The report displays its Summary chapter, which begins as follows:

Chapter 1. Summary

Input Model

File: C:\example.mdl
 Version: 1.1
 Time Stamp: Sat Jul 14 15:05:08 2007
 Author: scowan

Analysis Information

Design Verifier Version: 1.1
 Total Analysis Time: 0.13 secs
 Status: Completed normally
[Approximations:](#) 1
[Objectives Satisfied:](#) 2
 Objectives Satisfied - No Test Case: 0
 Objectives Proven Unsatisfiable: 0
 Objectives Undecided: 0
 Objectives Producing Errors: 0

The Summary chapter provides an overview of the analysis results. In particular, the Simulink Design Verifier software satisfied two test objectives in your model.

- b** In the Summary chapter under **Analysis Information**, click Objectives Satisfied.

The report displays its Objectives Satisfied table in the Test Objectives chapter.

Table 2.1. Objectives Satisfied

#:	Type	Model Item	Description
1	Decision	Switch	Switch "Switch": trigger >= threshold false (output is from 3rd input port)
2	Decision	Switch	Switch "Switch": trigger >= threshold true (output is from 1st input port)

This table lists the test objectives that the Simulink Design Verifier software satisfied. Specifically, it describes the test objectives that provide decision coverage for a Switch block. You can locate the model item by clicking Switch; the software highlights the corresponding Switch block in your model window.

- c In the Objectives Satisfied table under the # column, click 1.

The report displays additional information about test objective 1.

example

Objectives of: Switch

#:	Status	Test Cases	Description
1	Satisfied	TC 2	false (output is from 3rd input port)
2	Satisfied	TC 1	true (output is from 1st input port)

This table informs you that the Simulink Design Verifier software satisfied both test objectives associated with the Switch block in your model, for which it generated two test cases.

- d Under the **Test Cases** column of the table, click TC 2.

The report displays its Test Case 2 section.

Test Case 2

Summary

Length: 0.2 Seconds (2 sample periods)

Objective Count: 1

Objectives Reached At:

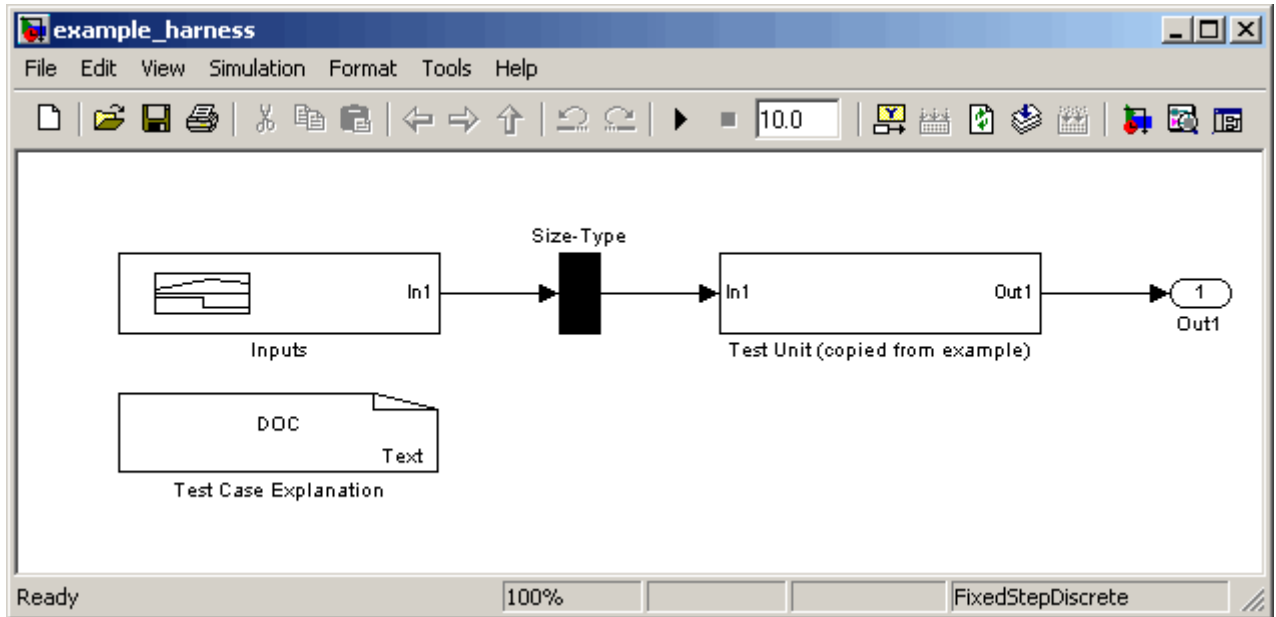
Step	Time	Objectives
1	0	1

Generated Input Data.

Time 0	
Step 1	
In1	-1

This section provides details about a test case that the Simulink Design Verifier software generated to achieve an objective in your model. This test case achieves test objective 1, which involves the Switch block passing its third input. Specifically, the software determined that a value of -1 for the Switch block control signal enables the block to pass its third input.

- 3 Review the harness model named `example_harness.mdl`, which appears as follows:



The harness model contains the following items:

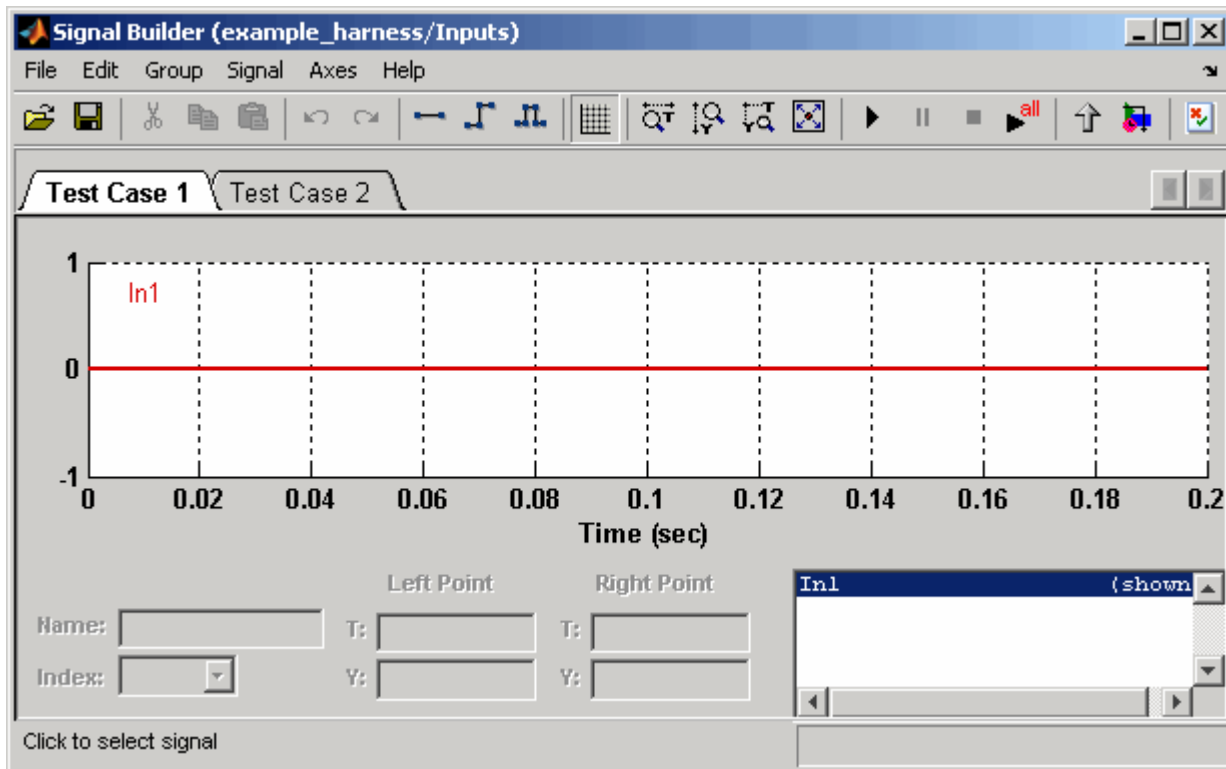
- Signal Builder block named **Inputs** — Contains groups of signals that achieve test objectives in your model.
- Subsystem block named **Test Unit** — Contains a copy of your model.
- DocBlock named **Test Case Explanation** — Provides a textual description of the test cases that the Simulink Design Verifier software generates.


Note See the *Simulink Reference* for more information about interacting with blocks such as the Signal Builder, Subsystem, and DocBlock.

To simulate the test harness and confirm that the test cases achieve complete decision coverage:

- Double-click the **Inputs** block.



The Signal Builder dialog box displays the test case signals.



- b In the Signal Builder dialog box, click the **Run all** button .

The Simulink software simulates the test harness using all the test cases, collects model coverage information, and displays a coverage report whose Summary section appears as follows:

Summary

Model Hierarchy/Complexity:	Test 1
	D1
1. example_harness	2 100% 
2. . . . Test Unit (copied from example)	1 100% 

The coverage report indicates the Simulink Design Verifier software generated test cases that achieve complete decision coverage for your example model (see “Understanding Model Coverage Reports” in the *Simulink® Verification and Validation™ User’s Guide*).

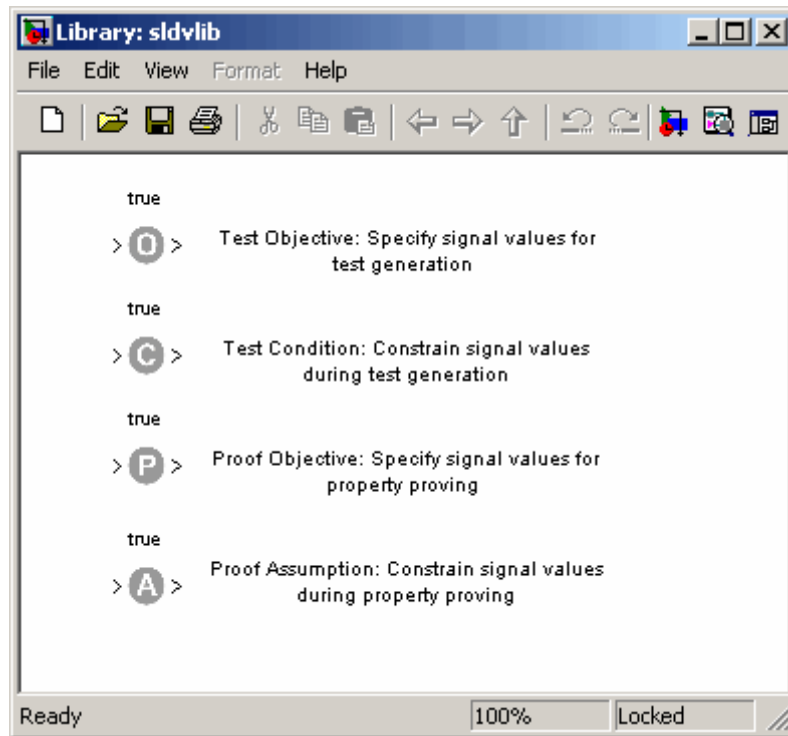
What to do next: Now you are ready to begin Task 5 of this example, “Customizing Test Generation” on page 6-21.

Customizing Test Generation

This section presents Task 5 of the process that describes how to generate test cases with the Simulink Design Verifier software. In this task, you modify the simple Simulink model for which you attained complete decision coverage in the previous task (see “Analyzing the Example Model” on page 6-13). Specifically, you customize test generation by adding and configuring a Test Condition block. To complete this task, perform the following steps:

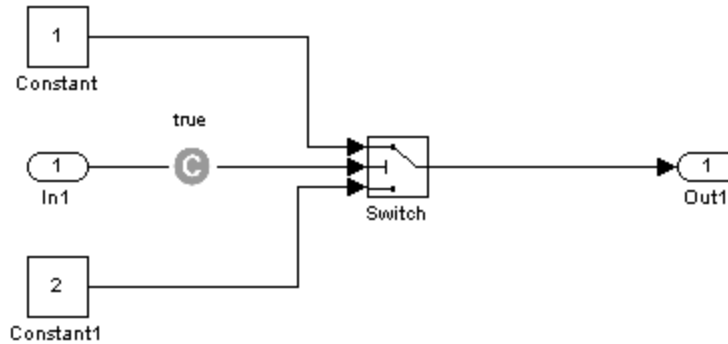
- 1 In the MATLAB® Command Window, enter `sldvlib`.

The Simulink Design Verifier library appears.



- 2 Copy the Test Condition block to your model by dragging it from the Simulink Design Verifier library to your model window.
- 3 In your model window, insert the Test Condition block between the Switch and Output blocks (see "Inserting Blocks in a Line" in the Simulink documentation for help with this step).

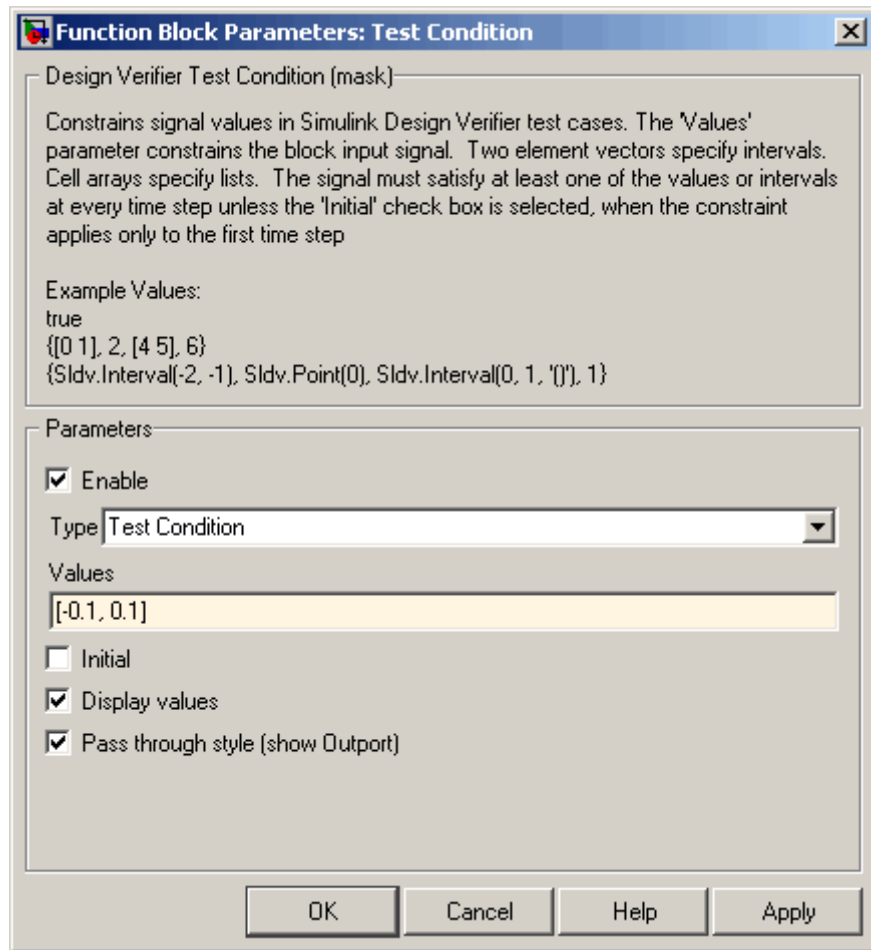
Your model should look like this:



- 4 Double-click the Test Condition block in your model to access its attributes.

The Test Condition block parameter dialog box appears.

- 5 In the **Values** box, enter $[-0.1, 0.1]$. When generating test cases for this model, the Simulink Design Verifier software will constrain the signal values entering the Switch block control port to the specified interval.



6 Click **OK** to apply your changes and close the Test Condition block parameter dialog box.

What to do next: Now you are ready to begin Task 6 of this example, “Reanalyzing the Example Model” on page 6-25.

Reanalyzing the Example Model

This section presents Task 6 of the process that describes how to generate test cases with the Simulink Design Verifier software. In this task, you execute the Simulink Design Verifier analysis on the simple Simulink model that you modified in the previous task (see “Customizing Test Generation” on page 6-21). To observe how a Test Condition block might affect test generation, compare the result of this analysis to the result that you obtained in a previous task (see “Analyzing the Example Model” on page 6-13). To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Generate Tests**.

The Simulink Design Verifier software displays a log window and begins analyzing your model to generate test cases.

When the software completes the analysis, it displays a new Simulink Design Verifier report named `example_report1.html`.

- 2 Review the Simulink Design Verifier report.
 - a In the **Table of Contents**, click Summary.

The report displays its Summary chapter, which begins as follows:

Chapter 1. Summary

Input Model

File: C:\example.mdl
Version: 1.2
Time Stamp: Sat Jul 14 15:15:31 2007
Author: scowan

Analysis Information

Design Verifier Version: 1.1
Total Analysis Time: 0.07 secs
Status: Completed normally
[Approximations:](#) 1
[Objectives Satisfied:](#) 2
Objectives Satisfied - No Test Case: 0
Objectives Proven Unsatisfiable: 0
Objectives Undecided: 0
Objectives Producing Errors: 0

The Summary chapter indicates that the Simulink Design Verifier software satisfied two test objectives in your model.

- b** In the Summary chapter under **Analysis Information**, click **Objectives Satisfied**.

The report displays its Objectives Satisfied table in the Test Objectives chapter.

Table 2.1. Objectives Satisfied

#:	Type	Model Item	Description
1	Decision	Switch	Switch "Switch": trigger >= threshold false (output is from 3rd input port)
2	Decision	Switch	Switch "Switch": trigger >= threshold true (output is from 1st input port)

With the following active constraints:

Name	Constraint
Test Condition	[-0.1, 0.1]

This table lists the test objectives that the Simulink Design Verifier software satisfied. It also identifies any active constraints that the software encountered during its analysis. Consequently, this section lists the Test Condition block that you added in the previous task to constrain the value of the Switch block control signal to the interval [-0.1, 0.1].

- c In the Objectives Satisfied table under the # column, click 1.

The report displays additional information about test objective 1, as shown here.

example

Objectives of: Switch

#:	Status	Test Cases	Description
1	Satisfied	TC 2	false (output is from 3rd input port)
2	Satisfied	TC 1	true (output is from 1st input port)

This table informs you that the Simulink Design Verifier software satisfied both test objectives associated with the Switch block in your model, for which it generated two test cases.

- d Under the **Test Cases** column of the table, click TC 2.

The report displays its Test Case 2 section, which appears as follows:

Test Case 2

Summary

Length: 0.2 Seconds (2 sample periods)

Objective Count: 1

Objectives Reached At:

Step	Time	Objectives
1	0	1

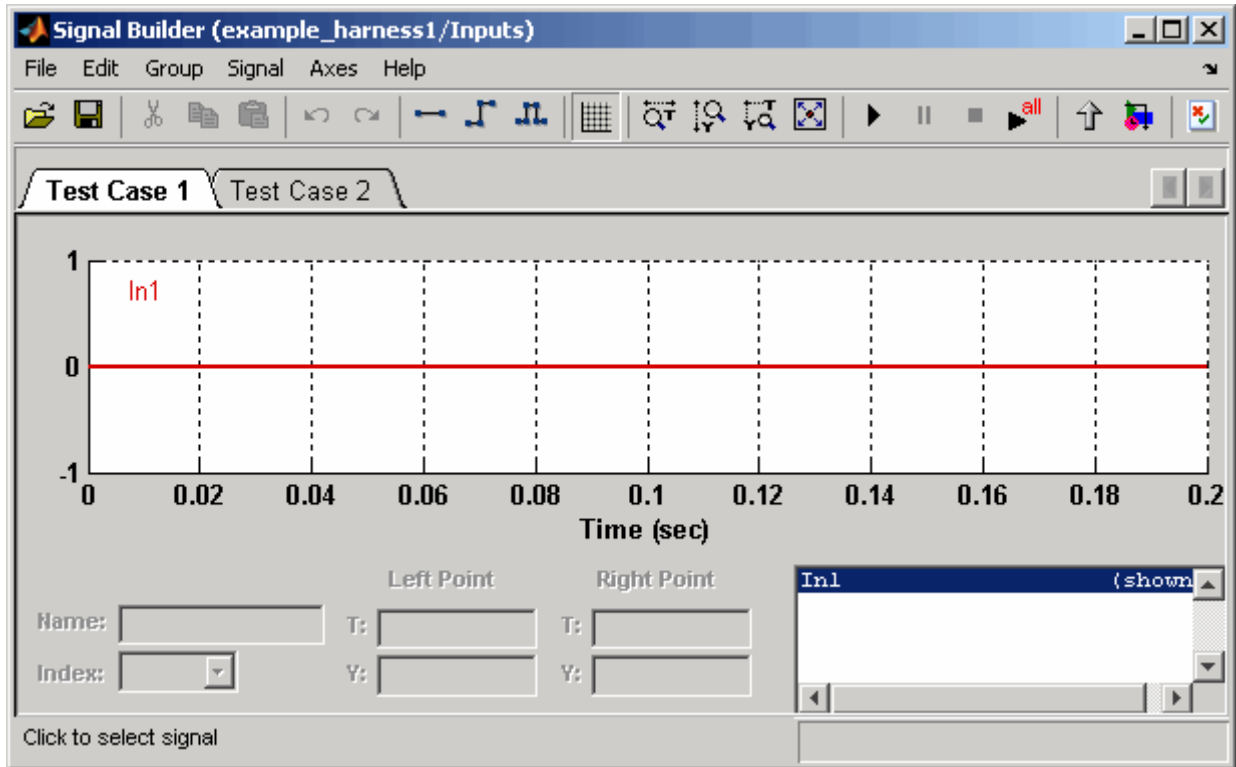
Generated Input Data.


Time 0	
Step 1	
In1	-0.05

This section provides details about a test case that the Simulink Design Verifier software generated to achieve an objective in your model. This test case achieves test objective 1, which involves the Switch block passing its third input. Although the Test Condition block restricted the domain of input signals to the interval $[-0.1, 0.1]$, the software determined that a value of -0.05 for the Switch block control signal satisfies the objective.

- 3 Simulate the harness model named `example_harness1.mdl` and confirm that the test case achieves complete decision coverage:
 - a Double-click the Inputs block.



The Signal Builder dialog box displays the test case signals.



- b** In the Signal Builder dialog box, click the **Run all** button .

The Simulink software simulates the test harness using both test cases, collects model coverage information, and displays a coverage report whose Summary section appears as follows:

Summary

Model Hierarchy/Complexity:		Test 1
		D1
1. example_harness1	3	100% 
2. ... Test Unit (copied from another example)	2	100% 

The coverage report indicates the Simulink Design Verifier software generated test cases that achieve complete decision coverage for your example model.

Proving Properties of a Model

This chapter describes how you can use the Simulink® Design Verifier™ software to prove properties of your model. The following sections introduce the notion of property proofs and present an example in which you prove a property of a simple Simulink® model:

About Property Proofs (p. 7-2)

Brief overview of proving properties with the Simulink Design Verifier software.

Basic Workflow for Proving Model Properties (p. 7-3)

Outlines a process for proving properties of your model.

Proving Model Properties Example (p. 7-4)

Provides an example that walks you through the process of proving model properties.

About Property Proofs

The Simulink® Design Verifier™ software can prove properties of your model. Here, the term *property* refers to a logical expression of signal values in a model. For example, you can specify that a signal in your model should attain a particular value or range of values during simulation. You can then use the Simulink Design Verifier software to prove whether such properties are valid. The software performs a formal analysis of your model to prove or disprove the specified properties. If the software disproves a property, it provides a counterexample that demonstrates a property violation.

The Simulink Design Verifier software provides two blocks that allow you to specify properties in your Simulink® models. Use the Proof Objective block to define the values of a signal that the Simulink Design Verifier software will prove. Use the Proof Assumption block to constrain the values of a signal during a proof. For more information about these blocks, refer to Chapter 10, “Block Reference”.

Note Blocks from the Model Verification library in the Simulink software behave like a Proof Objective block during Simulink Design Verifier proofs. Hence, you can use Assertion blocks and other Model Verification blocks to specify properties of your model. See “Model Verification” in the *Simulink Reference* for more information about these blocks.

The Simulink Design Verifier software also provides two functions that extend the Stateflow® action language, allowing you to specify properties in your Stateflow charts. These functions behave identically to the Proof Objective and Proof Assumption blocks. Use the following syntax to invoke these functions in a Stateflow chart:

```
dv.prove(expr, "{values}")  
dv.assume(expr, "{values}")
```

where *expr* represents the objective or assumption, e.g., $x > 0$, and the optional argument *values* specifies the intervals that comprise the proof objective or assumption. For more information about the *values* argument, see “Specifying Proof Objectives” on page 10-8 and “Specifying Proof Assumptions” on page 10-2.

Basic Workflow for Proving Model Properties

Here is the recommended workflow for proving properties of your model:

- 1** Ensure that your model is compatible for use with the Simulink® Design Verifier™ software (for an example, see “Checking Compatibility of the Example Model” on page 7-6).
- 2** Instrument your model with blocks that specify proof objectives and proof assumptions (for examples, see “Instrumenting the Example Model” on page 7-10 and “Customizing the Example Proof” on page 7-23).
- 3** Specify Simulink Design Verifier options that control how it proves the properties of your model (for an example, see “Configuring Property Proving Options” on page 7-13).
- 4** Execute the Simulink Design Verifier analysis and review its results (for examples, see “Analyzing the Example Model” on page 7-15 and “Reanalyzing the Example Model” on page 7-25).

See “Proving Model Properties Example” on page 7-4 for an exercise that demonstrates this workflow.

Proving Model Properties Example

In this section...
“About This Example” on page 7-4
“Constructing the Example Model” on page 7-5
“Checking Compatibility of the Example Model” on page 7-6
“Instrumenting the Example Model” on page 7-10
“Configuring Property Proving Options” on page 7-13
“Analyzing the Example Model” on page 7-15
“Customizing the Example Proof” on page 7-23
“Reanalyzing the Example Model” on page 7-25

About This Example

The sections that follow describe a simple Simulink® model, for which you prove a property that you specify using a Proof Objective block. This example will help you understand the property proving capabilities of the Simulink® Design Verifier™ software.

The following workflow guides you through the process of completing this example:

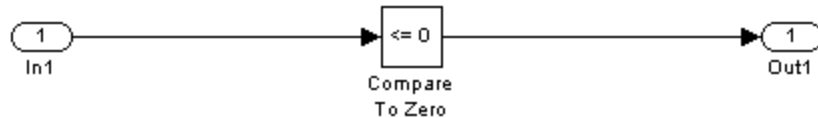
Task	Description	See...
1	Construct the example model.	“Constructing the Example Model” on page 7-5
2	Ensure your model’s compatibility with the Simulink Design Verifier software.	“Checking Compatibility of the Example Model” on page 7-6
3	Add a Proof Objective block to your model to prepare for its proof.	“Instrumenting the Example Model” on page 7-10

Task	Description	See...
4	Configure the Simulink Design Verifier software to prove properties.	“Configuring Property Proving Options” on page 7-13
5	Prove a property of your model and interpret the results.	“Analyzing the Example Model” on page 7-15
6	Add a Proof Assumption block to customize the proof.	“Customizing the Example Proof” on page 7-23
7	Prove a property of your modified model and interpret the results.	“Reanalyzing the Example Model” on page 7-25

Constructing the Example Model

This section presents Task 1 of the process that describes how to implement an example proof with the Simulink Design Verifier software. In this task, you construct a simple Simulink model that you use throughout the remaining tasks. To complete this task, perform the following steps:

- 1** Create an empty Simulink model (see “Creating a New Model” in the Simulink documentation for help with this step).
- 2** Copy the following blocks into your empty model window (see “Adding Blocks to Your Model” in the Simulink documentation for help with this step):
 - An Inport block, from the Sources library, to initiate the input signal whose value the Simulink® Design Verifier software controls
 - A Compare To Zero block to provide simple logic, from the Logic and Bit Operations library
 - An Outport block to receive the output signal, from the Sinks library
- 3** Connect these blocks such that your model appears similar to the following (see “Connecting Blocks in the Model Window” in the Simulink documentation for help with this step):



- 4 Save your model as `example.mdl` (see “Saving a Model” in the Simulink documentation for help with this step).

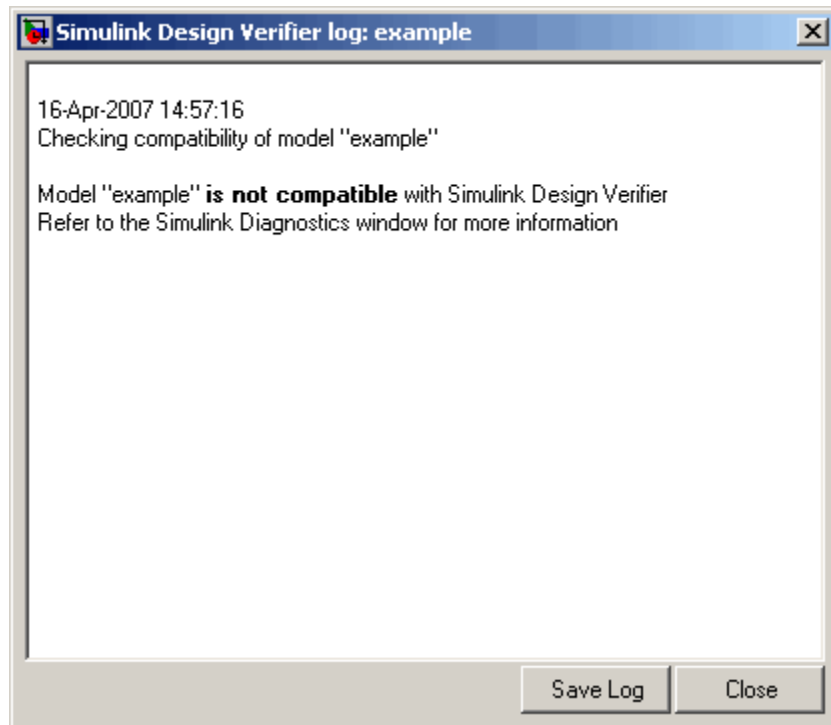
What to do next: Now you are ready to begin Task 2 of this example, “Checking Compatibility of the Example Model” on page 7-6.

Checking Compatibility of the Example Model

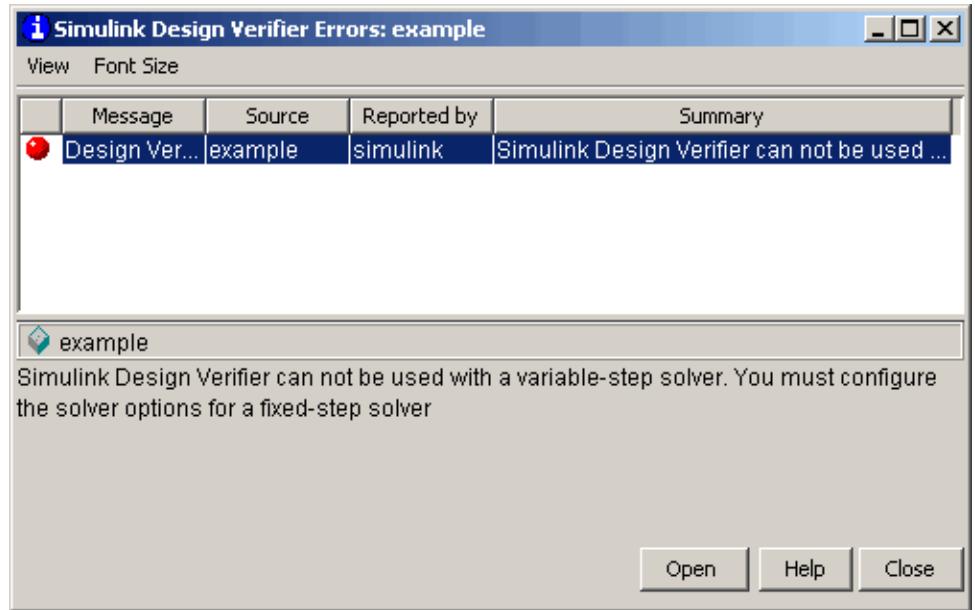
This section presents Task 2 of the process that describes how to implement an example proof with the Simulink Design Verifier software. In this task, you ensure that a model is compatible for use with the Simulink Design Verifier software. Specifically, you check the compatibility of the simple Simulink model that you created in the previous task (see “Constructing the Example Model” on page 7-5). To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Check Model Compatibility**.

The Simulink Design Verifier software displays the following log window, which indicates that your model is incompatible:



It also displays the following incompatibility error in the Simulation Diagnostics Viewer:



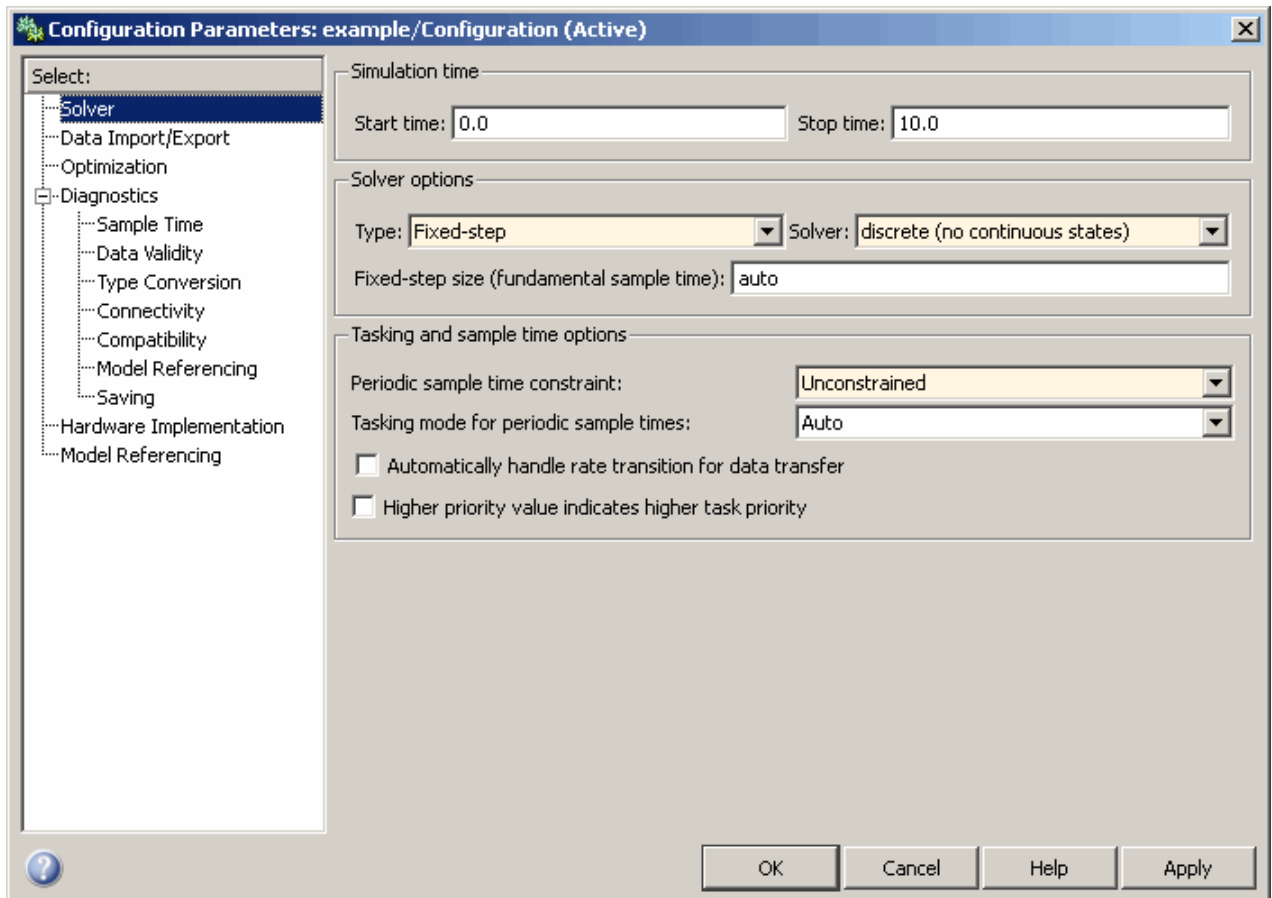
The error message informs you that the Simulink Design Verifier software does not support variable-step solvers. To work around this incompatibility, you must use a fixed-step solver.

- 2** In your Simulink model window, select **Simulation > Configuration Parameters**.

The Configuration Parameters dialog box appears.

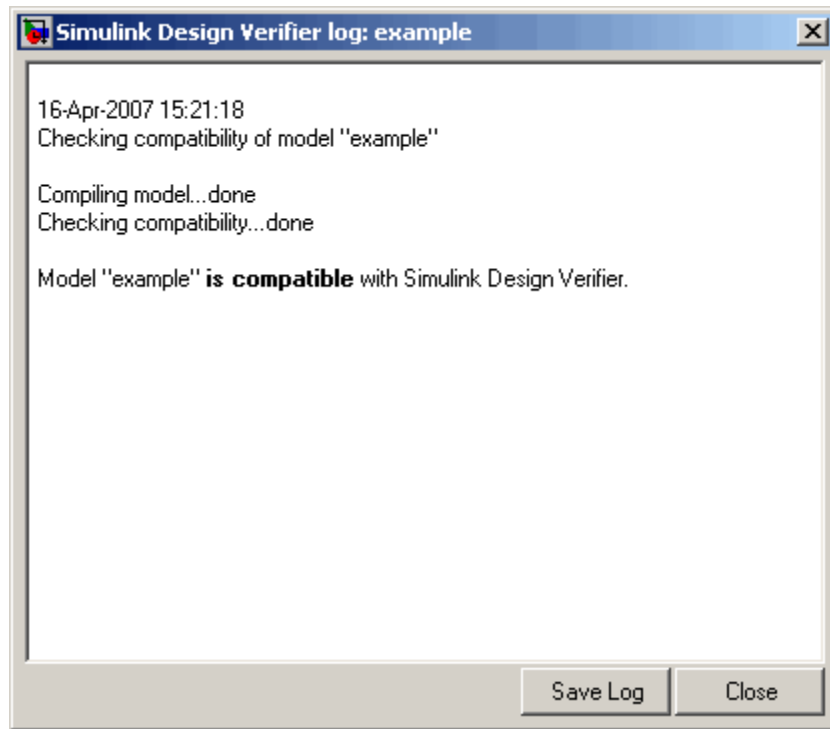
- 3** In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Solver** category (if not already selected). Under **Solver options** on the right side, set the **Type** option to Fixed-step, and then set the **Solver** option to discrete (no continuous states).

The Configuration Parameters dialog box appears as follows:



- 4 Click the **OK** button to apply your changes and close the Configuration Parameters dialog box.
- 5 Recheck the compatibility of your model. In your Simulink model window, select **Tools > Design Verifier > Check Model Compatibility**.

The Simulink Design Verifier software displays the following log window, which confirms that your model is compatible for analysis:



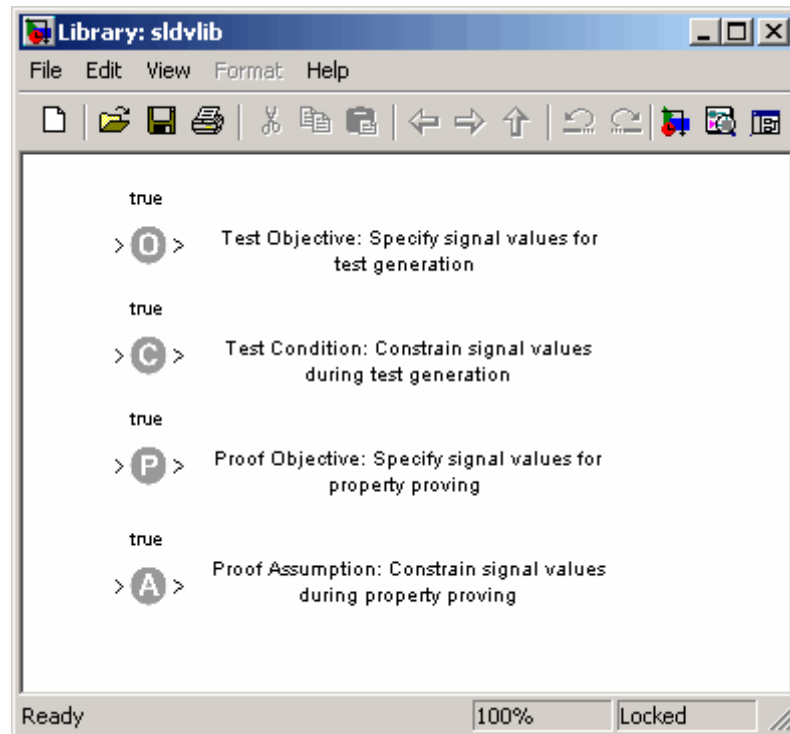
What to do next: Now you are ready to begin Task 3 of this example, “Instrumenting the Example Model” on page 7-10.

Instrumenting the Example Model

This section presents Task 3 of the process that describes how to implement an example proof with the Simulink Design Verifier software. In this task, you prepare a model so that you can prove its properties with the Simulink Design Verifier software. Specifically, you instrument the simple Simulink model that you created in a previous task (see “Constructing the Example Model” on page 7-5) by adding and configuring a Proof Objective block. To complete this task, perform the following steps:

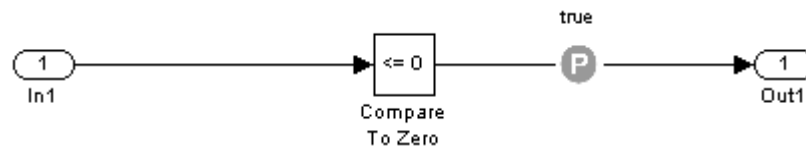
- 1 In the MATLAB® Command Window, enter `sldvlib`.

The Simulink Design Verifier library appears.



- 2** Copy the Proof Objective block to your model by dragging it from the Simulink Design Verifier library to your model window.
- 3** In your model window, insert the Proof Objective block between the Compare To Zero and Outport blocks (see “Inserting Blocks in a Line” in the Simulink documentation for help with this step).

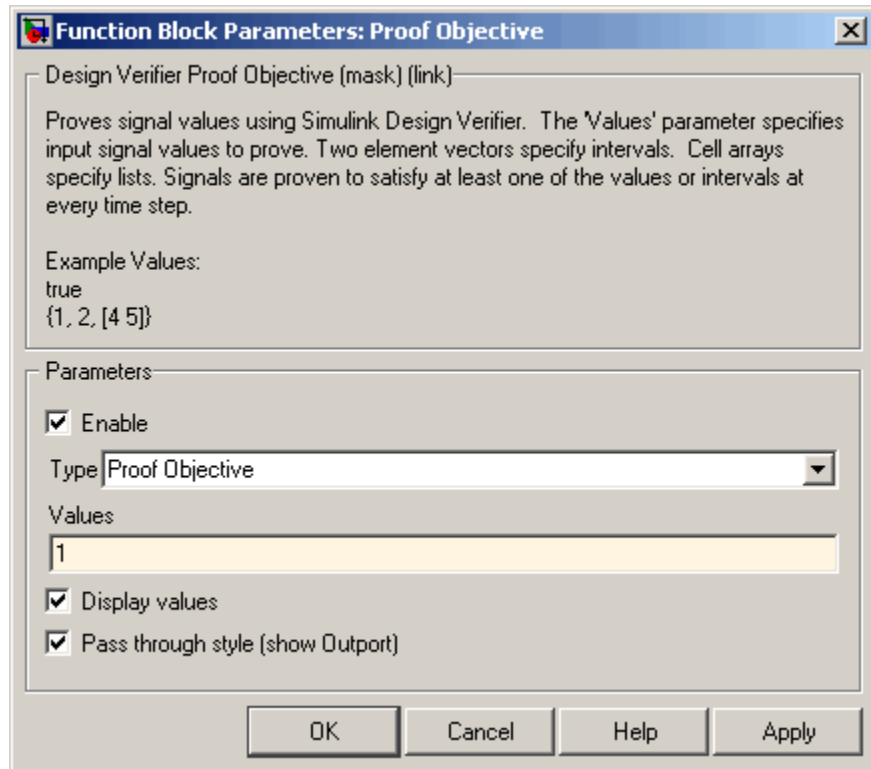
Your model should look like this:



- 4** Double-click the Proof Objective block in your model to access its attributes.

The Proof Objective block parameter dialog box appears.

- 5 In the **Values** box, enter 1. The Simulink Design Verifier software will attempt to prove that the signal output by the Compare To Zero block always attains this value for any signals that it receives.



- 6 Click **OK** to apply your changes and close the Proof Objective block parameter dialog box.

What to do next: Now you are ready to begin Task 4 of this example, “Configuring Property Proving Options” on page 7-13.

Configuring Property Proving Options

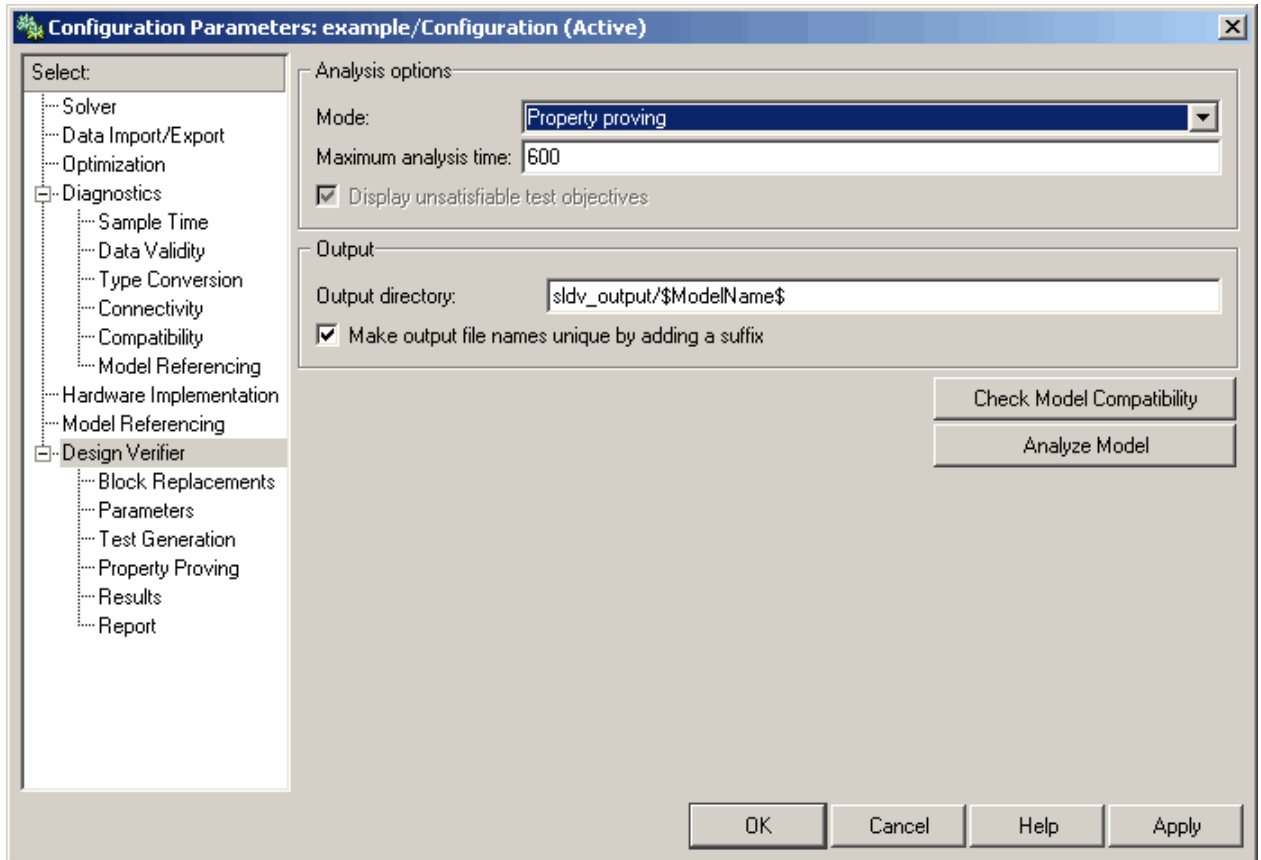
This section presents Task 4 of the process that describes how to implement an example proof with the Simulink Design Verifier software. In this task, you configure the Simulink Design Verifier software to prove properties of the simple Simulink model that you instrumented in the previous task (see “Instrumenting the Example Model” on page 7-10). To complete this task, perform the following steps:

- 1** In your Simulink model window, select **Tools > Design Verifier > Options** (see “Viewing Simulink® Design Verifier™ Options” on page 5-2 for help with this step).

The Simulink Design Verifier software displays its options in the Configuration Parameters dialog box.

- 2** In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Design Verifier** category (if not already selected). Under **Analysis options** on the right side, set the **Mode** option to Property proving.

The Configuration Parameters dialog box appears as follows:



3 Click **OK** to apply your changes and close the Configuration Parameters dialog box.

Note Using the **Property Proving** pane, you can optionally specify values for other parameters that control how the Simulink Design Verifier software proves properties of your model. See “Property Proving Pane” on page 5-11 for more information.

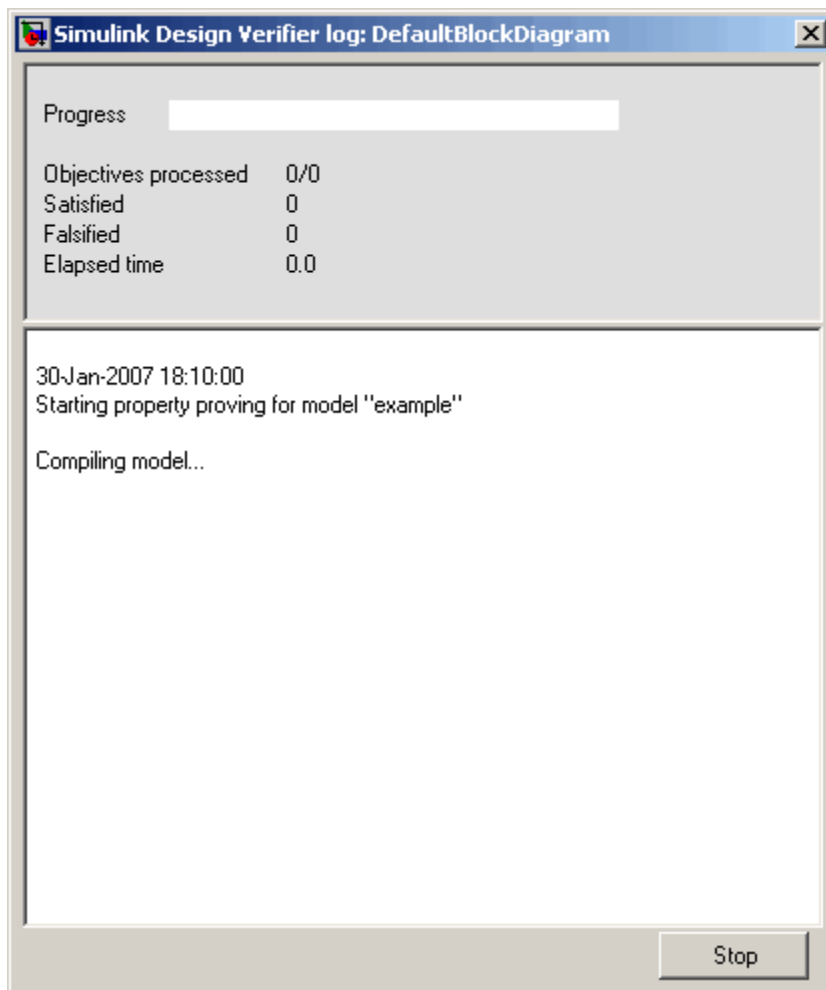
What to do next: Now you are ready to begin Task 5 of this example, “Analyzing the Example Model” on page 7-15.

Analyzing the Example Model

This section presents Task 5 of the process that describes how to implement an example proof with the Simulink Design Verifier software. In this task, you execute the Simulink Design Verifier analysis, which you configured in the previous task (see “Configuring Property Proving Options” on page 7-13). The Simulink Design Verifier software proves a property of your example model and produces results for you to interpret. To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Prove Properties**.

The Simulink Design Verifier software begins analyzing your model to prove its properties. During its analysis, the software displays a log window.



The Simulink Design Verifier log window updates you on the progress of the proof, providing information such as the number of objectives processed and how many of those objectives were either satisfied or falsified. Also, this dialog box includes a **Stop** button that you can click to terminate the proof at anytime.

When the Simulink Design Verifier software completes its analysis, it displays the following items:

- Simulink Design Verifier report — The software displays an HTML report named `example_report.html`.
- Test harness — The software displays a harness model named `example_harness.mdl`.

The remaining steps in this section help you interpret the results that you obtained.

- 2 Review the Simulink Design Verifier report. The report includes the following **Table of Contents** whose items you can click to navigate to particular chapters and sections:

Table of Contents

- [1. Summary](#)
- [2. Proof Objectives](#)
 - [Status](#)
 - [example](#)
- [3. Counterexamples](#)
 - [Counterexample 1](#)
- [4. Approximations](#)

List of Tables

- 2.1. [Objectives Falsified with Counterexamples](#)

- a In the **Table of Contents**, click Summary.

The report displays its Summary chapter, which begins as follows:

Chapter 1. Summary

Input Model

File: C:\example.mdl
 Version: 1.1
 Time Stamp: Sat Jul 14 15:58:26 2007
 Author: scowan

Analysis Information

Design Verifier Version: 1.1
 Total Analysis Time: 0.05 secs
 Status: Completed normally
[Approximations:](#) 1
 Objectives Proven Valid: 0
[Objectives Falsified with Counterexamples:](#) 1
 Objectives Falsified - No Counterexample: 0
 Objectives Undecided: 0
 Objectives Producing Errors: 0

The Summary chapter provides an overview of the analysis results. In particular, the Simulink Design Verifier software identified a counterexample that falsifies an objective in your model.

- b** In the Summary chapter under **Analysis Information**, click **Objectives Falsified with Counterexamples**.

The report displays its Objectives Falsified with Counterexamples table in the Proof Objectives chapter.

Table 2.1. Objectives Falsified with Counterexamples

#:	Type	Model Item	Description
1	Custom Proof Objective	Proof Objective	Proof Objective "Proof Objective" : 1

This table lists the proof objectives that the Simulink Design Verifier software disproved using a counterexample it generated. You can locate the objective in your model window by clicking **Proof Objective**; the software highlights the corresponding **Proof Objective** block in your model window.

- c In the **Objectives Falsified with Counterexamples** table under the **#** column, click 1.

The report displays information about proof objective 1.

example

Objectives of: Proof Objective

#:	Status	Test Cases	Description
1	Falsified	TC 1	: 1

This table informs you that the Simulink Design Verifier software disproved a proof objective that you specified in your model, for which it generated a counterexample.

- d Under the **Test Cases** column of the table, click [TC 1](#).

The report displays its **Counterexample 1** section.

Counterexample 1

Summary

Length: 0.2 Seconds (2 sample periods)

Objective Count: 1

Objectives Reached At:

Objectives Falsified

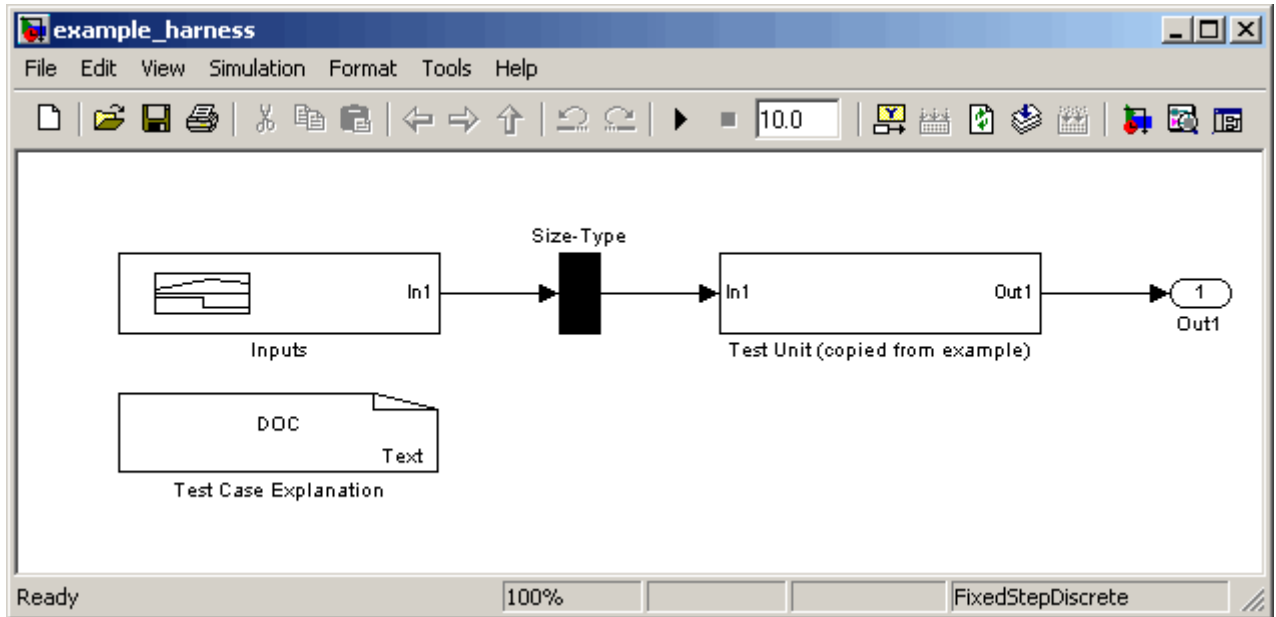
1

Generated Input Data.

Time	0
Step	1
In1	255

This section provides details about the counterexample that the Simulink Design Verifier software generated to disprove an objective in your model. In this counterexample, a signal value of 255 falsifies the objective that you specified using the Proof Objective block in your model. That is, 255 is not less than or equal to 0, which causes the Compare To Zero block to return 0 (false) instead of 1 (true).

- 3 Review the harness model named `example_harness.mdl`, which appears as follows:



The harness model contains the following items:

- Signal Builder block named **Inputs** — Contains groups of signals that falsify proof objectives in your model.
- Subsystem block named **Test Unit** — Contains a copy of your model.
- DocBlock named **Test Case Explanation** — Provides a textual description of the counterexamples that the Simulink Design Verifier software generates.

Note See the *Simulink Reference* for more information about interacting with blocks such as the Signal Builder, Subsystem, and DocBlock.

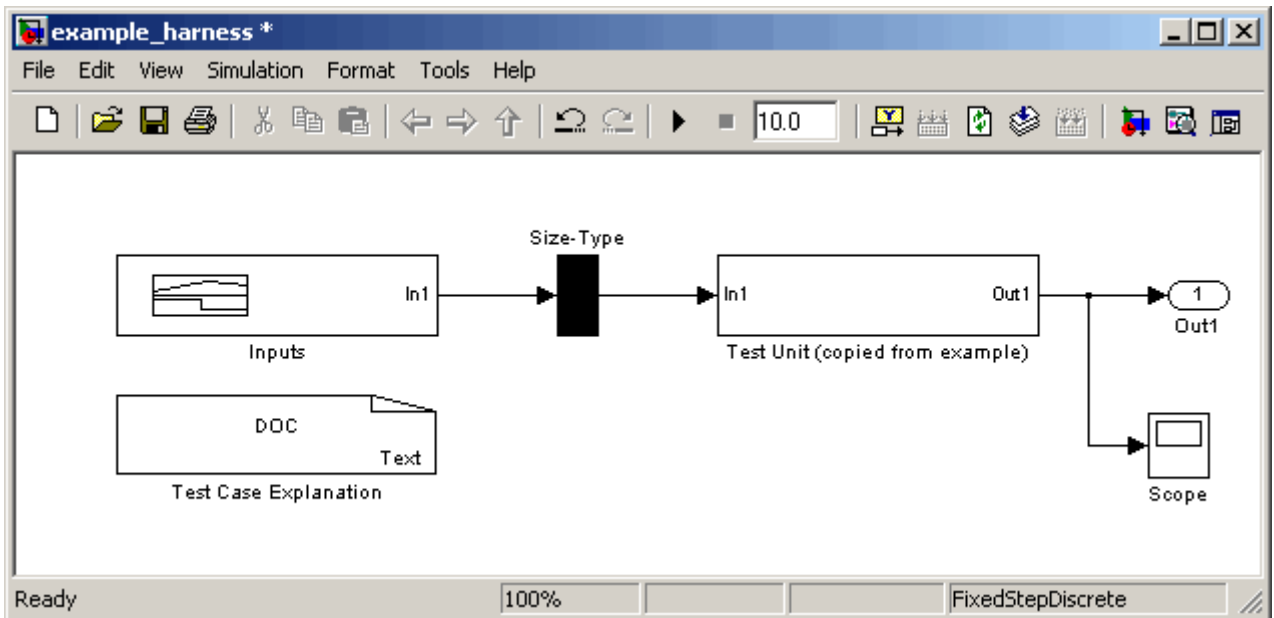
You can simulate the harness model to observe the counterexample that falsifies the proof objective in your model:

- In the MATLAB Command Window, enter `simulink` to open the Simulink library (if not already open).

The Simulink library window appears.

- b** From the Sinks library, copy a Scope block into your harness model window. The Scope block will allow you to see the value of the signal output by the Compare To Zero block in your model.
- c** In your harness model window, connect the output signal of the Test Unit subsystem to the Scope block.

Your model should appear similar to the following:

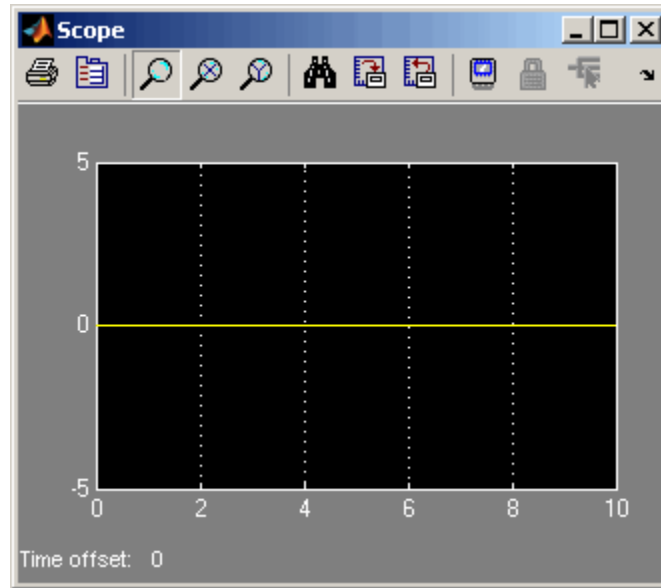


- d** In your harness model window, select **Simulation > Start** to begin the simulation.

The Simulink software simulates the harness model.

- e** In your harness model window, double-click the Scope block to open its display window.

The Scope window appears as follows:



The Scope block displays the value of the signal output by the Compare To Zero block in your model. In this example, the Compare To Zero block returns 0 (false) throughout the simulation. Recall from a previous step (see “Instrumenting the Example Model” on page 7-10) that you specified that the proof objective in your model is 1 (true). Hence, the counterexample that the Signal Builder block supplies falsifies the proof objective.

What to do next: Now you are ready to begin Task 6 of this example, “Customizing the Example Proof” on page 7-23.

Customizing the Example Proof

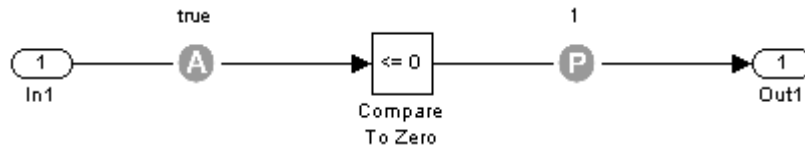
This section presents Task 6 of the process that describes how to implement an example proof with the Simulink Design Verifier software. In this task, you modify the simple Simulink model whose proof objective the Simulink Design Verifier software disproved in the previous task (see “Analyzing the Example Model” on page 7-15). Specifically, you customize the proof by adding and configuring a Proof Assumption block. To complete this task, perform the following steps:

- 1 If the Simulink Design Verifier library is not already open, type `sldvlib` in the MATLAB Command Window.

The Simulink Design Verifier library appears.

- 2 Copy the Proof Assumption block to your model (example.mdl) by dragging it from the Simulink Design Verifier library to your model window.
- 3 In your model window, insert the Proof Assumption block between the Inport and Compare To Zero blocks.

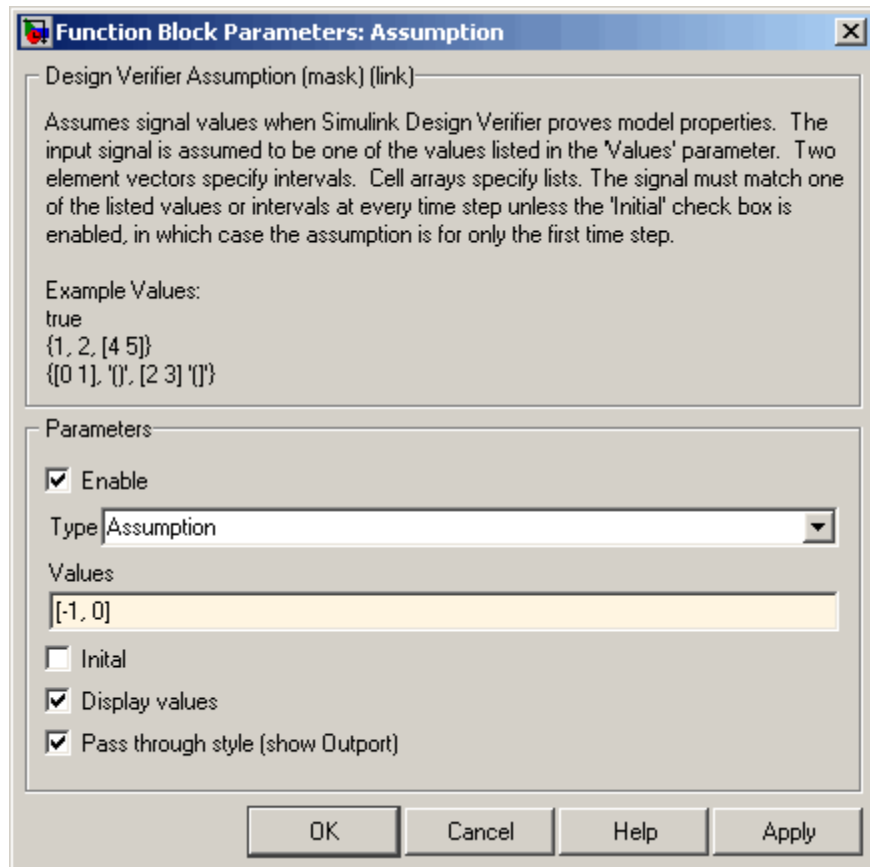
Your model should appear similar to the following:



- 4 Double-click the Proof Assumption block in your model to access its attributes.

The Proof Assumption block parameter dialog box appears.

- 5 In the **Values** box, enter `[-1, 0]`. When proving properties of this model, the Simulink Design Verifier software will constrain the signal values entering the Compare To Zero block to the specified interval.



- 6 Click **OK** to apply your changes and close the Proof Assumption block parameter dialog box.

What to do next: Now you are ready to begin Task 7 of this example, “Reanalyzing the Example Model” on page 7-25.

Reanalyzing the Example Model

This section presents Task 7 of the process that describes how to implement an example proof with the Simulink Design Verifier software. In this task, you execute the Simulink Design Verifier analysis on the simple Simulink model that you modified in the previous task (see “Customizing the Example Proof”

on page 7-23). To observe how Proof Assumption blocks affect proofs, compare the result of this analysis to the result that you obtained in a previous task (see “Analyzing the Example Model” on page 7-15). To complete this task, perform the following steps:

- 1 In your Simulink model window, select **Tools > Design Verifier > Prove Properties**.

The Simulink Design Verifier software displays a log window and begins analyzing your model to prove its properties.

When the software completes the analysis, it displays a new Simulink Design Verifier report named `example_report1.html`.

Note If the Simulink Design Verifier software satisfies all proof objectives in your model, it does not generate a harness model.

- 2 Review the Simulink Design Verifier report.
 - a In the **Table of Contents**, click Summary.

The report displays its Summary chapter, which begins as follows:

Chapter 1. Summary

Input Model

File: C:\example.mdl
 Version: 1.2
 Time Stamp: Sat Jul 14 15:58:26 2007
 Author: scowan

Analysis Information

Design Verifier Version:	1.1
Total Analysis Time:	0.05 secs
Status:	Completed normally
Approximations:	1
Objectives Proven Valid:	1
Objectives Falsified with Counterexamples:	0
Objectives Falsified - No Counterexample:	0
Objectives Undecided:	0
Objectives Producing Errors:	0

The Summary chapter indicates that the Simulink Design Verifier software proved an objective in your model.

- b** In the Summary chapter under **Analysis Information**, click [Objectives Proven Valid](#).

The report displays its Objectives Proven Valid table in the Proof Objectives chapter.

Table 2.1. Objectives Proven Valid

#:	Type	Model Item	Description
1	Custom Proof Objective	Proof Objective	Proof Objective "Proof Objective" : 1

With the following active constraints:

Name	Constraint
Assumption	[-1, 0]

This table lists the proof objectives that the Simulink Design Verifier software proved to be valid. It also identifies any active constraints on which the validity of the objectives depend. Consequently, this section lists the Proof Assumption block that you added in the previous task to constrain the signal value to the interval [-1, 0].

- c In the Objectives Proven Valid table under the # column, click 1.

The report displays information about proof objective 1, as shown here.

example

Objectives of: Proof Objective

#:	Status	Test Cases	Description
1	Proven valid	n/a	: 1

This table informs you that the Simulink Design Verifier software proved an objective that you specified in your model. Because the Proof Assumption block restricted the domain of the input signals to the interval [-1, 0], the software was able to prove that this interval contains no values that are greater than zero, thereby satisfying the proof objective.

Reviewing the Results

The Simulink® Design Verifier™ software produces several artifacts after it analyzes your model. Depending on the analysis, the software can generate a test harness model, a report, and a data file. The following sections illustrate each of these items and describe their contents.

Exploring Test Harness Models
(p. 8-2)

Describes a basic test harness model.

Understanding Simulink® Design
Verifier™ Reports (p. 8-8)

Describes the different parts of a
Simulink Design Verifier report.

Examining Simulink® Design
Verifier™ Data Files (p. 8-23)

Describes the contents of a Simulink
Design Verifier data file.

Exploring Test Harness Models

In this section...
“About Test Harness Models” on page 8-2
“Anatomy of a Test Harness” on page 8-2
“Simulating the Test Harness” on page 8-6

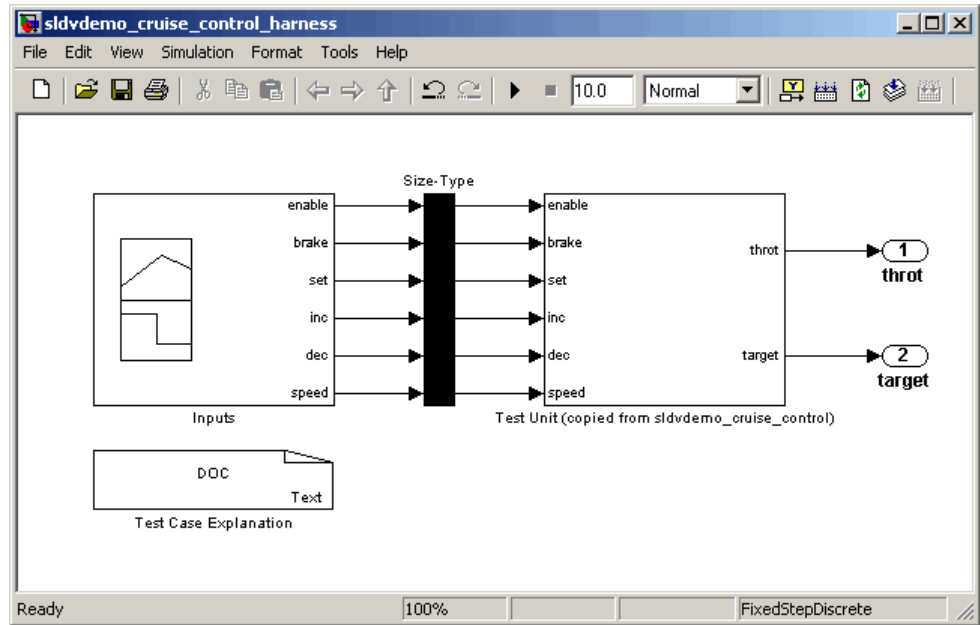
About Test Harness Models

When you enable the **Save test harness as model** parameter (see “Results Pane” on page 5-12), the Simulink® Design Verifier™ software generates a test harness model after it completes its analysis. If the software’s **Mode** parameter specifies **Test generation**, the harness model contains test cases that achieve test objectives. Otherwise, the software’s **Mode** parameter specifies **Property proving** and the harness model contains counterexamples that falsify proof objectives.

Note The Simulink Design Verifier software can generate a harness model only when the top level of the system you are analyzing contains an **Inport** block.

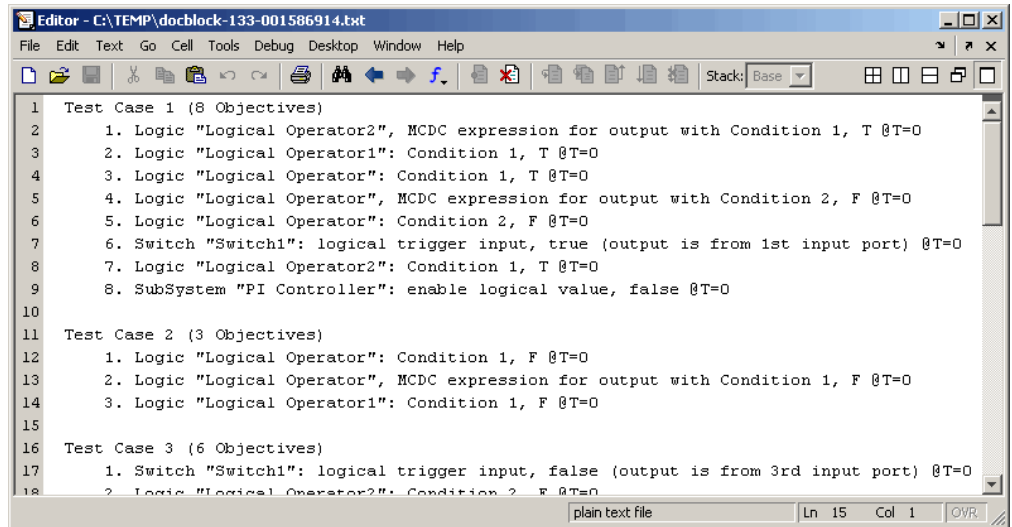
Anatomy of a Test Harness

When the Simulink Design Verifier software completes its analysis, it produces a test harness model that looks like this:



The harness model contains the following items:

- Test Case Explanation** — This DocBlock documents the test cases or counterexamples that the Simulink Design Verifier software generates. Double-click the Test Case Explanation block to view a description of each test case or counterexample. The block lists either the test objectives that each test case achieves or the proof objectives that each counterexample falsifies.

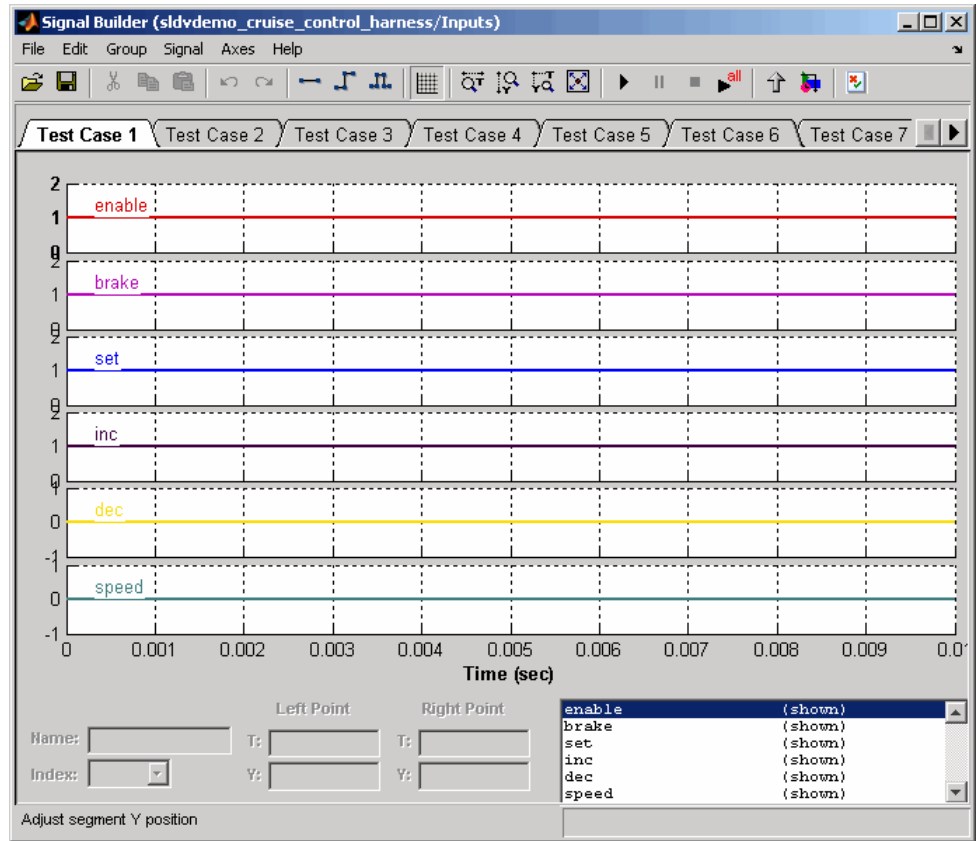


The screenshot shows a text editor window titled "Editor - C:\TEMP\docblock-133-001586914.txt". The window contains a list of test case objectives. The first test case has 8 objectives, the second has 3, and the third has 6. The objectives are listed as follows:

```
1 Test Case 1 (8 Objectives)
2   1. Logic "Logical Operator2", MCDC expression for output with Condition 1, T @T=0
3   2. Logic "Logical Operator1": Condition 1, T @T=0
4   3. Logic "Logical Operator": Condition 1, T @T=0
5   4. Logic "Logical Operator", MCDC expression for output with Condition 2, F @T=0
6   5. Logic "Logical Operator": Condition 2, F @T=0
7   6. Switch "Switch1": logical trigger input, true (output is from 1st input port) @T=0
8   7. Logic "Logical Operator2": Condition 1, T @T=0
9   8. SubSystem "PI Controller": enable logical value, false @T=0
10
11 Test Case 2 (3 Objectives)
12   1. Logic "Logical Operator": Condition 1, F @T=0
13   2. Logic "Logical Operator", MCDC expression for output with Condition 1, F @T=0
14   3. Logic "Logical Operator1": Condition 1, F @T=0
15
16 Test Case 3 (6 Objectives)
17   1. Switch "Switch1": logical trigger input, false (output is from 3rd input port) @T=0
18   2. Logic "Logical Operator2": Condition 2, F @T=0
```

The status bar at the bottom of the window shows "plain text file", "Ln 15 Col 1", and "OVR".

- **Inputs** — This Signal Builder block contains signals that comprise the test cases or counterexamples that the Simulink Design Verifier software generated. Double-click the Inputs block to open the Signal Builder dialog box and view its signals.



Each signal group represents a unique test case or counterexample. In the Signal Builder dialog box, select a group's tab to view the signals associated with a particular test case or counterexample. See “Working with Signal Groups” in *Using Simulink®* for more information about interacting with the Signal Builder dialog box.

- **Size-Type** — This Subsystem block transmits signals from the Inputs block to the Test Unit block. It ensures that the signals are of the appropriate size and data type, which the Test Unit block expects.
- **Test Unit** — This Subsystem block contains a copy of the original model that the Simulink Design Verifier software analyzed.

Simulating the Test Harness

The test harness model enables you to simulate a copy of your original model using the test cases or counterexamples that the Simulink Design Verifier software generates. Using the test harness model, you can simulate

- A counterexample
- A single test case, for which the Simulink® Verification and Validation™ software collects and displays model coverage information
- All test cases, for which the Simulink Verification and Validation software collects and displays cumulative model coverage information

Note By default, the Simulink Design Verifier software enables coverage reporting for test harness models that contain test cases. Although it enables coverage reporting with particular options selected, you can customize the settings to meet your needs. For more information, see “Specifying Model Coverage Reporting Options” in the *Simulink Verification and Validation User’s Guide*.


To simulate a single test case or counterexample:

- 1** In the test harness model, double-click the Inputs block.

The Signal Builder dialog box appears.

- 2** In the Signal Builder dialog box, select the tab associated with a particular test case or counterexample.

The Signal Builder dialog displays the signals that comprise the selected test case or counterexample.


- 3** In the Signal Builder dialog box, click the **Start simulation** button 

The Simulink software simulates the test harness model using the signals associated with the selected test case or counterexample. When simulating a test case, the Simulink Verification and Validation software collects model coverage information and displays a coverage report.

To simulate all test cases and measure their combined model coverage:

- 1 In the test harness model, double-click the Inputs block.

The Signal Builder dialog box appears.

- 2 In the Signal Builder dialog box, click the **Run all** button .

The Simulink software simulates the test harness model using all test cases, while the Simulink Verification and Validation software collects model coverage information and displays a coverage report.

See “Simulating with Signal Groups” in *Using Simulink* for more information about simulating models containing Signal Builder blocks.

Understanding Simulink® Design Verifier™ Reports

In this section...

“About Simulink® Design Verifier™ Reports” on page 8-8

“Front Matter” on page 8-8

“Summary Chapter” on page 8-9

“Block Replacements Summary Chapter” on page 8-14

“Test/Proof Objectives Chapter” on page 8-14

“Test Cases / Counterexamples Chapter” on page 8-19

“Approximations Chapter” on page 8-22

About Simulink® Design Verifier™ Reports

When you enable the **Generate report of the results** parameter (see “Report Pane” on page 5-14), the Simulink® Design Verifier™ software generates an HTML report after it completes its analysis. If the software’s **Mode** parameter specifies **Test generation**, the report describes the model’s test objectives and any corresponding test cases that result from the analysis. Otherwise, the software’s **Mode** parameter specifies **Property proving**, and the report describes the model’s proof objectives and any counterexamples that result from the analysis.

Front Matter

The report begins with two sections: title and table of contents.

Simulink Design Verifier Report

sldvdemo_flipflop

scowan

03-Jul-2007 09:45:52

Table of Contents

- [1. Summary](#)
- [2. Test Objectives](#)
 - [Status](#)
 - [sldvdemo_flipflop](#)
 - [D Flip-Flop](#)
- [3. Test Cases](#)
 - [Test Case 1](#)
 - [Test Case 2](#)
 - [Test Case 3](#)
 - [Test Case 4](#)
- [4. Approximations](#)

List of Tables

- 2.1. [Objectives Satisfied](#)

The title section lists the following information:

- Model or subsystem name that the Simulink Design Verifier software analyzed
- User name associated with the current MATLAB® session
- Date and time that the Simulink Design Verifier software generated the report

The table of contents follows the title section. Clicking items in the table of contents allows you to navigate quickly to particular chapters and sections.

Summary Chapter

The Summary chapter provides an overview of the Simulink Design Verifier analysis. It contains the following sections:

- “Input Model” on page 8-10
- “Analysis Information” on page 8-10
- “Output Files” on page 8-12
- “Options” on page 8-12

Input Model

The Input Model section provides information about the current version of the model.

Input Model	
File:	C:\MATLAB\toolbox\sldv\sldvdemos\sldvdemo_cruise_control.mdl
Version:	1.46
Time Stamp:	Sat Mar 3 02:53:32 2007
Author:	

The Input Model section lists the following:

- Path and file name of the model that the Simulink Design Verifier software analyzed
- Model version
- Date and time that the model was last saved
- Name of the person who last saved the model

See “Managing Model Versions” in *Using Simulink*[®] for details about specifying this information for your models.

Analysis Information

The Analysis Information section summarizes the results of the Simulink Design Verifier analysis. It looks like the following when the Simulink Design Verifier software generates test cases for a model:

Analysis Information

Design Verifier Version:	1.1
Total Analysis Time:	6.85 secs
Status:	Completed normally
Approximations:	1
Objectives Satisfied:	34
Objectives Satisfied - No Test Case:	0
Objectives Proven Unsatisfiable:	0
Objectives Undecided:	0
Objectives Producing Errors:	0

The Analysis Information section lists the following information for all analyses:

- Version of the Simulink Design Verifier software
- Total time the Simulink Design Verifier software spent to complete its analysis
- Completion status of the Simulink Design Verifier analysis
- Total number of different approximation schemes the Simulink Design Verifier software used in its analysis (see “Approximations Chapter” on page 8-22)
- Total number of test or proof objectives for which the Simulink Design Verifier software was unable to decide an outcome
- Total number of test or proof objectives that produced errors

If the Simulink Design Verifier software’s **Mode** parameter specifies Test generation, the Analysis Information section also lists:

- Total number of test objectives that the software satisfied
- Total number of test objectives that the software satisfied without generating test cases
- Total number of test objectives that the software determined to be unsatisfiable

Otherwise, if the Simulink Design Verifier software’s **Mode** parameter specifies Property proving, the Analysis Information section lists:

- Total number of proof objectives that the software proved valid
- Total number of proof objectives that the software disproved, for which it generated counterexamples that falsify each objective
- Total number of proof objectives that the software disproved without generating counterexamples

See “Test/Proof Objectives Chapter” on page 8-14 for more information related to the status of test and proof objectives.

Output Files

The Output Files section provides information about the artifacts that the Simulink Design Verifier software produced after it analyzed a model.

Output Files

```
Harness model: C:\sldv_output\sldvdemo_cruise_control\sldvdemo_cruise_control_harness.mdl  
Data file:      C:\sldv_output\sldvdemo_cruise_control\sldvdemo_cruise_control_sldvdata.mat  
Report:        C:\sldv_output\sldvdemo_cruise_control\sldvdemo_cruise_control_report.html
```

The Output Files section lists the following:

- Path and file name of the test harness model (see “Exploring Test Harness Models” on page 8-2)
- Path and file name of the Simulink Design Verifier data file (see “Examining Simulink® Design Verifier™ Data Files” on page 8-23)
- Path and file name of the Simulink Design Verifier report

Options

The Options section provides information about the Simulink Design Verifier analysis settings.

Options.

Parameter	Setting
Mode	TestGeneration
MaxProcessTime	60
DisplayUnsatisfiableObjectives	on
OutputDir	sldv_output/\$ModelName\$
MakeOutputFilesUnique	off
BlockReplacement	off
Parameters	on
ParametersConfigFileName	sldv_params_template.m
ModelCoverageObjectives	MCDC
TestConditions	UseLocalSettings
TestObjectives	UseLocalSettings
MaxTestCaseSteps	500
TestSuiteOptimization	CombinedObjectives
SaveHarnessModel	on
HarnessModelFileName	\$ModelName\$_harness
SaveDataFile	on
DataFileName	\$ModelName\$_sldvdata
SaveReport	on
ReportFileName	\$ModelName\$_report
ReportIncludeGraphics	off
DisplayReport	on

The Options section lists the names of parameters that affected the Simulink Design Verifier analysis, as well as the values those parameters specified. See “sldvoptions Object Parameters” on page 9-8 for more information about the parameters that this section displays.

Block Replacements Summary Chapter

The Block Replacements Summary chapter provides an overview of the block replacements that the Simulink Design Verifier software executed. It appears only if the Simulink Design Verifier software replaced any blocks in a model. The chapter displays a table that looks like the following:

#:	Replacement Rule / Block Type	Rule Description	Replaced Blocks
1	blkrep_rule_lookup_normal.m /Lookup	Inserts test objectives for each interval of 1-D lookup table blocks.	./Lookup Table
2	blkrep_rule_mpswitch2_normal.m /MultiPortSwitch	Constrains the first input to a 2-input Multiport switch block to prevent simulation errors.	./Multiport Switch

Each row of the table corresponds to a particular block replacement rule that the Simulink Design Verifier software applied to the model. The table lists the following:

- Name of the M-file that represents the block replacement rule, and the value of the BlockType parameter the rule specifies
- Description of the rule, which the MaskDescription parameter of the replacement block specifies
- Name of the block(s) that the Simulink Design Verifier software replaced in the model

See Chapter 3, “Working with Block Replacements” for more information.

Test/Proof Objectives Chapter

The Test/Proof Objectives chapter provides an overview of a model’s objectives. It contains sections similar to the following:

- “Status” on page 8-15
- “Model Hierarchy” on page 8-18

Status

The Status section summarizes all test or proof objectives in a model, including an objective's type, the model item to which it corresponds, and its description. This section displays each objective in one of the following tables associated with the objective's status:

- **Objectives Undecided** — Lists the test or proof objectives for which the Simulink Design Verifier software was unable to determine an outcome in the allotted time. In this case, either the software exceeded its analysis time limit (which the **Maximum analysis time** parameter specifies) or you aborted the analysis before it completed processing these objectives.

Table 2.3. Objectives Undecided

#:	Type	Model Item	Description
3	Decision	Switch	Switch "Switch", trigger >= threshold, F

- **Objectives Producing Errors** — Lists the test or proof objectives for which the Simulink Design Verifier software encountered errors during its analysis. In this case, analyzing these objectives involves nonlinear arithmetic, which the software does not support.

Table 2.4. Objectives Producing Errors

#:	Type	Model Item	Description
5	Decision	Saturation	Saturate "Saturation", input > lower limit, F
8	Decision	Saturation	Saturate "Saturation", input >= upper limit, T

If the Simulink Design Verifier software's **Mode** parameter specifies Test generation, the Status section also includes the following tables:

- **Objectives Proven Unsatisfiable** — Lists the test objectives that the Simulink Design Verifier software determined to be unsatisfiable. In this case, the software determined that there are no test cases that achieve these objectives.

Table 2.1. Objectives Proven Unsatisfiable

#:	Type	Model Item	Description
9	Custom Test Objective	Test Objective	Test Objective "Test Objective":1

- **Objectives Satisfied** — Lists test objectives that the Simulink Design Verifier software satisfied. In this case, the software generated test cases that achieve these objectives.

Table 2.2. Objectives Satisfied

#:	Type	Model Item	Description
1	Decision	Abs	Abs "Abs", input < 0, F
2	Decision	Abs	Abs "Abs", input < 0, T
4	Decision	Switch	Switch "Switch", trigger >= threshold, T
6	Decision	Saturation	Saturate "Saturation", input > lower limit, T
7	Decision	Saturation	Saturate "Saturation", input >= upper limit, F

- **Objectives Satisfied - No Test Case** — Lists test objectives that the Simulink Design Verifier software satisfied without generating test cases. In this case, you might have specified a test objective on a signal whose value the software cannot control; or the software might have encountered a divide-by-zero error when instantiating a test case.

Table 2.1. Objectives Satisfied - No Test Case

#:	Type	Model Item	Description
1	Custom Test Objective	Test Objective	Test Objective "Test Objective": 0.9
2	Custom Test Objective	Test Objective	Test Objective "Test Objective": 1.1

Otherwise, if the Simulink Design Verifier software's **Mode** parameter specifies Property proving, the Status section includes:

- **Objectives Proven Valid** — Lists the proof objectives that the Simulink Design Verifier software proved valid.

Table 2.1. Objectives Proven Valid

#:	Type	Model Item	Description
1	Custom Proof Objective	Proof Objective1	Proof Objective "Proof Objective" : 1

- **Objectives Falsified with Counterexamples** — Lists the proof objectives that the Simulink Design Verifier software disproved. In this case, the software generated counterexamples that falsify these objectives.

Table 2.1. Objectives Falsified with Counterexamples

#:	Type	Model Item	Description
2	Custom Proof Objective	Proof Objective	Proof Objective "Proof Objective" : 1

- **Objectives Falsified - No Counterexample** — Lists the proof objectives that the Simulink Design Verifier software disproved without generating counterexamples. In this case, you might have specified a proof objective on a signal whose value the software cannot control; or the software might have encountered a divide-by-zero error when instantiating a counterexample.

Table 2.1. Objectives Falsified - No Counterexample

#:	Type	Model Item	Description
1	Custom Proof Objective	Proof Objective	Proof Objective "Proof Objective" : 0.9
2	Custom Proof Objective	Proof Objective	Proof Objective "Proof Objective" : 1.1

Note The Status section displays only the tables that contain one or more objectives.

Model Hierarchy

Following the Status section is a series of sections that represent the model hierarchy—from the root level to the model's subsystems and Stateflow® charts. Each section summarizes all the test or proof objectives that a particular hierarchical level of the model contains.

For example, suppose a model named `my_model` contains Abs and Switch blocks in its root level. If you use the Simulink Design Verifier software to generate tests for this model, the report displays a section that lists only the test objectives associated with the root-level model:

my_model

Objectives of: Abs

#:	Status	Test Cases	Description
1	Satisfied	TC 1	input < 0, F
2	Satisfied	TC 3	input < 0, T

Objectives of: Switch

#:	Status	Test Cases	Description
3	Undecidable	n/a	trigger >= threshold, F
4	Satisfied	TC 1	trigger >= threshold, T

Further, suppose that the root level of this same model includes a subsystem named `my_subsystem`, which contains a Test Objective block. In another section, the report lists the test objective associated with this subsystem:

my_subsystem

Objectives of: Test Objective

#:	Status	Test Cases	Description
5	Satisfied	TC 2	1

Each section lists objectives that correspond to particular model items in a model hierarchy. This includes the following information:

- Status of a test or proof objective
- Test case that achieves a test objective, or counterexample that falsifies a proof objective
- Description of a test or proof objective

Test Cases / Counterexamples Chapter

The Test Cases / Counterexamples chapter provides an overview of the test cases or counterexamples that the Simulink Design Verifier software generated during its analysis. Depending on whether the software's **Mode** parameter specifies Test generation or Property proving, this chapter includes sections associated with the following:

- “Test Cases” on page 8-19
- “Counterexamples” on page 8-21

Test Cases

If the Simulink Design Verifier software's **Mode** parameter specifies Test generation, the Test Cases chapter includes a series of sections that summarize the test cases the software generated.

Test Case 1

Summary

Length: 0.2 Seconds (3 sample periods)

Objective Count: 4

Objectives Reached At:

Step	Time	Objectives
2	0.1	1 4 5 10

Generated Input Data.

Time	0	0.1
Step	1	2
signal_A	0	1
signal_B	1	0
signal_C	0	1

Each section lists the following information about a test case:

- Length of the signals that comprise the test case
- Total number of test objectives that the test case achieves
- Time step and corresponding time at which the test case achieves particular test objectives
- Values of the signals that comprise the test case

Note The Generated Input Data table can display a dash (-) instead of a number as a signal value. In this case, the value of the signal at that time step does not affect the test objective. In the test harness model, the Inputs block represents these values with zeros unless you enable the **Randomize data that does not affect outcome** parameter (see “Randomize data that does not affect outcome” on page 5-14).

Counterexamples

If the Simulink Design Verifier software's **Mode** parameter specifies Property proving, the Counterexamples chapter includes a series of sections that summarize the counterexamples the software generated.

Counterexample 1

Summary

Length: 0.2 Seconds (2 sample periods)

Objective Count: 1

Objectives Reached At:

Objectives Falsified

2

Generated Input Data.

Time 0	
Step 1	
In4	0
In1	-

Each section lists the following information about a counterexample:

- Length of the signals that comprise the counterexample
- Total number of proof objectives that the counterexample falsifies
- Particular proof objectives that the counterexample falsifies
- Values of the signals that comprise the counterexample

Note The Generated Input Data table can display a dash (-) instead of a number as a signal value. In this case, the value of the signal at that time step does not affect the proof objective. In the test harness model, the Inputs block represents these values with zeros unless you enable the **Randomize data that does not affect outcome** parameter (see “Randomize data that does not affect outcome” on page 5-14).

Approximations Chapter

The Approximations chapter provides an overview of the approximations that the Simulink Design Verifier software uses. It displays a table that appears like this:

#:	Type	Description
1	Rational approximation	Model includes floating point arithmetic that Simulink Design Verifier approximates with rational number arithmetic.
2	Lookup table (2-D) linearization	Model includes 2-D lookup tables. Simulink Design Verifier approximates nonlinear 2-D interpolation with linear interpolation by fitting planes to each interpolation interval.

Each row of the table describes a specific type of approximation that the Simulink Design Verifier software used during its analysis of the model.

Note Review the analysis results carefully when the Simulink Design Verifier software uses approximations. In rare cases, an approximation can result in test cases that fail to achieve test objectives or counterexamples that fail to falsify proof objectives. For example, suppose the software generates a test case signal that should achieve an objective by exceeding a threshold; however, a floating-point-roundoff error might prevent that signal from attaining the threshold value.

Examining Simulink® Design Verifier™ Data Files

In this section...

“About Simulink® Design Verifier™ Data Files” on page 8-23

“Anatomy of the sldvData Structure” on page 8-23

“Simulating Models with Simulink® Design Verifier™ Data Files” on page 8-28

About Simulink® Design Verifier™ Data Files

When you enable the **Save test data to file** parameter (see “Results Pane” on page 5-12), the Simulink® Design Verifier™ software generates a data file after it completes its analysis. The data file is a MAT-file that contains a structure named `sldvData`. This structure stores all the data that the software gathers and produces during its analysis of a model. Although the software displays the same data graphically in the test harness model and report, you might like to use the data file to conduct your own analysis or generate a custom report.

Anatomy of the sldvData Structure

When the Simulink Design Verifier software completes its analysis, it produces a MAT-file that contains a structure named `sldvData`. To explore the contents of the `sldvData` structure:

- 1 Generate test cases for the `sldvdemo_flipflop` model (see “Running a Demo Model” on page 1-6).

The Simulink Design Verifier software produces a data file named `sldvdemo_flipflop_sldvdata.mat` in the `sldv_output\sldvdemo_flipflop` directory.

- 2 At the MATLAB® prompt, enter the following command:

```
load('sldv_output\sldvdemo_flipflop\sldvdemo_flipflop_sldvdata.mat')
```

The MATLAB software loads the `sldvData` structure into its workspace. This structure contains the Simulink Design Verifier analysis results of the `sldvdemo_flipflop` model.

3 At the MATLAB prompt, enter `sldvData`.

The MATLAB Command Window displays the following field names that constitute the structure:

```
sldvData =

    AnalysisInformation: [1x1 struct]
      ModelObjects: [1x2 struct]
      Objectives: [1x12 struct]
      TestCases: [1x4 struct]
```

See “Structures” in the MATLAB documentation for more information about working with structures.

The following sections describe the contents of each primary field in the `sldvData` structure:

- “AnalysisInformation Field” on page 8-24
- “ModelObjects Field” on page 8-25
- “Objectives Field” on page 8-26
- “TestCases Field” on page 8-26

AnalysisInformation Field

In the `sldvData` structure, the `AnalysisInformation` field lists settings of particular analysis options and related information. The following table describes each subfield of the `AnalysisInformation` field.

Subfield Name	Description
Mode	String specifying the value of the <code>Mode</code> parameter associated with the model’s <code>sldvoptions</code> object (see “ <code>sldvoptions</code> Object Parameters” on page 9-8).
SampleTimes	For internal use only.
InputPortInfo	Cell array of structures specifying information about each <code>Inport</code> block in the top-level system.

Subfield Name	Description
ProvingStrategy	String specifying the value of the ProvingStrategy parameter associated with the model's sldvoptions object (see “sldvoptions Object Parameters” on page 9-8).
Status	String specifying the completion status of the Simulink Design Verifier analysis.
MaxViolation-Steps	String specifying the value of the MaxViolationSteps parameter associated with the model's sldvoptions object (see “sldvoptions Object Parameters” on page 9-8).
MaxProcessTime	String specifying the value of the MaxProcessTime parameter associated with the model's sldvoptions object (see “sldvoptions Object Parameters” on page 9-8).

ModelObjects Field

In the sldvData structure, the ModelObjects field lists the model items and their associated objectives. The following table describes each subfield of the ModelObjects field.

Subfield Name	Description
descr	String specifying the full path to a model object, including objects in a Stateflow® chart.
s1Path	String specifying the full path to a Simulink® model object.
sfObjType	String specifying the type of a Stateflow object, e.g., S for state and T for transition.
sfObjNum	Integer representing the unique identifier of a Stateflow object.
objectives	Vector of integers representing the indices of objectives associated with a model object.
handle	Real number specifying the handle of a model object.

Objectives Field

In the `sldvData` structure, the `Objectives` field lists information about each objective, such as its type, status, and description. The following table describes each subfield of the `Objectives` field.

Subfield Name	Description
<code>type</code>	String specifying the type of an objective.
<code>status</code>	String specifying the status of an objective.
<code>descr</code>	String specifying the description of an objective.
<code>label</code>	String specifying the label of an objective.
<code>outcomeValue</code>	Integer specifying an objective's outcome.
<code>coveragePointIdx</code>	Integer representing the index of a coverage point with which an objective is associated.
<code>modelObjectIdx</code>	Integer representing the index of a model object with which an objective is associated.
<code>testCaseIdx</code>	Integer representing the index of a test case or counterexample that addresses an objective.

TestCases Field

In the `sldvData` structure, the `TestCases` field lists information about each test case or counterexample, such as its signal values and either the test objectives that it achieves or the proof objectives that it falsifies. The following table describes each subfield of the `TestCases` field.

Subfield Name	Description
<code>timeValues</code>	Vector specifying the time values associated with signals in a test case or counterexample.
<code>dataValues</code>	Cell array specifying the data values associated with signals in a test case or counterexample.

Subfield Name	Description
paramValues	<p>Structure specifying the parameter values associated with a test case or counterexample. Its fields include:</p> <p>name — String specifying the name of a parameter.</p> <p>value — Number specifying the value of a parameter.</p> <p>noEffect — Logical value specifying whether a parameter's value affects an objective.</p>
stepValues	<p>Vector specifying the number of time steps that comprise signals in a test case or counterexample.</p>
objectives	<p>Structure specifying objectives that a test case or a counterexample addresses. Its fields include:</p> <p>objectiveIdx — Integer representing the index of an objective that a test case achieves or a counterexample falsifies.</p> <p>atTime — Time value at which either a test case achieves an objective or a counterexample falsifies an objective.</p> <p>atStep — Time step at which either a test case achieves an objective or a counterexample falsifies an objective.</p>
dataNoEffect	<p>Cell array of logical vectors specifying whether a signal's data values affect an objective. The vector uses 1 to indicate that a signal's data value does not affect an objective; otherwise, it uses 0.</p>
signalLabels	<p>Cell array of strings specifying the labels of signals in a test case or counterexample.</p>

Subfield Name	Description
portdimensions	Cell array of vectors specifying the dimensions of signals in a test case or counterexample.
expectedOutput	Cell array of vectors specifying the output values that result from simulating the model using the test case signals. Each cell represents the output values associated with a different Outport block in the top-level system. See “Include expected output values” on page 5-14 for more information.

Simulating Models with Simulink® Design Verifier™ Data Files

The `sldvrntest` function enables you to simulate a model using test cases or counterexamples that reside in a Simulink Design Verifier data file. For example, suppose the following command specifies the location of the data file that the Simulink Design Verifier software produced after analyzing the `sldvdemo_flipflop` model (see “Running a Demo Model” on page 1-6):

```
sldvDataFile = 'sldv_output\sldvdemo_flipflop\sldvdemo_flipflop_sldvdata.mat'
```

Use the `sldvrntest` function to simulate the `sldvdemo_flipflop` model using test case 2 in the data file:

```
output = sldvrntest('sldvdemo_flipflop', sldvDataFile, 2)
```

See `sldvrntest` in Chapter 9, “Function Reference” for more information.

Analyzing Large Models and Improving Performance

This chapter describes some practical strategies for analyzing larger models with the Simulink® Design Verifier™ software and troubleshooting errors and warnings. These strategies will help you to get the most benefit from the Simulink Design Verifier software.

In general, you will see improved performance by breaking your model into smaller components and running Simulink Design Verifier analyses on them. The strategies in this chapter compliment a divide-and-conquer approach so you can run a Simulink Design Verifier analysis on larger portions of your system.

- “How the Simulink® Design Verifier™ Software Works” on page A-2
- “Sources of Model Complexity” on page A-5
- “Handling Models with Large Numbers of Inputs” on page A-6
- “Reducing Complexity from Floating-Point Operations and Nonlinear Arithmetic” on page A-7
- “Partitioning Inputs and Generating Tests Incrementally” on page A-9
- “Handling Models with Large State Spaces” on page A-11
- “Handling Problems with Counters and Timers” on page A-12
- “Strategies for Proving Properties of Large Models” on page A-13

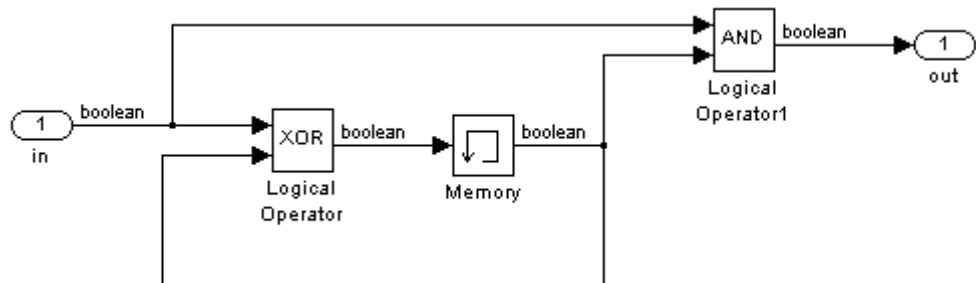
How the Simulink® Design Verifier™ Software Works

The Simulink® Design Verifier™ software is a very efficient search tool that explores the simulation behavior of a model. It searches the possible values of model inputs to find a simulation that satisfies an objective. The exact definition of these search objectives comes from the Simulink Design Verifier configuration options and your model's structure.

The search always begins with the initial configuration of the model (at $t=0$) and can span an arbitrary number of time steps. Generally, there are an infinite number of search paths because the values of inputs are independent from one time step to the next, and there is no fixed limit to the number of time steps. If there were no way to reduce the search space, the Simulink Design Verifier software would never be able to stop its analysis.

The search is fundamentally limited by tracking the persistent information in the model such as discrete states, data-store memories, and persistent variables. Once a search has explored all possible inputs from all possible configurations, the results are equivalent to having performed a complete search of every possible infinite sequence of inputs.

Consider a simple Simulink® model with two Logical Operator blocks and a Memory block:



The persistent information in this model is limited to the Boolean value of the Memory block. The input to the model is a single Boolean value. Therefore the complete behavior of the model, including the behavior that would result from an arbitrarily long sequence of inputs, is described by the following table:

#	Input	Memory Value	Output	Next Memory Value
1	false	false	false	false
2	true	false	false	true
3	false	true	false	true
4	true	true	true	false

If you run the Simulink Design Verifier software to find a test case with a true output, it looks through this table to see if such a scenario is possible.

Once the Simulink Design Verifier software discovers a configuration that satisfies an objective, it needs to find a path to reach this configuration from the initial conditions. If the initial memory value is true, the test case would only need to be a single time step where the input was true. If the initial value of the memory is false, the test case would need to force the memory to be true and the test case would be a sequence of row 2 followed by row 4.

There are an infinite number of test cases that will cause the output to be true, and regardless of the state value, the output can be held false for an arbitrary time before making it true. When the Simulink Design Verifier software searches, it returns the first case it encounters that satisfies the objective. This will invariably be the simulation with the fewest time steps. Sometimes this result is undesirable because it is unrealistic or does not satisfy some other test requirement.

The same basic principles from this example apply to property proving and test generation. During test generation the search criteria is explicitly specified from the options. During property proving the objective is the opposite of the proof to be satisfied. If that objective is satisfied the path is returned as a counterexample of the proof. If the search is completed without finding a satisfying path the proof completes successfully.

In larger more complicated models, the Simulink Design Verifier software uses mathematical techniques to simplify the search problem. It can identify portions of the model that do not affect the objectives of interest. It can discover relationships within the model that reduce the complexity of the search, and it can reuse the intermediate results from one objective to another.

Ultimately the problem is reduced to a search through the logical values that describe your model.

The Simulink Design Verifier software is particularly efficient at simplifying linear arithmetic of floating-point numbers by approximating them with rational numbers. The Simulink Design Verifier software discovers how the logical relationships between these variables affect the proof and test objectives. This enables the Simulink Design Verifier software to support supervisory logic that is commonly found in embedded controls designs.

Sources of Model Complexity

A model can complicate the search process in the following ways:

- Size of inputs
- Number of possible configurations
- Ability to reach one configuration from another

You need to understand these sources of complexity and the strategies to reduce their impact to get the best performance from the Simulink® Design Verifier™ software.

Handling Models with Large Numbers of Inputs

Input complexity comes from the number of inputs, the type of the inputs, and the way the inputs affect the model state and the objectives of the analysis. Because the search is performed on a logical simplification of your model, the Simulink® Design Verifier™ software is more efficient at handling logical inputs than integer or floating-point inputs.

Floating-point inputs can be efficiently handled when their values impact the design through linear inequalities such as $x < y$ or $a > 0$. Nonlinear arithmetic of floating-point numbers is not supported by the Simulink Design Verifier software, as occurs with multiplication or division unless one of the multiply operands or the divisor is a constant.

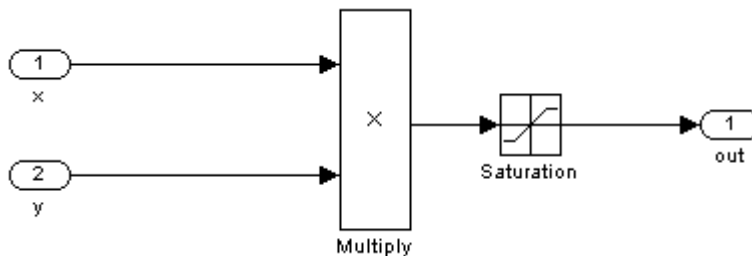
Input complexity can also result from certain cast operations. For example, casting a double to an int8 can introduce a nonlinearity in certain situations.

You can reduce input complexity by separating the logical and arithmetic portions of a design and restricting a Simulink Design Verifier analysis to the logical portion.

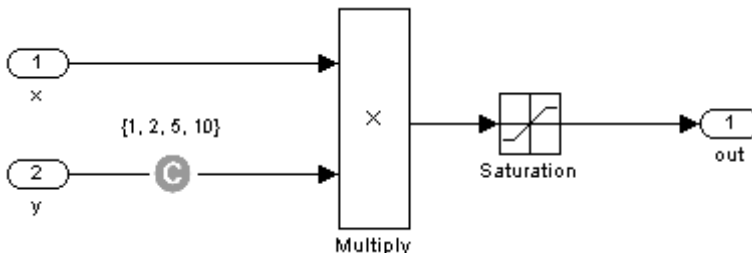
Reducing Complexity from Floating-Point Operations and Nonlinear Arithmetic

Another very effective strategy is to restrict the floating-point inputs to a set of representative values or, ideally, a single constant value. This process, called discretization, treats the free floating-point input as though it were an enumeration. Discretization is the simplest way to handle nonlinear arithmetic from multiplication and division.

Consider the following model with a Product block feeding a Saturation block:

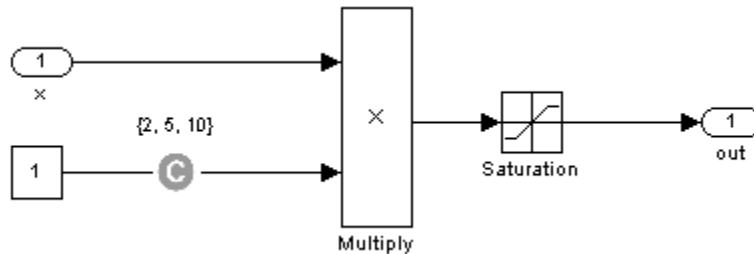


The Simulink® Design Verifier™ software generates errors when attempting to satisfy the upper and lower limits of the Saturation block. You can work around these errors by restricting one of the inputs to be a range of values. For example, if you restrict the second input (y) to be either 1, 2, 5, or 10, the Simulink Design Verifier software will produce test cases for all inputs:



You can also constrain signals that are intermediate or output values of the model. Sometimes this makes it easier to work around multiplication or divisions that are contained inside lower-level subsystems and do not depend

on input values. In these situations you must be careful to avoid creating constraints that contradict the model. This occurs when a constraint can never be satisfied because it contradicts some aspect of the model or some other constraint. Here is a simple example of a contradictory model:



When you work with very large models that have many multiplication and division operations, it is often easier to add constraints to all of the floating-point inputs rather than to identify the precise set of inputs that require constraints.

As you create large models you should identify sets of values for each Input port that are required to satisfy your testing needs. For example, if you have an input for model speed, and within your design there are paths of execution that are conditioned on speed being above or below thresholds of 80, 150, 600 and 8000 RPM, you might choose to restrict speed values to be either 50, 100, 200, 1000, 5000 or 10000 RPM so that every threshold can be either active or inactive.

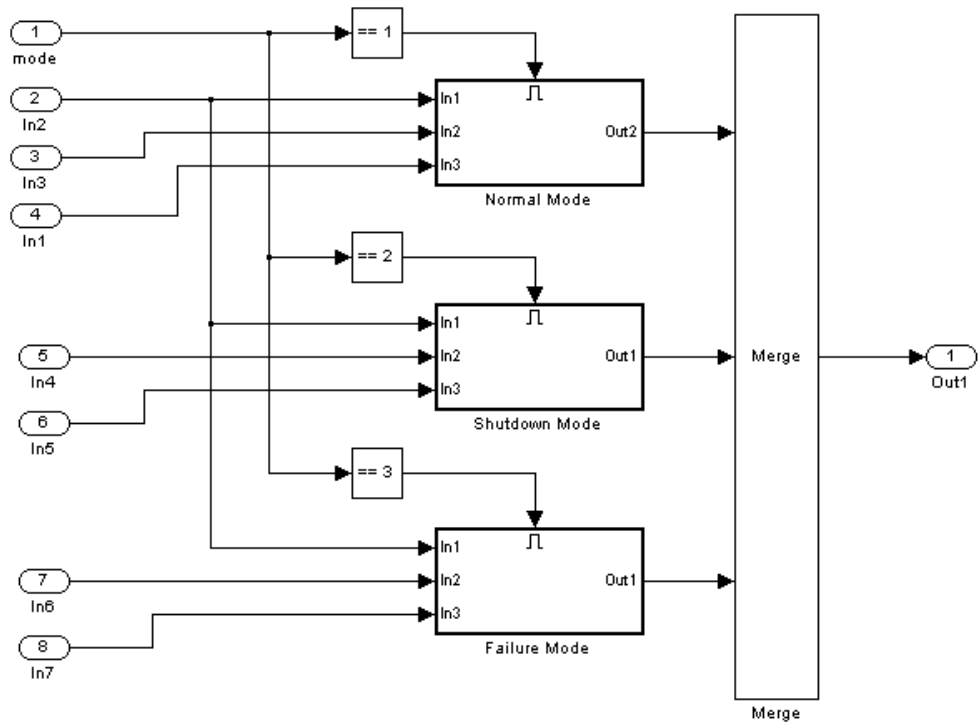
Partitioning Inputs and Generating Tests Incrementally

Like other Simulink® parameters, constraint values can be shared across several blocks by referencing a common workspace variable and they can be initialized from M-files. If you have several inputs related to speed, such as desired speed, measured speed, and average speed, you might choose to constrain all of them to the same set of values.

You can use parameterized constraints and successive runs of the Simulink® Design Verifier™ software to implement an incremental test generation strategy:

- 1** Partition inputs so that some are held constant, some are restricted to sets of constants, and some are free.
- 2** Generate test cases and run those test cases to collect model coverage.
- 3** Choose new values and new partitions of inputs.
- 4** Generate tests for missing coverage using `sldvgencov` and the current test coverage.
- 5** Repeat steps 3 and 4 until sufficient coverage is generated.

You should choose partitions of inputs that enable further simplification when a Simulink Design Verifier analysis runs. Consider the following model, which has three mutually independent enabled subsystems:



You can incrementally generate test cases for each of these subsystems by constraining the first input to the appropriate constant value before running a Simulink Design Verifier analysis. In this way, as tests are created for each of the subsystems, the complexity of the other two is ignored.

Handling Models with Large State Spaces

Persistent design variables impact the complexity of analysis in much the same way as input complexity. As such, many of the same strategies can be used to simplify the complexity of the state space that must be searched. States that are delayed values of inputs can be simplified by applying constraints to the input signal that is delayed. States that are contained within conditionally executed subsystems can be simplified by constraining the input so the system does not execute.

States that are computed from previous state values present a special challenge. For example, the integrator value in a PID controller can be restricted only to a set of values if that set includes all reachable values from the initial value or the input is forced to be 0. Both of these limitations are usually not practical and would probably make test generation less complete.

An alternative strategy is to leverage any existing simulation data to help satisfy your testing needs. If you have existing test data, you can run this on your model and collect model coverage. Using the `sldvgencov` function, you can ignore model coverage objectives that have already been satisfied in simulation when you supply a coverage data object.

Handling Problems with Counters and Timers

Complexity from states occurs from both the size of the state representation and the number of time steps that are required to transition from one state to another. The Simulink® Design Verifier™ software searches through sequences of time steps, starting from the default configuration, to find input values that reach a state that satisfies an objective. The search process proceeds in a breadth-first manner. All configurations that can be reached in a single time step are investigated before any of the configurations that can be reached in two time steps. Likewise, all configurations that can be reached in two time steps are investigated before any configuration that requires three or more time steps, etc.

Models that contain time delays, such as countdown timers, hinder a Simulink Design Verifier analysis by forcing the search to span large numbers of time steps. By design, the value of a counter can reach n only when its previous value is $n-1$.

Similar effects can also occur when systems use extensive averaging and filtering to delay the response to a change in inputs. Any aspect of the design that delays the response will cause the test sequences to contain more time steps and longer test cases that are more difficult to identify.

There are some basic strategies you can use to improve performance in models that have delays:

- 1** Make time delays calibratable parameters and choose very small values when running a Simulink Design Verifier analysis. It is likely that a system with a logical error when a time delay is set to 2000 steps will still demonstrate that error if the time delay is changed to 2 steps. If your system has several delays, you might want to choose small but unique values for each of them so that your delays will be progressively satisfied.
- 2** Choose higher frequency cutoffs for filters and fewer samples to average so that filtering delays are minimized.

Strategies for Proving Properties of Large Models

Property proving uses the same underlying techniques as test generation and suffers from the same performance limitations; but unlike test generation, it is often impossible to simplify the problem without compromising the validity of the results. Simple proof objectives that are not affected by model dynamics can often be proven quickly. Otherwise, a successful proof requires that the Simulink® Design Verifier™ software searches through all reachable configurations of your model—even the ones that are reached only after long time delays. The computation time and memory required to search a model completely often make an exhaustive proof impractical.

Alternatively, you can use the bounded model checking capability in the Simulink Design Verifier software to examine properties in larger, more complicated models. Using bounded model checking, you restrict the search for property violations to a predefined limit of time steps. If a violation is not detected, you can be confident that it is impossible to violate the property with any input sequence having fewer time steps than the specified limit; however, you will not prove that the property is true because there might be a counterexample having more time steps than the specified limit.

To configure the Simulink Design Verifier software for bounded model checking, on the **Design Verifier > Property Proving** pane of the Configuration Parameters dialog box, specify the value of the **Strategy** parameter as `Find violation`. When you use this strategy, the **Maximum violation steps** parameter becomes active so that you can specify an upper bound for the number of time steps in the search. See “Property Proving Pane” on page 5-11 for more information.

An effective strategy for proving properties combines proving and searching for violations to get the most benefit from the Simulink Design Verifier software that is practical for the properties and models under investigation:

- 1 Start with the `Proving` strategy and use a relatively short processing time limit, such as 5-10 minutes. If there are trivial counterexamples or if your properties do not depend on model dynamics, the Simulink Design Verifier analysis should complete in that amount of time.
- 2 Switch to the `Find violation` strategy and choose a small bound on the number of violation steps, such as 4-6. If your properties have simple

counterexamples, the Simulink Design Verifier software should discover them.

- 3** If you do not find any violations with a small bound, increase the bound and look for longer counterexamples. You probably will want to increase the bound in several increments and observe the processing time and memory consumption. System resources might limit the length of violation that can be searched. You should also consider the dynamics of your model and the number of time steps that are needed to transition between an arbitrary pair of configurations. If you choose too large of a bound, the violation search can be more complex than the unbounded proof.
- 4** If you are able to run violation searches with relatively large bounds, e.g., 30-50 time steps, you can switch back to the Proving strategy and use a longer time limit, such as several hours.

Function Reference

sldvblockreplacement

Purpose Replace model blocks to support Simulink® Design Verifier™ analysis

Syntax `[status, newmodel] = sldvblockreplacement(model)`
`[status, newmodel] = sldvblockreplacement(model, options)`

Description `[status, newmodel] = sldvblockreplacement(model)` copies `model` and replaces specified model blocks and other model components to prepare the model for a Simulink Design Verifier analysis. This function replaces the blocks of the model according to the block replacement rules specified in the configuration settings associated with `model`. This function returns a handle to the new model in `newmodel`. The `sldvblockreplacement` function returns 1 upon successful completion and 0 otherwise.

`[status, newmodel] = sldvblockreplacement(model, options)` copies `model` and replaces specified model blocks and other model components to prepare the model for a Simulink Design Verifier analysis. This function replaces the blocks of the model according to the block replacement rules using the `sldvoptions` object specified by `options`. This function returns a handle to the new model in `newmodel`.

See Also `sldvoptions`

Purpose Check model for compatibility with Simulink® Design Verifier™ analysis

Syntax

```
status = sldvcompat(model)
status = sldvcompat(block)
status = sldvcompat(model, options)
```

Description `status = sldvcompat(model)` returns 1 if `model` is compatible with the Simulink Design Verifier software and 0 otherwise. When checking for compatibility, the Simulink Design Verifier software replaces model blocks if this option has been enabled.

Note If you call this function without specifying a model, the function operates on the current system.

`status = sldvcompat(block)` converts the Simulink® block into a temporary model, then checks the compatibility of that model with the Simulink Design Verifier software. The function destroys the temporary model after the compatibility check.

`status = sldvcompat(model, options)` checks the subsystem specified by `model` for compatibility with the Simulink Design Verifier software using the `sldvoptions` object specified by `options`.

Examples The following commands open the `vdp` demo model and check for its compatibility with the Simulink Design Verifier software:

```
vdp
status = sldvcompat('vdp')
```

The Simulink Design Verifier software displays the result as follows:

```
Checking compatibility of model "vdp"

Model "vdp" is not compatible with Simulink Design Verifier

status =

    0
```

The following commands open `sldvdemo_flipflop` and check for its compatibility with the Simulink Design Verifier software:

```
sldvdemo_flipflop
status = sldvcompat('sldvdemo_flipflop')
```

The Simulink Design Verifier software displays the results as follows:

```
Checking compatibility of model "sldvdemo_flipflop"

Compiling model...done
Checking compatibility...done

Model "sldvdemo_flipflop" is compatible with
    Simulink Design Verifier.

ans =

    1
```

See Also

`sldvoptions`, `sldvrun`

Purpose Extract subsystem contents into new model for Simulink® Design Verifier™ analysis

Syntax `[status, modelH] = sldvextract(blockH)`

Description `[status, modelH] = sldvextract(blockH)` extracts the contents of the subsystem that `blockH` specifies, with which it creates a new model that you can analyze using the Simulink Design Verifier software. The `sldvextract` function returns the handle of the new model in `modelH`. It returns `status` as 1 upon successful completion and 0 otherwise.

sldvgencov

Purpose Run Simulink® Design Verifier™ analysis to obtain missing model coverage

Syntax `[status, cvdo] = sldvgencov(model, options, startcov)`

Description `[status, cvdo] = sldvgencov(model, options, startcov)` runs a Simulink Design Verifier analysis on the specified model using the `sldvoptions` object specified by `options`. The analysis ignores all model coverage objects that are satisfied in the `cvdata` object specified by `startcov`. The `sldvgencov` function returns 1 for `status` if the Simulink Design Verifier software was successful or 0 otherwise. It also measures the coverage in the new tests and returns the resulting `cvdata` object `cvdo`.

See Also `sldvoptions`, `sldvrun`

Purpose Merge test cases and initializations into one model

Syntax `status = sldvharnessmerge(name, models,
initialization_commands)`

Description `status = sldvharnessmerge(name, models, initialization_commands)` collects the test data and initialization commands from each test harness model listed in `models` and saves them in `name`. This function assumes that you have created each test harness model with the Simulink® Design Verifier™ software, either with the `sldvrun` function or the **Design Verifier > Generate Tests** menu item.

If `name` does not exist, this function creates it as a copy of the first model in `models`. This function then copies the data from the other models into this model. If `name` was created from a previous `sldvharnessmerge` run, subsequent runs of this function for `name` will maintain the correct structure and initialization from that earlier run. If `name` matches an existing Simulink® model, this function merges the test data from `models` into `name`.

- `models` can be a cell array of model names or an array of model handles.
- `initialization_commands` must be a cell array of strings the same length as `models`. `initialization_commands` define parameter settings for the test cases of each test harness model. Each time a model test case executes, the associated initialization command is evaluated in the base workspace.

Consider using `sldvharnessmerge` with `sldvgencov` to combine test cases that use different sets of parameter values.

See Also `sldvgencov`

sldvoptions

Purpose Access Simulink® Design Verifier™ options object

Syntax
options = sldvoptions
options = sldvoptions(model)

Description options = sldvoptions returns a Simulink Design Verifier options object that contains default values for its parameters (see “sldvoptions Object Parameters” on page 9-8).
options = sldvoptions(model) returns the Simulink Design Verifier options object attached to model.

sldvoptions Object Parameters The following table lists and describes parameters that comprise a Simulink Design Verifier options object.

Parameter	Description	Values
Assertions	Set by the Assertion blocks option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
BlockReplacement	Set by the Apply block replacements option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	'on' {'off'}

Parameter	Description	Values
BlockReplacementModel-FileName	Set by the File path of the output model option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	string {'\$ModelName\$_replacement'}
BlockReplacementRules-List	Set by the List of block replacement rules option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	string {'<FactoryDefaultRules>'}
DataFileName	Set by the Data file name option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	string {'\$ModelName\$_sldvdata'}
DisplayReport	Set by the Display report option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	{'on'} 'off'
DisplayUnsatisfiable-Objectives	Set by the Display unsatisfiable test objectives option on the Design Verifier pane of the Configuration Parameters dialog box.	{'on'} 'off'

sldvoptions

Parameter	Description	Values
HarnessModelFileName	Set by the Harness model file name option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	string { '\$ModelName\$_harness' }
MakeOutputFilesUnique	Set by the Make output file names unique by adding a suffix check box on the Design Verifier pane of the Configuration Parameters dialog box.	{ 'on' } 'off'
MaxProcessTime	Set by the Maximum analysis time option on the Design Verifier pane of the Configuration Parameters dialog box.	double { '600' }
MaxTestCaseSteps	Set by the Maximum test case steps option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	int32 { '500' }
MaxViolationSteps	Set by the Maximum violation steps option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	int32 { '20' }

Parameter	Description	Values
Mode	Set by the Mode option on the Design Verifier pane of the Configuration Parameters dialog box.	{'TestGeneration'} 'PropertyProving'
ModelCoverageObjectives	Set by the Model coverage objectives option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'None' 'Decision' 'ConditionDecision' {'MCDC'}
OutputDir	Set by the Output directory option on the Design Verifier pane of the Configuration Parameters dialog box.	string {'sldv_output/\$ModelName\$'}
Parameters	Set by the Apply parameters option on the Design Verifier > Parameters pane of the Configuration Parameters dialog box.	{'on'} 'off'
ParametersConfigFile-Name	Set by the Parameter configuration file option on the Design Verifier > Parameters pane of the Configuration Parameters dialog box.	string {'sldv_params_template.m'}

sldvoptions

Parameter	Description	Values
ProofAssumptions	Set by the Proof assumptions option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
ProvingStrategy	Set by the Strategy option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'FindViolation' {'Prove'} 'ProveWithViolationDetection'
RandomizeNoEffectData	Set by the Randomize data that does not affect outcome option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}
ReportFileName	Set by the Report file name option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	string {'\$ModelName\$_report'}
ReportIncludeGraphics	Set by the Include screen shots and plots option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	'on' {'off'}

Parameter	Description	Values
SaveDataFile	Set by the Save test data to file option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	{'on'} 'off'
SaveExpectedOutput	Set by the Include expected output values option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}
SaveHarnessModel	Set by the Save test harness as model option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	{'on'} 'off'
SaveReport	Set by the Generate report of the results option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	{'on'} 'off'
TestConditions	Set by the Test conditions option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}

sldvoptions

Parameter	Description	Values
TestObjectives	Set by the Test objectives option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
TestSuiteOptimization	Set by the Test suite optimization option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	{'CombinedObjectives'} 'IndividualObjectives' 'LargeModel'

See Also

sldvblockreplacement, sldvcompat, sldvgencov, sldvrun

Purpose Run Simulink® Design Verifier™ analysis on model or system

Syntax

```
status = sldvrun(model)
status = sldvrun(block)
status = sldvrun(model, options)
[status, filenames] = sldvrun(model, options)
```

Description `status = sldvrun(model)` runs a Simulink Design Verifier analysis on the specified model. The Simulink Design Verifier software uses the configuration settings associated with `model` (if available); otherwise, the software uses its default configuration settings. Upon completion, `sldvrun` returns one of the following values for `status`:

- -1 — Maximum processing time was exceeded.
- 0 — An error occurred.
- 1 — Preprocessing completed normally.

Note If you call this function without specifying a model, the function operates on the current system.

`status = sldvrun(block)` converts the Simulink® block into a new model, then runs a Simulink Design Verifier analysis on the new model. The Simulink Design Verifier software uses the configuration settings associated with the parent model of `block` (if available); otherwise, the software uses its default configuration settings.

`status = sldvrun(model, options)` runs a Simulink Design Verifier analysis on the model specified by `model`. The Simulink Design Verifier software uses the `sldvoptions` object specified by `options`.

`[status, filenames] = sldvrun(model, options)` runs a Simulink Design Verifier analysis on the model specified by `model`. This function returns `status` and `filenames`, a structure whose fields list the names of the files that the Simulink Design Verifier software generates:

sldvrun

- `HarnessModel` — Simulink harness model.
- `DataFile` — MAT-file that contains raw input data.
- `Report` — HTML report that documents the results.
- `ExtractedModel` — Simulink model extracted from subsystem.
- `BlockReplacementModel` — Simulink model obtained from block replacements.

See Also

`sldvcompat`, `sldvgencov`, `sldvoptions`

Purpose Simulate model using test case in Simulink® Design Verifier™ data file

Syntax

```
data = sldvrntest(model, sldvDataFile, testIdx)
data = sldvrntest(model, sldvDataFile)
[data, cvdo] = sldvrntest(model, sldvDataFile, testIdx,
    true)
[data, cvdo] = sldvrntest(model, sldvDataFile, [], true)
```

Description `data = sldvrntest(model, sldvDataFile, testIdx)` simulates model using input signals associated with a single test case that the Simulink Design Verifier software generated. `testIdx` specifies the index of the test case that the MAT-file, `sldvDataFile`, contains. This function returns `data`, a structure whose fields list the simulation results:

- `T` — Contains the simulation time vector.
- `X` — Contains the simulation state matrix.
- `Y` — Contains the simulation output matrix.

`data = sldvrntest(model, sldvDataFile)` simulates model using all test cases that the MAT-file, `sldvDataFile`, contains.

`[data, cvdo] = sldvrntest(model, sldvDataFile, testIdx, true)` simulates model using the test case that `testIdx` indexes in the MAT-file `sldvDataFile`. The Simulink® Verification and Validation™ software collects model coverage information during the simulation, which the function returns in the `cvdata` object `cvdo`.

`[data, cvdo] = sldvrntest(model, sldvDataFile, [], true)` simulates model using all test cases that the MAT-file, `sldvDataFile`, contains. The Simulink Verification and Validation software collects model coverage information during the simulation, which the function returns in the `cvdata` object `cvdo`.

See Also `cvsim` (in the *Simulink Verification and Validation User's Guide*), `sim` (in the *Simulink® Reference*)

Block Reference

Proof Assumption

Purpose Constrain signal values when proving model properties

Library Simulink Design Verifier

Description



When operating in property proving mode, the Simulink® Design Verifier™ software proves that properties of your model satisfy specified criteria (see Chapter 7, “Proving Properties of a Model”). In this mode, you can use Proof Assumption blocks to define assumptions for signals in your model. The **Values** parameter lets you specify constraints on signal values during a property proof. Use the **Initial** parameter to specify whether the constraint applies throughout the entire proof or only at its beginning. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software proves or disproves that the properties of your model satisfy specified criteria.

The block’s parameter dialog box also allows you to

- Enable or disable the assumption.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink® and Real-Time Workshop® software ignore the Proof Assumption block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Proof Assumption block only when proving model properties.

Specifying Proof Assumptions

Use the **Values** parameter to constrain signal values in property proofs. Specify any combination of scalars and intervals in the form of a MATLAB® cell array (see “Cell Arrays” in the MATLAB documentation for information about working with cell arrays).

Tip If the **Values** parameter specifies only one scalar value, you do not need to enter it in the form of a MATLAB cell array.

Scalar values each comprise a single cell in the array, for example:

```
{0, 5}
```

A closed interval comprises a two-element vector as a cell in the array, where each element specifies an interval endpoint:

```
{[1, 2]}
```

Alternatively, you can specify scalar values using the `Sldv.Point` constructor, which accepts a single value as its argument. You can specify intervals using the `Sldv.Interval` constructor, which requires two input arguments, i.e., a lower bound and an upper bound for the interval. Optionally, you can provide one of the following strings as a third input argument that specifies inclusion or exclusion of the interval endpoints:

- `'()'` — Defines an open interval.
- `'[]'` — Defines a closed interval.
- `'(]'` — Defines a left-open interval.
- `'[)'` — Defines a right-open interval.

Note By default, `Sldv.Interval` considers an interval to be closed if you omit its third input argument.

As an example, the **Values** parameter

```
{0, [1, 3]}
```

specifies:

Proof Assumption

- 0 — a scalar
- [1, 3] — a closed interval

The **Values** parameter

```
{Sldv.Interval(0, 1, '[') ), Sldv.Point(1)}
```

specifies:

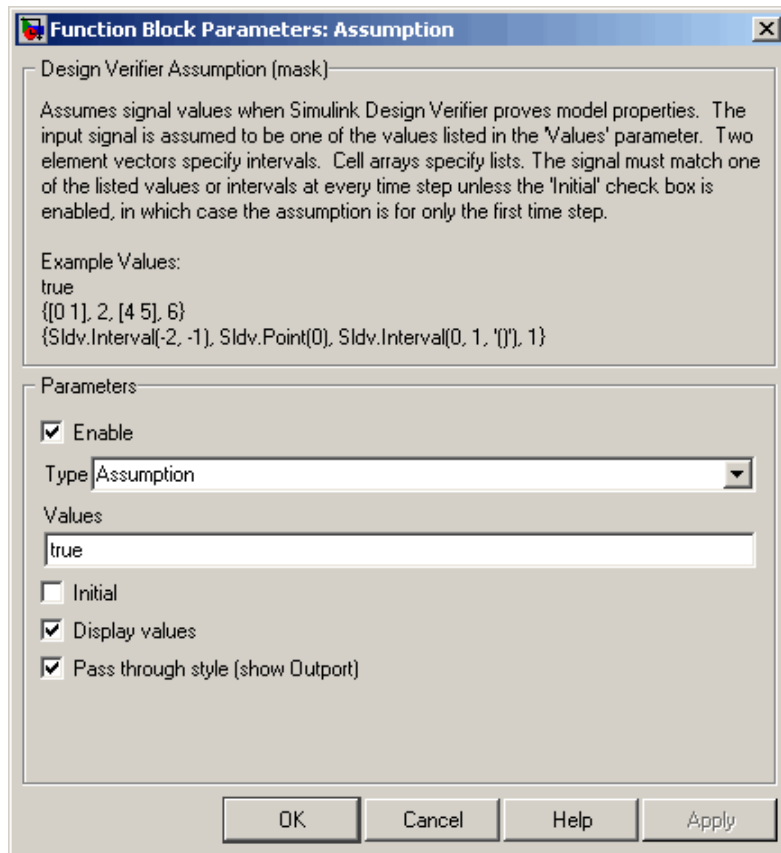
- Sldv.Interval(0, 1, '[') — the right-open interval [0, 1)
- Sldv.Point(1) — a scalar

If you specify multiple scalars and intervals for a Proof Assumption block, the Simulink Design Verifier software combines them using a logical OR operation during the property proof. In this case, the software considers the entire assumption to be satisfied if any single scalar or interval is satisfied.

Data Type Support

The Proof Assumption block accepts signals of all built-in data types supported by the Simulink software. For a discussion on the data types supported by the Simulink software, see “Data Types Supported by Simulink” in *Using Simulink*.

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), the Simulink Design Verifier software uses the block when proving properties of a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Proof Assumption block did not exist. If this option is not selected, the block appears grayed out in the model editor.

Proof Assumption

Type

Specify whether the block behaves as a Proof Assumption or Test Condition block. Select `Test Condition` to transform the Proof Assumption block into a Test Condition block.

Values

Specify the proof assumption (see “Specifying Proof Assumptions” on page 10-2).

Initial

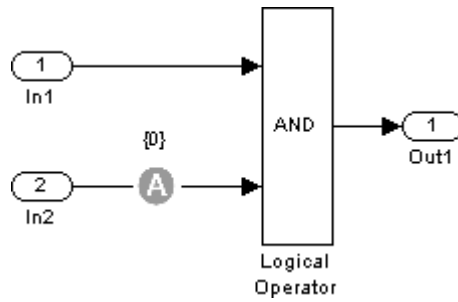
Specify whether the **Values** parameter applies at the beginning of or throughout the entire proof. If selected, the block constrains only the initial value of its input signal at the start of a proof analysis ($t=0$). If not selected (the default), the block constrains its signal value for the entire proof.

Display values

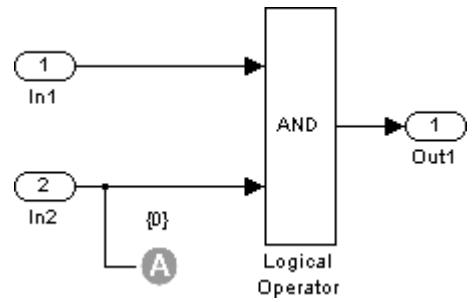
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

Pass through style

Specify whether the block displays an output port in the model editor. If selected (the default), the block displays its output port, allowing its input signal to pass through as the block output. If not selected, the block hides its output port and terminates the input signal. The following figure illustrates the appearance of the block in each case.



Pass through style: selected



Pass through style: deselected

See Also

Proof Objective, Test Condition

Proof Objective

Purpose Define objectives that signals must satisfy when proving model properties

Library Simulink Design Verifier

Description



When operating in property proving mode, the Simulink® Design Verifier™ software proves that properties of your model satisfy specified criteria (see Chapter 7, “Proving Properties of a Model”). In this mode, you can use Proof Objective blocks to define proof objectives for signals in your model. The **Values** parameter lets you specify values that a signal must achieve for at least one time step during a proof. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software proves or disproves that the properties of your model satisfy specified criteria.

The block’s parameter dialog box also allows you to

- Enable or disable the objective.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink® and Real-Time Workshop® software ignore the Proof Objective block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Proof Objective block only when proving model properties.

Specifying Proof Objectives

Use the **Values** parameter to define values that a signal must achieve during a proof simulation. Specify any combination of scalars and intervals in the form of a MATLAB® cell array (see “Cell Arrays” in the MATLAB documentation for information about working with cell arrays).

Tip If the **Values** parameter specifies only one scalar value, you do not need to enter it in the form of a MATLAB cell array.

Scalar values each comprise a single cell in the array, for example:

```
{0, 5}
```

A closed interval comprises a two-element vector as a cell in the array, where each element specifies an interval endpoint:

```
{[1, 2]}
```

Alternatively, you can specify scalar values using the `Sldv.Point` constructor, which accepts a single value as its argument. You can specify intervals using the `Sldv.Interval` constructor, which requires two input arguments, i.e., a lower bound and an upper bound for the interval. Optionally, you can provide one of the following strings as a third input argument that specifies inclusion or exclusion of the interval endpoints:

- `'()'` — Defines an open interval.
- `'[]'` — Defines a closed interval.
- `'(]'` — Defines a left-open interval.
- `'[])'` — Defines a right-open interval.

Note By default, `Sldv.Interval` considers an interval to be closed if you omit its third input argument.

As an example, the **Values** parameter

```
{0, [1, 3]}
```

specifies:

Proof Objective

- 0 — a scalar
- [1, 3] — a closed interval

The **Values** parameter

```
{Sldv.Interval(0, 1, '['), Sldv.Point(1)}
```

specifies:

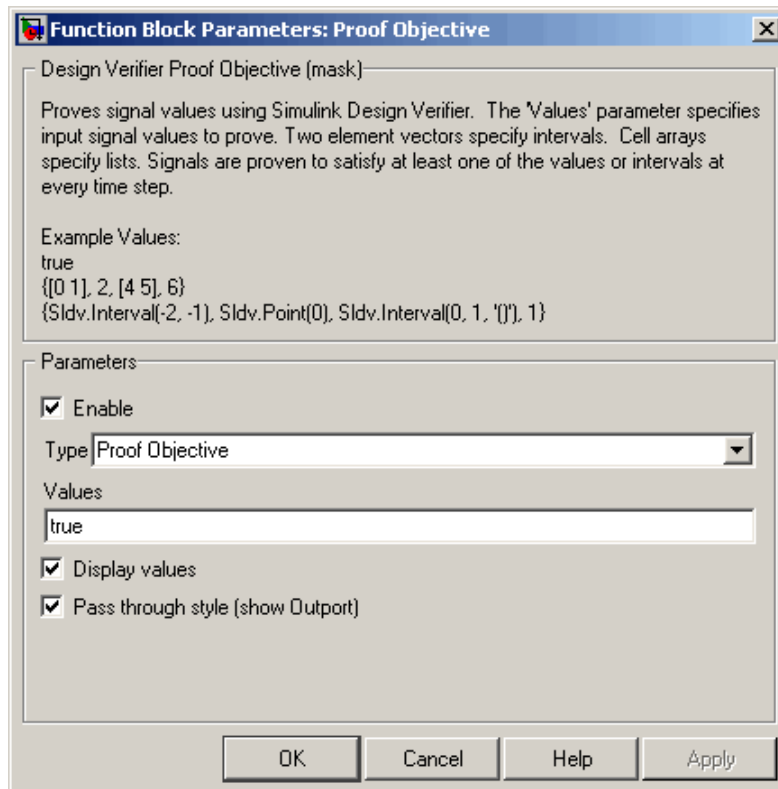
- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

If you specify multiple scalars and intervals for a Proof Objective block, the Simulink Design Verifier software combines them using a logical OR operation during the property proof. In this case, the software considers the entire proof objective to be satisfied if any single scalar or interval is satisfied.

Data Type Support

The Proof Objective block accepts signals of all built-in data types supported by the Simulink software. For a discussion on the data types supported by the Simulink software, see “Data Types Supported by Simulink” in *Using Simulink*.

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), the Simulink Design Verifier software uses the block when proving properties of a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Proof Objective block did not exist. If this option is not selected, the block appears grayed out in the model editor.

Proof Objective

Type

Specify whether the block behaves as a Proof Objective or Test Objective block. Select Test Objective to transform the Proof Objective block into a Test Objective block.

Values

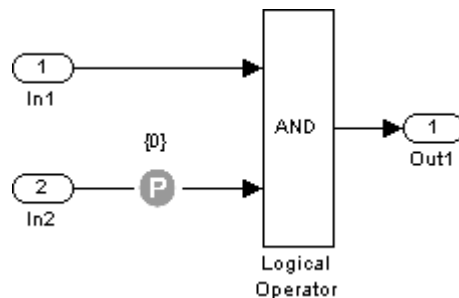
Specify the proof objective (see “Specifying Proof Objectives” on page 10-8).

Display values

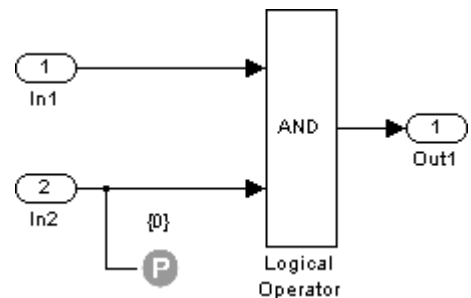
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

Pass through style

Specify whether the block displays an output port in the model editor. If selected (the default), the block displays its output port, allowing its input signal to pass through as the block output. If not selected, the block hides its output port and terminates the input signal. The following figure illustrates the appearance of the block in each case.



Pass through style: selected



Pass through style: deselected

See Also

Proof Assumption, Test Objective

Purpose Constrain signal values in test cases

Library Simulink Design Verifier

Description

true



When operating in test generation mode, the Simulink® Design Verifier™ software produces test cases that satisfy specified criteria (see Chapter 6, “Generating Test Cases”). In this mode, you can use Test Condition blocks to define test conditions for signals in your model. The **Values** parameter lets you specify constraints on signal values during a test case simulation. Use the **Initial** parameter to specify whether the constraint applies throughout the entire test case simulation or only at its beginning. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software attempts to produce test cases that satisfy the condition.

The block’s parameter dialog box also allows you to

- Enable or disable the condition.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink® and Real-Time Workshop® software ignore the Test Condition block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Test Condition block only when generating test cases for a model.

Specifying Test Conditions

Use the **Values** parameter to constrain signal values in test cases. Specify any combination of scalars and intervals in the form of a MATLAB® cell array (see “Cell Arrays” in the MATLAB documentation for information about working with cell arrays).

Test Condition

Tip If the **Values** parameter specifies only one scalar value, you do not need to enter it in the form of a MATLAB cell array.

Scalar values each comprise a single cell in the array, for example:

`{0, 5}`

A closed interval comprises a two-element vector as a cell in the array, where each element specifies an interval endpoint:

`{[1, 2]}`

Alternatively, you can specify scalar values using the `Sldv.Point` constructor, which accepts a single value as its argument. You can specify intervals using the `Sldv.Interval` constructor, which requires two input arguments, i.e., a lower bound and an upper bound for the interval. Optionally, you can provide one of the following strings as a third input argument that specifies inclusion or exclusion of the interval endpoints:

- `'()'` — Defines an open interval.
- `'[]'` — Defines a closed interval.
- `'(]'` — Defines a left-open interval.
- `'][)'` — Defines a right-open interval.

Note By default, `Sldv.Interval` considers an interval to be closed if you omit its third input argument.

As an example, the **Values** parameter

`{0, [1, 3]}`

specifies:

- 0 — a scalar
- [1, 3] — a closed interval

The **Values** parameter

```
{Sldv.Interval(0, 1, '[') }, Sldv.Point(1)}
```

specifies:

- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

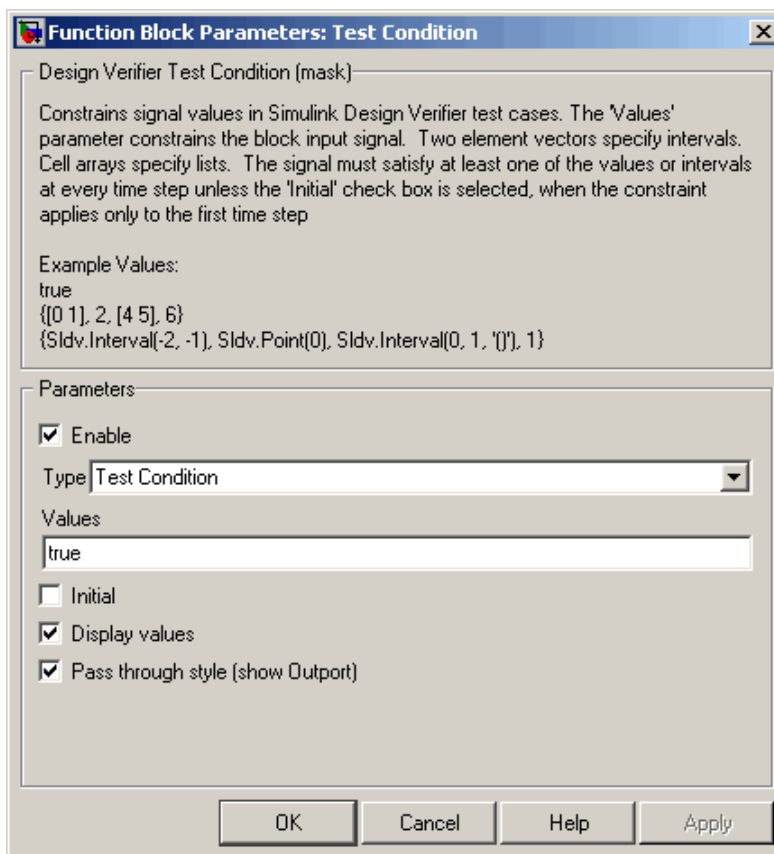
If you specify multiple scalars and intervals for a Test Condition block, the Simulink Design Verifier software combines them using a logical OR operation when generating test cases. Consequently, the software considers the entire test condition to be satisfied if any single scalar or interval is satisfied.

Data Type Support

The Test Condition block accepts signals of all built-in data types supported by the Simulink software. For a discussion on the data types supported by the Simulink software, see “Data Types Supported by Simulink” in *Using Simulink*.

Test Condition

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), the Simulink Design Verifier software uses the block when generating tests for a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Test Condition block did not exist. If this option is not selected, the block appears grayed out in the model editor.

Type

Specify whether the block behaves as a Test Condition or Proof Assumption block. Select Assumption to transform the Test Condition block into a Proof Assumption block.

Values

Specify the test condition (see “Specifying Test Conditions” on page 10-13).

Initial

Specify whether the **Values** parameter applies at the beginning of or throughout the entire test case simulation. If selected, the block constrains only the initial value of its input signal at the start of a test case simulation ($t=0$). If not selected (the default), the block constrains its signal value for the entire test case simulation.

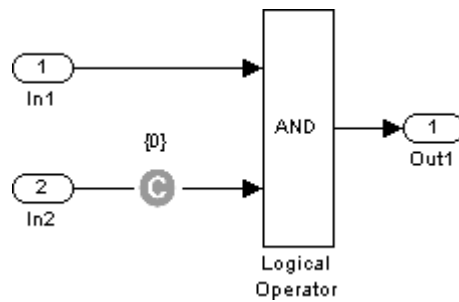
Display values

Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

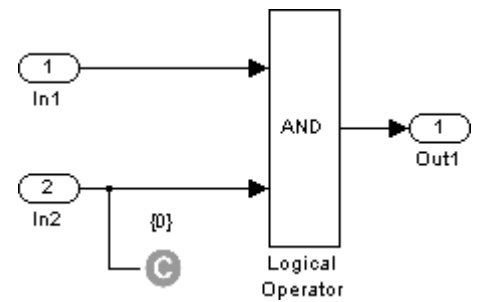
Pass through style

Specify whether the block displays an output port in the model editor. If selected (the default), the block displays its output port, allowing its input signal to pass through as the block output. If not selected, the block hides its output port and terminates the input signal. The following figure illustrates the appearance of the block in each case.

Test Condition



Pass through style: selected



Pass through style: deselected

See Also

Proof Assumption, Test Objective

Purpose

Define custom objectives that signals must satisfy in test cases

Library

Simulink Design Verifier

Description

true



When operating in test generation mode, the Simulink® Design Verifier™ software produces test cases that satisfy specified criteria (see Chapter 6, “Generating Test Cases”). In this mode, you can use Test Objective blocks to define custom test objectives for signals in your model. The **Values** parameter lets you specify values that a signal must achieve for at least one time step during a test case simulation. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software attempts to produce test cases that satisfy the objective.

The block’s parameter dialog box also allows you to

- Enable or disable the objective.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink® and Real-Time Workshop® software ignore the Test Objective block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Test Objective block only when generating test cases for a model.

Specifying Test Objectives

Use the **Values** parameter to define custom objectives that signals must satisfy in test cases. Specify any combination of scalars and intervals in the form of a MATLAB® cell array (see “Cell Arrays” in the MATLAB documentation for information about working with cell arrays).

Test Objective

Tip If the **Values** parameter specifies only one scalar value, you do not need to enter it in the form of a MATLAB cell array.

Scalar values each comprise a single cell in the array, for example:

`{0, 5}`

A closed interval comprises a two-element vector as a cell in the array, where each element specifies an interval endpoint:

`{[1, 2]}`

Alternatively, you can specify scalar values using the `Sldv.Point` constructor, which accepts a single value as its argument. You can specify intervals using the `Sldv.Interval` constructor, which requires two input arguments, i.e., a lower bound and an upper bound for the interval. Optionally, you can provide one of the following strings as a third input argument that specifies inclusion or exclusion of the interval endpoints:

- `'()'` — Defines an open interval.
- `'[]'` — Defines a closed interval.
- `'()'` — Defines a left-open interval.
- `'[]'` — Defines a right-open interval.

Note By default, `Sldv.Interval` considers an interval to be closed if you omit its third input argument.

As an example, the **Values** parameter

`{0, [1, 3]}`

specifies:

- 0 — a scalar
- [1, 3] — a closed interval

The **Values** parameter

```
{Sldv.Interval(0, 1, '[') }, Sldv.Point(1)}
```

specifies:

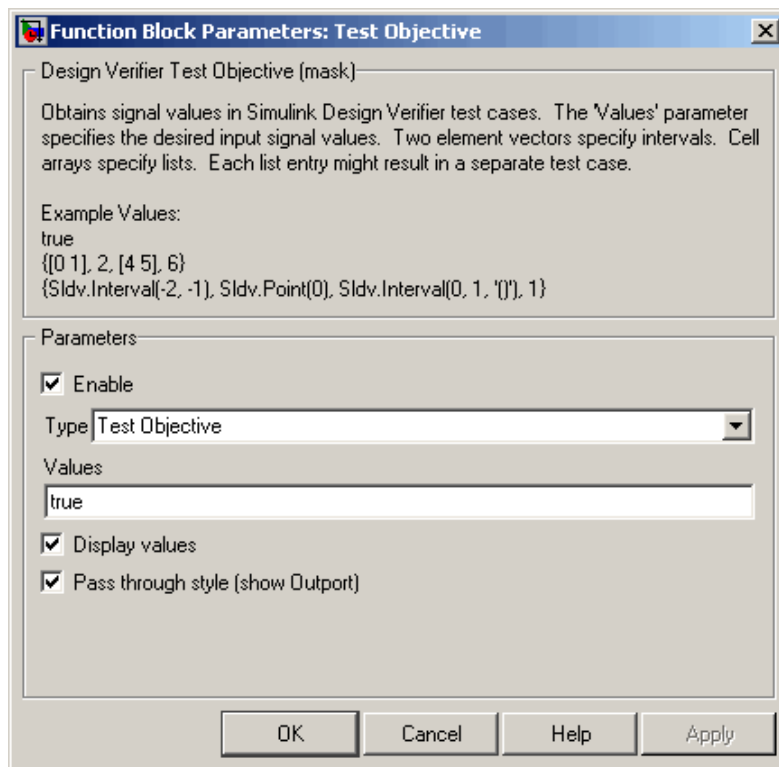
- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

Data Type Support

The Test Objective block accepts signals of all built-in data types supported by the Simulink software. For a discussion on the data types supported by the Simulink software, see “Data Types Supported by Simulink” in *Using Simulink*.

Test Objective

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), the Simulink Design Verifier software uses the block when generating tests for a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Test Objective block did not exist. If this option is not selected, the block appears grayed out in the model editor.

Type

Specify whether the block behaves as a Test Objective or Proof Objective block. Select Proof Objective to transform the Test Objective block into a Proof Objective block.

Values

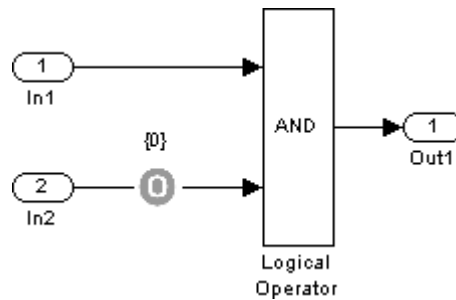
Specify the test objective (see “Specifying Test Objectives” on page 10-19).

Display values

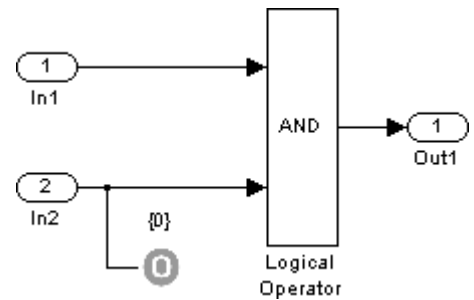
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

Pass through style

Specify whether the block displays an output port in the model editor. If selected (the default), the block displays its output port, allowing its input signal to pass through as the block output. If not selected, the block hides its output port and terminates the input signal. The following figure illustrates the appearance of the block in each case.



Pass through style: selected



Pass through style: deselected

See Also

Proof Objective, Test Condition

Verification Subsystem

Purpose

Represent subsystem that specifies proof or test objectives without impacting simulation results or generated code

Library

Simulink Design Verifier

Description



This block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem that specifies proof or test objectives for use with the Simulink® Design Verifier™ software. The Real-Time Workshop® software ignores Verification Subsystem blocks during code generation, behaving as if the subsystems do not exist. A Verification Subsystem block allows you to add Simulink Design Verifier components to a model without affecting its generated code.

To create a Verification Subsystem in your model:

- 1 Copy the Verification Subsystem block from the Simulink Design Verifier library into your model.
- 2 Open the Verification Subsystem block by double-clicking it.
- 3 In the Verification Subsystem window, add blocks that specify proof or test objectives. Use Inport blocks to represent input from outside the subsystem.

The Verification Subsystem block in the Simulink Design Verifier library is preconfigured to work correctly. For correct behavior, a Verification Subsystem block must

- Contain no Outport blocks.
- Enable its **Treat as Atomic Unit** parameter.
- Specify its **Mask type** parameter as `VerificationSubsystem`.

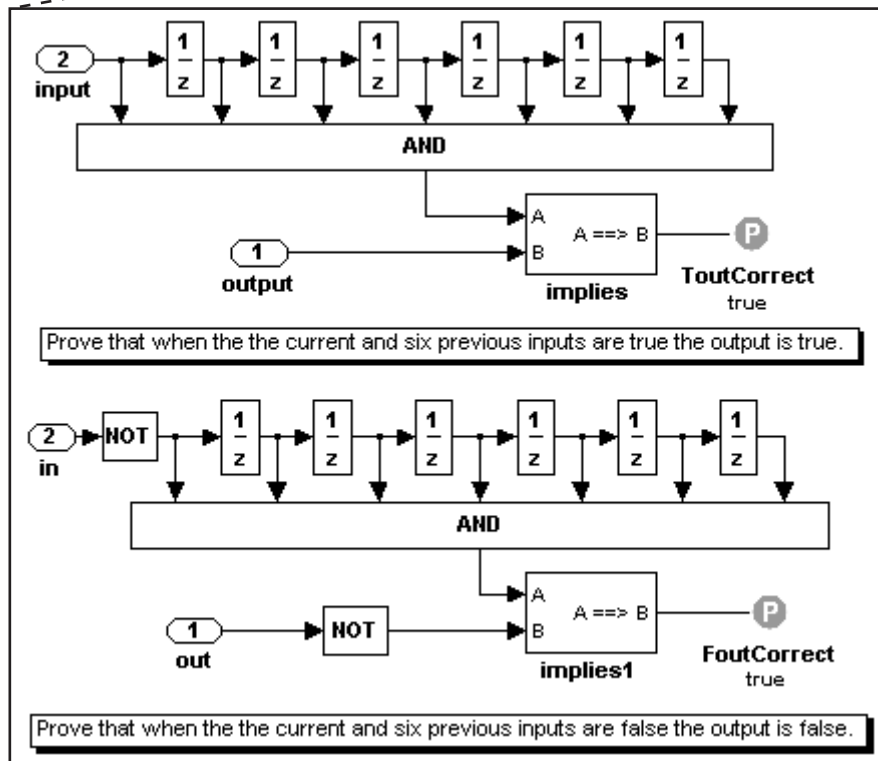
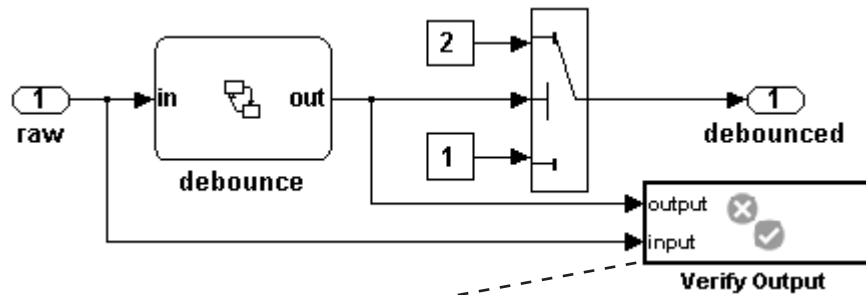
Note If you alter a Verification Subsystem block so that it no longer behaves correctly, the Simulink Design Verifier software displays a warning.

See the Subsystem block in the *Simulink*[®] *Reference* and “Creating Subsystems” in *Using Simulink* for more information.

Examples

The `sldvdemo_debounce_validprop` demo model includes a Verification Subsystem that specifies two proof objectives, as shown in the following figure.

Verification Subsystem



See Also Proof Assumption, Proof Objective, Test Condition, Test Objective

Verification Subsystem

Configuration Parameters

Design Verifier Pane (p. 11-2)	General parameters for controlling analysis options and Simulink® Design Verifier™ output
Design Verifier Pane: Block Replacements (p. 11-9)	Parameters for controlling how the Simulink Design Verifier software preprocesses the models it analyzes
Design Verifier Pane: Parameters (p. 11-14)	Parameters for controlling how the Simulink Design Verifier software uses parameter configurations when analyzing models
Design Verifier Pane: Test Generation (p. 11-18)	Parameters for controlling how the Simulink Design Verifier software generates test cases for models it analyzes
Design Verifier Pane: Property Proving (p. 11-26)	Parameters for controlling how the Simulink Design Verifier software proves properties of the models it analyzes
Design Verifier Pane: Results (p. 11-32)	Parameters for controlling how the Simulink Design Verifier software handles the results that it generates
Design Verifier Pane: Report (p. 11-42)	Parameters for controlling how the Simulink Design Verifier software reports its results
Parameter Command-Line Information Summary (p. 11-48)	Summary of model parameters for configuring the Simulink Design Verifier software

Design Verifier Pane

The image shows a software configuration window titled "Design Verifier Pane". It is divided into two main sections: "Analysis options" and "Output".

Analysis options:

- Mode: A dropdown menu set to "Test generation".
- Maximum analysis time: A text input field containing "600".
- Display unsatisfiable test objectives

Output:

- Output directory: A text input field containing "sldv_output/\$ModelName\$".
- Make output file names unique by adding a suffix

At the bottom right of the window, there are two buttons: "Check Model Compatibility" and "Analyze Model".

In this section...

“Design Verifier Pane Overview” on page 11-3

“Mode” on page 11-4

“Maximum analysis time” on page 11-5

“Display unsatisfiable test objectives” on page 11-6

“Output directory” on page 11-7

“Make output file names unique by adding a suffix” on page 11-8

Design Verifier Pane Overview

Specify analysis options and configure Simulink® Design Verifier™ output.

Mode

Specify whether the Simulink Design Verifier software generates test cases or proves properties.

Settings

Default: Test generation

Test generation
Generates test cases for a model.

Property proving
Proves properties of a model.

Tip

The Simulink Design Verifier software specifies the value of this option automatically if you start an analysis by selecting from the **Tools** menu either **Design Verifier > Generate Tests** or **Design Verifier > Prove Properties**.

Dependency

Selecting Test generation enables the **Display unsatisfiable test objectives** parameter.

Command-Line Information

Parameter: DVMode

Type: string

Value: 'TestGeneration' | 'PropertyProving'

Default: 'TestGeneration'

See Also

- Generating Test Cases
- Proving Properties of a Model

Maximum analysis time

Specify the maximum time (in seconds) that the Simulink Design Verifier software spends analyzing a model.

Settings

Default: 600

The value that you enter represents the maximum number of seconds the Simulink Design Verifier software analyzes your model.

Command-Line Information

Parameter: DVMaxProcessTime

Type: double

Value: any valid value

Default: 600

Display unsatisfiable test objectives

Specify whether to display a warning for unsatisfiable test objectives.

Settings

Default: on



Displays a warning in the Simulation Diagnostics Viewer when the Simulink Design Verifier software is unable to satisfy a test objective.



Does not display a warning when the Simulink Design Verifier software is unable to satisfy a test objective.

Dependency

This parameter is enabled by **Mode**.

Command-Line Information

Parameter: DVDisplayUnsatisfiableObjectives

Type: string

Value: 'on' | 'off'

Default: 'on'

Output directory

Specify a directory to which the Simulink Design Verifier software writes its output.

Settings

Default: sldv_output/\$ModelName\$

- Enter a path that is either absolute or relative to the current directory.
- \$ModelName\$ is a token that represents the model name.

Tip

You can use the following parameters to customize the names and locations of Simulink Design Verifier output:

- **Harness model file name**
- **Data file name**
- **Report file name**
- **File path of the output model**

Command-Line Information

Parameter: DVOutputDir

Type: string

Value: any valid path

Default: 'sldv_output/\$ModelName\$'

Make output file names unique by adding a suffix

Specify whether the Simulink Design Verifier software makes its output file names unique by appending a numeric suffix.

Settings

Default: on



On

Appends an incremental numeric suffix to Simulink Design Verifier output file names. Selecting this option prevents the software from overwriting existing files that have the same name.



Off

Does not append a suffix to Simulink Design Verifier output file names. In this case, the software might overwrite existing files that have the same name.

Command-Line Information

Parameter: DVMakeOutputFilesUnique

Type: string

Value: 'on' | 'off'

Default: 'on'

Design Verifier Pane: Block Replacements

Block replacements

Apply block replacements

List of block replacement rules (in order of priority):

Output model

File path of the output model:

In this section...

“Block Replacements Pane Overview” on page 11-10

“Apply block replacements” on page 11-11

“List of block replacement rules” on page 11-12

“File path of the output model” on page 11-13

Block Replacements Pane Overview

Specify options that control how the Simulink® Design Verifier™ software preprocesses the models it analyzes.

See Also

Working with Block Replacements

Apply block replacements

Specify whether the Simulink Design Verifier software replaces blocks in a model before its analysis.

Settings

Default: off

- On
Replaces blocks in a model before the Simulink Design Verifier software analyzes it.
- Off
Does not replace blocks in a model before the Simulink Design Verifier software analyzes it.

Dependencies

This parameter enables **List of block replacement rules** and **File path of the output model**.

Command-Line Information

Parameter: DVBlockReplacement

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Working with Block Replacements

List of block replacement rules

Specify a list of block replacement rules that the Simulink Design Verifier software executes before its analysis.

Settings

Default: <FactoryDefaultRules>

- Specify block replacement rules as a list delimited by spaces, commas, or carriage returns.
- The Simulink Design Verifier software processes block replacement rules in the order that you list them.
- If you specify the default value, the Simulink Design Verifier software uses its factory default block replacement rules.

Dependency

This parameter is enabled by **Apply block replacements**.

Command-Line Information

Parameter: DVBlockReplacementRulesList

Type: string

Value: any rules

Default: '<FactoryDefaultRules>'

See Also

Working with Block Replacements

File path of the output model

Specify a directory and file name to which the Simulink Design Verifier software saves the model that results after applying block replacement rules.

Settings

Default: \$ModelName\$_replacement

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name by which the Simulink Design Verifier software saves the model that results after applying block replacement rules.
- \$ModelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Apply block replacements**.

Command-Line Information

Parameter: DVBlockReplacementModelFileName

Type: string

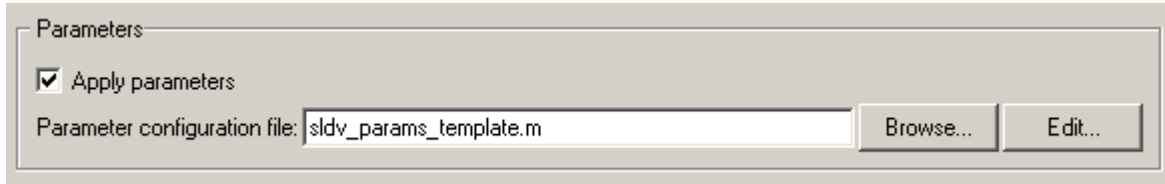
Value: any valid path and file name

Default: '\$ModelName\$_replacement'

See Also

Working with Block Replacements

Design Verifier Pane: Parameters



The image shows a software dialog box titled "Parameters". It contains a checked checkbox labeled "Apply parameters". Below this is a text field labeled "Parameter configuration file:" containing the text "sldv_params_template.m". To the right of the text field are two buttons: "Browse..." and "Edit...".

In this section...

“Parameters Pane Overview” on page 11-15

“Apply parameters” on page 11-16

“Parameter configuration file” on page 11-17

Parameters Pane Overview

Specify options that control how the Simulink® Design Verifier™ software uses parameter configurations when analyzing models.

See Also

Specifying Parameter Configurations

Apply parameters

Specify whether the Simulink Design Verifier software uses parameter configurations when analyzing a model.

Settings

Default: on



On

The Simulink Design Verifier software uses parameter configurations when analyzing a model.



Off

The Simulink Design Verifier software does not use parameter configurations when analyzing a model.

Dependency

This parameter enables **Parameter configuration file**.

Command-Line Information

Parameter: DVParameters

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Specifying Parameter Configurations

Parameter configuration file

Specify an M-file function that defines parameter configurations for a model.

Settings

Default: `sldv_params_template.m`

- The default file, `sldv_params_template.m`, is a template that you can edit and save. The comments in the template explain the syntax you use to specify parameter configurations.
- Click the **Browse** button to select an existing M-file function using a file chooser dialog box.
- Click the **Edit** button to open the specified M-file function in an editor.

Dependency

This parameter is enabled by **Apply parameters**.

Command-Line Information

Parameter: `DVParametersConfigFileName`

Type: string

Value: any valid M-file function

Default: `'sldv_params_template.m'`

See Also

Specifying Parameter Configurations

Design Verifier Pane: Test Generation

Test generation	
Model coverage objectives:	MCDC
Test conditions:	Enable all
Test objectives:	Enable all
Maximum test case steps:	500
Test suite optimization:	Combined objectives

In this section...

“Test Generation Pane Overview” on page 11-19

“Model coverage objectives” on page 11-20

“Test conditions” on page 11-21

“Test objectives” on page 11-22

“Maximum test case steps” on page 11-23

“Test suite optimization” on page 11-24

Test Generation Pane Overview

Specify options that control how the Simulink® Design Verifier™ software generates tests for the models it analyzes.

See Also

Generating Test Cases

Model coverage objectives

Specify the type of model coverage that the Simulink Design Verifier software attempts to achieve.

Settings

Default: MCDC

None

Generates test cases that achieve only the custom objectives that you specified in your model using, for example, Test Objective blocks.

Decision

Generates test cases that achieve decision coverage.

Condition Decision

Generates test cases that achieve condition and decision coverage.

MCDC

Generates test cases that achieve modified condition/decision coverage (MCDC).

Command-Line Information

Parameter: DVModelCoverageObjectives

Type: string

Value: 'None' | 'Decision' | 'ConditionDecision' | 'MCDC'

Default: 'MCDC'

See Also

Generating Test Cases

Test conditions

Specify whether Test Condition blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Test Condition blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Test Condition blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Test Condition blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVTestConditions

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Test Condition
- Generating Test Cases

Test objectives

Specify whether Test Objective blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Test Objective blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Test Objective blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Test Objective blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVTestObjectives

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Test Objective
- Generating Test Cases

Maximum test case steps

Specify the maximum number of simulation steps the Simulink Design Verifier software takes when attempting to satisfy a test objective.

Settings

Default: 500

You can specify a value that represents the maximum number of simulation steps the Simulink Design Verifier software takes when attempting to satisfy a test objective.

Command-Line Information

Parameter: DVMaxTestCaseSteps

Type: int32

Value: any valid value

Default: 500

See Also

Generating Test Cases

Test suite optimization

Specify the optimization strategy to use when generating test cases.

Settings

Default: Combined objectives

Combined objectives

Minimizes the number of test cases in a suite by generating cases that address more than one test objective. Each test case tends to be long, i.e., it includes many time steps.

Individual objectives

Maximizes the number of test cases in a suite by generating cases that each address only one test objective. Each test case tends to be short, i.e., it includes only a few time steps.

Large model

Minimizes the number of test cases in a suite by generating cases that address more than one test objective. This strategy is tailored for large, complex models; consequently, it tends to use all the time that the **Maximum analysis time** option allots.

Tip

If an analysis using the Combined objectives strategy returns objectives without an outcome, set this option to Individual objectives and reanalyze the model. The Individual objectives strategy is better at proving whether objectives are unsatisfiable. However, set this option to Large model if the model has both of the following characteristics:

- Nonlinearities, such as those that result from multiplying or dividing the model's input signals
- Numerous test objectives, such as those that result when using blocks that receive model coverage

The Large model strategy performs an analysis that is tailored to large, complex models; but, this strategy tends to use all the time that the **Maximum analysis time** option allots.

Command-Line Information

Parameter: DVTestSuiteOptimization

Type: string

Value: 'CombinedObjectives' | 'IndividualObjectives' |
'LargeModel'

Default: 'CombinedObjectives'

See Also

Generating Test Cases

Design Verifier Pane: Property Proving

Property proving	
Assertion blocks:	Enable all <input type="button" value="v"/>
Proof assumptions:	Enable all <input type="button" value="v"/>
Strategy:	Find violation <input type="button" value="v"/>
Maximum violation steps:	20

In this section...

“Property Proving Pane Overview” on page 11-27

“Assertion blocks” on page 11-28

“Proof assumptions” on page 11-29

“Strategy” on page 11-30

“Maximum violation steps” on page 11-31

Property Proving Pane Overview

Specify options that control how the Simulink® Design Verifier™ software proves properties for the models it analyzes.

See Also

Proving Properties of a Model

Assertion blocks

Specify whether Assertion blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Assertion blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Assertion blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Assertion blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVAssertions

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Assertion
- Proving Properties of a Model

Proof assumptions

Specify whether Proof Assumption blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Proof Assumption blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Proof Assumption blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Proof Assumption blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVProofAssumptions

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Proof Assumption
- Proving Properties of a Model

Strategy

Specify the strategy that the Simulink Design Verifier software uses when proving properties.

Settings

Default: Prove

Prove

Performs property proofs.

Find violation

Searches for property violations within the number of simulation steps specified by the **Maximum violation steps** option.

Prove with violation detection

Searches for property violations within the number of simulation steps specified by the **Maximum violation steps** option; then it attempts to prove properties for which it failed to detect a violation.

Dependency

Selecting Find violation or Prove with violation detection enables the **Maximum violation steps** parameter.

Command-Line Information

Parameter: DVProvingStrategy

Type: string

Value: 'Prove' | 'FindViolation' | 'ProveWithViolationDetection'

Default: 'Prove'

See Also

Proving Properties of a Model

Maximum violation steps

Specify the maximum number of simulation steps over which the Simulink Design Verifier software searches for property violations.

Settings

Default: 20

The Simulink Design Verifier software does not search beyond the maximum number of simulation steps that you specify. Therefore, it cannot identify violations that might occur later in a simulation.

Dependency

This parameter is enabled by **Strategy**.

Command-Line Information

Parameter: DVMaxViolationSteps

Type: int32

Value: any valid value

Default: 20

See Also

Proving Properties of a Model

Design Verifier Pane: Results

Harness model options

Save test harness as model

Harness model file name:

Data file options

Save test data to file

Data file name:

Include expected output values

Randomize data that do not affect the outcome

In this section...

“Results Pane Overview” on page 11-33

“Save test harness as model” on page 11-34

“Harness model file name” on page 11-35

“Save test data to file” on page 11-36

“Data file name” on page 11-37

“Include expected output values” on page 11-38

“Randomize data that does not affect outcome” on page 11-40

Results Pane Overview

Specify options that control how the Simulink® Design Verifier™ software handles the results that it generates.

See Also

Reviewing the Results

Save test harness as model

Save the test harness that the Simulink Design Verifier software generates as a model file.

Settings

Default: on



On

Saves the test harness that the Simulink Design Verifier software generates as a model file.



Off

Does not save the test harness that the Simulink Design Verifier software generates.

Dependency

This parameter enables **Harness model file name**.

Command-Line Information

Parameter: DVSaveHarnessModel

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Harness model file name

Specify a directory and file name with which the Simulink Design Verifier software saves the test harness it generates.

Settings

Default: \$modelName\$_harness

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name by which the Simulink Design Verifier software saves the test harness it generates.
- \$modelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Save test harness as model**.

Command-Line Information

Parameter: DVHarnessModelFileName

Type: string

Value: any valid path and file name

Default: '\$modelName\$_harness'

See Also

Reviewing the Results

Save test data to file

Save the test data that the Simulink Design Verifier software generates to a MAT-file.

Settings

Default: on



On

Saves the test data that the Simulink Design Verifier software generates to a MAT-file.



Off

Does not save the test data that the Simulink Design Verifier software generates.

Dependency

This parameter enables **Data file name**.

Command-Line Information

Parameter: DVSaveDataFile

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Data file name

Specify a directory and file name with which the Simulink Design Verifier software saves the MAT-file it generates.

Settings

Default: \$ModelName\$_sldvdata

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name by which the Simulink Design Verifier software saves the MAT-file it generates.
- \$ModelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Save test data to file**.

Command-Line Information

Parameter: DVDataFileName

Type: string

Value: any valid path and file name

Default: '\$ModelName\$_sldvdata'

See Also

Reviewing the Results

Include expected output values

Simulate the model using test case signals and include the output values in the Simulink Design Verifier data file.

Settings

Default: off



On

Simulates the model using the test case signals that the Simulink Design Verifier software produces. For each test case, the software collects the simulation output values associated with Outport blocks in the top-level system and includes those values in the MAT-file that it generates.



Off

Does not simulate the model and collect output values for inclusion in the MAT-file that the Simulink Design Verifier software generates.

Tips

- The `TestCases.expectedOutput` subfield of the MAT-file contains the output values. For more information, see “Anatomy of the `sldvData` Structure”.
- When **Include expected output values** is enabled, the Simulink Design Verifier software successively simulates the model using each test case that it generates. Enabling this option requires more time for the Simulink Design Verifier software to complete its analysis.

Dependency

This parameter is enabled by **Save test data to file**.

Command-Line Information

Parameter: DVSaveExpectedOutput

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

Randomize data that does not affect outcome

Use random values instead of zeros for input signals that have no impact on test or proof objectives.

Settings

Default: off



On

Assigns random values to test case or counterexample signals that do not affect the outcome of test or proof objectives in a model. This option can enhance traceability and improve your regression tests.

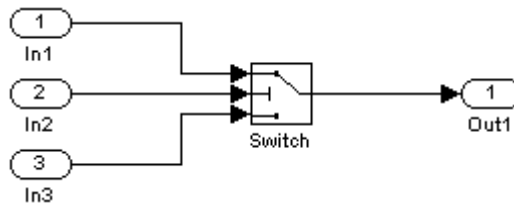


Off

Assigns zeros to test case or counterexample signals that do not affect the outcome of test or proof objectives in a model.

Tips

- This option assigns random values to test case or counterexample signals that otherwise would be zero. In the Simulink Design Verifier report, the Generated Input Data table always displays a dash (–) for such signals.
- Enable this option to enhance traceability when simulating test cases or counterexamples. For instance, consider the following model:



Only the signal entering the Switch block's control port impacts its decision coverage. If the **Randomize data that does not affect outcome** parameter is off, the Simulink Design Verifier software uses zeros to represent the signals from In1 and In3. When inspecting the results from test case or counterexample simulations, it is unclear which of these signals passes through the Switch block because they have the same value. But if the **Randomize data that does not affect outcome** parameter is on, the

software uses unique values to represent each of those signals. In this case, it is easier to determine which signal passes through the Switch block.

Dependency

This parameter is enabled by **Save test data to file**.

Command-Line Information

Parameter: DVRandomizeNoEffectData

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

Design Verifier Pane: Report

Report

Generate report of the results

Report file name:

Include screen shots and plots

Display report

In this section...

“Report Pane Overview” on page 11-43

“Generate report of the results” on page 11-44

“Report file name” on page 11-45

“Include screen shots and plots” on page 11-46

“Display report” on page 11-47

Report Pane Overview

Specify options that control how the Simulink® Design Verifier™ software reports its results.

See Also

Reviewing the Results

Generate report of the results

Generate and save a Simulink Design Verifier report.

Settings

Default: on

On
Saves the HTML report that the Simulink Design Verifier software generates.

Off
Does not generate a Simulink Design Verifier report.

Dependencies

When this parameter is enabled, you must enable **Save test harness as model**.

This parameter enables the following parameters:

- **Report file name**
- **Include screen shots and plots**
- **Display report**

Command-Line Information

Parameter: DVSaveReport

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Report file name

Specify a directory and file name with which the Simulink Design Verifier software saves the report it generates.

Settings

Default: \$ModelName\$_report

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name by which the Simulink Design Verifier software saves the report it generates.
- \$ModelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Generate report of the results**.

Command-Line Information

Parameter: DVReportFileName

Type: string

Value: any valid path and file name

Default: '\$ModelName\$_report'

See Also

Reviewing the Results

Include screen shots and plots

Include images in the report that the Simulink Design Verifier software generates after completing its analysis.

Settings

Default: off



On

Includes images in the report that the Simulink Design Verifier software generates after completing its analysis. Specifically, the report displays images of your model and any signals that comprise its test cases or counterexamples.



Off

Suppresses images in the report that the Simulink Design Verifier software generates after completing its analysis.

Dependency

This parameter is enabled by **Generate report of the results**.

Command-Line Information

Parameter: DVReportIncludeGraphics

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

Display report

Display the report that the Simulink Design Verifier software generates after completing its analysis.

Settings

Default: on



On

Displays the report that the Simulink Design Verifier software generates after completing its analysis.



Off

Does not display the report that the Simulink Design Verifier software generates after completing its analysis.

Dependency

This parameter is enabled by **Generate report of the results**.

Command-Line Information

Parameter: DVDisplayReport

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Parameter Command-Line Information Summary

The following table lists parameters that you can use to configure the behavior of the Simulink® Design Verifier™ software. Use the `get_param` and `set_param` functions to retrieve and specify values for these parameters programmatically.

For each parameter listed in the table, the **Description** column indicates where you can set its value on the Configuration Parameters dialog box. The **Values** column shows the type of value required, the possible values (separated with a vertical line), and the default value (enclosed in braces).

Parameter	Description	Values
DVAssertions	Set by the Assertion blocks option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
DVBlockReplacement	Set by the Apply block replacements option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	'on' {'off'}
DVBlockReplacementModel-FileName	Set by the File path of the output model option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	string {'\$modelName\$_replacement'}
DVBlockReplacementRules-List	Set by the List of block replacement rules option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	string {'<FactoryDefaultRules>'}

Parameter	Description	Values
DVDataFileName	Set by the Data file name option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	string { '\$ModelName\$_sldvdata' }
DVDisplayReport	Set by the Display report option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	{ 'on' } 'off'
DVDisplayUnsatisfiable-Objectives	Set by the Display unsatisfiable test objectives option on the Design Verifier pane of the Configuration Parameters dialog box.	{ 'on' } 'off'
DVHarnessModelFileName	Set by the Harness model file name option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	string { '\$ModelName\$_harness' }
DVMakeOutputFilesUnique	Set by the Make output file names unique by adding a suffix check box on the Design Verifier pane of the Configuration Parameters dialog box.	{ 'on' } 'off'
DVMaxProcessTime	Set by the Maximum analysis time option on the Design Verifier pane of the Configuration Parameters dialog box.	double { '600' }

Parameter	Description	Values
DVMaxTestCaseSteps	Set by the Maximum test case steps option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	int32 {'500'}
DVMaxViolationSteps	Set by the Maximum violation steps option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	int32 {'20'}
DVMode	Set by the Mode option on the Design Verifier pane of the Configuration Parameters dialog box.	{'TestGeneration'} 'PropertyProving'
DVModelCoverageObjectives	Set by the Model coverage objectives option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'None' 'Decision' 'ConditionDecision' {'MCDC'}
DVOutputDir	Set by the Output directory option on the Design Verifier pane of the Configuration Parameters dialog box.	string { 'sldv_output/\$ModelName\$' }
DVParameters	Set by the Apply parameters option on the Design Verifier > Parameters pane of the Configuration Parameters dialog box.	{'on'} 'off'

Parameter	Description	Values
DVParametersConfigFileName	Set by the Parameter configuration file option on the Design Verifier > Parameters pane of the Configuration Parameters dialog box.	string {'sldv_params_template.m'}
DVProofAssumptions	Set by the Proof assumptions option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
DVProvingStrategy	Set by the Strategy option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'FindViolation' {'Prove'} 'ProveWithViolationDetection'
DVRandomizeNoEffectData	Set by the Randomize data that does not affect outcome option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}
DVReportFileName	Set by the Report file name option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	string {'\$ModelName\$_report'}
DVReportIncludeGraphics	Set by the Include screen shots and plots option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	'on' {'off'}

Parameter	Description	Values
DVSaveDataFile	Set by the Save test data to file option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	{'on'} 'off'
DVSaveExpectedOutput	Set by the Include expected output values option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}
DVSaveHarnessModel	Set by the Save test harness as model option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	{'on'} 'off'
DVSaveReport	Set by the Generate report of the results option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	{'on'} 'off'
DVTestConditions	Set by the Test conditions option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}

Parameter	Description	Values
DVTestObjectives	Set by the Test objectives option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
DVTestSuiteOptimization	Set by the Test suite optimization option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	{'CombinedObjectives'} 'IndividualObjectives' 'LargeModel'

Simulink[®] Block Support

The following table summarizes the Simulink® Design Verifier™ software's support for Simulink® blocks. For each block, the third column indicates any support notes (SNs), which provide information you will need when using the block with the Simulink Design Verifier software. All support notes appear at the end of the table.

Sublibrary	Block	Support Notes
Additional Math and Discrete: Additional Discrete	Fixed-Point State-Space	Not supported
	Transfer Fcn Direct Form II	Not supported
	Transfer Fcn Direct Form II Time Varying	Not supported
	Unit Delay Enabled	—
	Unit Delay Enabled External IC	—
	Unit Delay Enabled Resettable	—
	Unit Delay Enabled Resettable External IC	—
	Unit Delay External IC	—
	Unit Delay Resettable	—
	Unit Delay Resettable External IC	—
	Unit Delay With Preview Enabled	—
	Unit Delay With Preview Enabled Resettable	—
	Unit Delay With Preview Enabled Resettable External RV	—
	Unit Delay With Preview Resettable	—
Unit Delay With Preview Resettable External RV	—	
Additional Math and Discrete: Increment/Decrement	Decrement Real World	—
	Decrement Stored Integer	—
	Decrement Time To Zero	Not supported
	Decrement To Zero	—
	Increment Real World	—
	Increment Stored Integer	—

Sublibrary	Block	Support Notes
Continuous	Derivative	Not supported
	Integrator	SN10
	State-Space	Not supported
	Transfer Fcn	Not supported
	Transport Delay	Not supported
	Variable Time Delay	Not supported
	Variable Transport Delay	Not supported
	Zero-Pole	Not supported
Discontinuities	Backlash	Not supported
	Coulomb & Viscous Friction	—
	Dead Zone	Not supported
	Dead Zone Dynamic	—
	Hit Crossing	—
	Quantizer	—
	Rate Limiter	SN11
	Rate Limiter Dynamic	—
	Relay	Not supported
	Saturation	—
	Saturation Dynamic	—
	Wrap To Zero	—

Sublibrary	Block	Support Notes
Discrete	Difference	—
	Discrete Derivative	Not supported
	Discrete Filter	Not supported
	Discrete FIR Filter	—
	Discrete State-Space	Not supported
	Discrete Transfer Fcn	Not supported
	Discrete Zero-Pole	Not supported
	Discrete-Time Integrator	—
	First-Order Hold	—
	Integer Delay	Not supported
	Memory	—
	Tapped Delay	Not supported
	Transfer Fcn First Order	—
	Transfer Fcn Lead or Lag	—
	Transfer Fcn Real Zero	—
	Unit Delay	—
Zero-Order Hold	—	

Sublibrary	Block	Support Notes
Logic and Bit Operations	Bit Clear	—
	Bit Set	—
	Bitwise Operator	—
	Combinatorial Logic	Not supported
	Compare To Constant	—
	Compare To Zero	—
	Detect Change	—
	Detect Decrease	—
	Detect Fall Negative	—
	Detect Fall Nonpositive	—
	Detect Increase	—
	Detect Rise Nonnegative	—
	Detect Rise Positive	—
	Extract Bits	—
	Interval Test	—
	Interval Test Dynamic	—
	Logical Operator	—
	Relational Operator	—
Shift Arithmetic	SN16	

Sublibrary	Block	Support Notes
Lookup Tables	Cosine	Not supported
	Direct Lookup Table (n-D)	Not supported
	Interpolation Using Prelookup	Not supported
	Lookup Table	SN1
	Lookup Table (2-D)	SN1
	Lookup Table (n-D)	SN1, SN2, SN5
	Lookup Table Dynamic	Not supported
	Prelookup	Not supported
	Sine	Not supported

Sublibrary	Block	Support Notes
Math Operations	Abs	—
	Add	—
	Algebraic Constraint	—
	Assignment	—
	Bias	—
	Complex to Magnitude-Angle	—
	Complex to Real-Imag	—
	Divide	—
	Dot Product	—
	Gain	SN7
	Magnitude-Angle to Complex	Not supported
	Math Function	SN3
	Matrix Concatenate	—
	MinMax	—
	MinMax Running Resettable	—
	Permute Dimensions	—
	Polynomial	—
	Product	SN7
	Product of Elements	SN7
	Real-Imag to Complex	Not supported
	Reshape	—
Rounding Function	—	
Sign	—	

Sublibrary	Block	Support Notes
Math Operations (continued)	Sine Wave Function	Not supported
	Slider Gain	—
	Squeeze	—
	Subtract	—
	Sum	—
	Sum of Elements	—
	Trigonometric Function	Not supported
	Unary Minus	Not supported
	Vector Concatenate	—
	Weighted Sample Time Math	Not supported
Model Verification	Assertion	SN8
	Check Discrete Gradient	—
	Check Dynamic Gap	—
	Check Dynamic Lower Bound	—
	Check Dynamic Range	—
	Check Dynamic Upper Bound	—
	Check Input Resolution	—
	Check Static Gap	—
	Check Static Lower Bound	—
	Check Static Range	—
	Check Static Upper Bound	—

Sublibrary	Block	Support Notes
Ports & Subsystems	Atomic Subsystem	—
	Code Reuse Subsystem	—
	Configurable Subsystem	—
	Enabled Subsystem	—
	Enabled and Triggered Subsystem	SN15
	For Iterator Subsystem	—
	Function-Call Generator	—
	Function-Call Subsystem	SN13
	If	SN17
	If Action Subsystem	—
	Model	Not supported
	Subsystem	—
	Switch Case	—
	Switch Case Action Subsystem	—
	Triggered Subsystem	SN15
While Iterator Subsystem	—	

Sublibrary	Block	Support Notes
Signal Attributes	Bus to Vector	—
	Data Type Conversion	—
	Data Type Conversion Inherited	—
	Data Type Duplicate	—
	Data Type Propagation	—
	Data Type Scaling Strip	—
	IC	—
	Probe	—
	Rate Transition	—
	Signal Conversion	—
	Signal Specification	—
	Weighted Sample Time	Not supported
	Width	Not supported

Sublibrary	Block	Support Notes
Signal Routing	Bus Assignment	SN6
	Bus Creator	SN6
	Bus Selector	SN6
	Data Store Memory	—
	Data Store Read	—
	Data Store Write	—
	Demux	—
	Environment Controller	—
	From	—
	Goto	—
	Goto Tag Visibility	—
	Index Vector	—
	Manual Switch	—
	Merge	—
	Multiport Switch	—
	Mux	—
	Selector	—
Switch	—	

Sublibrary	Block	Support Notes
Sinks	Display	—
	Floating Scope	—
	Outport (Out1)	SN12
	Scope	—
	Stop Simulation	Not supported
	Terminator	—
	To File	—
	To Workspace	—
XY Graph	—	

Sublibrary	Block	Support Notes
Sources	Band-Limited White Noise	Not supported
	Chirp Signal	Not supported
	Clock	—
	Constant	—
	Counter Free-Running	—
	Counter Limited	—
	Digital Clock	—
	From File	Not supported
	From Workspace	Not supported
	Ground	—
	Inport (In1)	SN12
	Pulse Generator	SN9
	Ramp	—
	Random Number	Not supported
	Repeating Sequence	Not supported
	Repeating Sequence Interpolated	Not supported
	Repeating Sequence Stair	—
	Signal Builder	Not supported
	Signal Generator	Not supported
	Sine Wave	Not supported
Step	—	
Uniform Random Number	Not supported	

Sublibrary	Block	Support Notes
User-Defined	Embedded MATLAB Function	SN14
	Fcn	SN4, SN17
	Level-2 M-file S-Function	Not supported
	MATLAB Fcn	Not supported
	S-Function	Not supported
	S-Function Builder	Not supported

Symbol	Note
—	The Simulink Design Verifier software supports the block and requires no special notes.
SN1	Input and output must have the same data type, either single or double.
SN2	Not supported when either the Interpolation method or the Extrapolation method parameter specifies Cubic Spline.
SN3	Supports the following Function parameter settings for floating-point input and output signals: magnitude ² , square, conj, reciprocal, mod, transpose, hermitian. Supports the following Function parameter settings for integer or fixed-point input and output signals: sqrt, conj.
SN4	Supports all operators except ^, and supports only the mathematical functions abs, ceil, fabs, floor, rem, and sgn.
SN5	Supports only Number of table dimensions that specify either 1 or 2.
SN6	Supports only virtual signal buses.
SN7	Supports only the Element-wise option for the Multiplication parameter.

Symbol	Note
SN8	Not supported when input signals specify a fixed-point data type using [Slope Bias] scaling.
SN9	Supports only <code>Sample</code> based for the Pulse type parameter; also, must specify a discrete sample time.
SN10	The Simulink Design Verifier software supports the Integrator block only when its parameters have the following settings: <ul style="list-style-type: none"> • External reset — none • Initial condition source — internal • Limit output — Off • Show saturation port — Off • Show state port — Off
SN11	Supports only input and output signals of data type <code>single</code> or <code>double</code> .
SN12	Not supported when the Specify properties via bus object parameter is selected.
SN13	Subsystem analysis is not supported for Function-Call Subsystem blocks that are triggered by wide (i.e., nonscalar) signals.
SN14	See “Limitations of Support for the Embedded MATLAB™ Subset” on page 2-7 for more information.
SN15	Not supported when the trigger control signal specifies a fixed-point data type.
SN16	Not supported when the Number of bits to shift right parameter specifies a vector and the block’s input or output signal has a data type other than <code>single</code> or <code>double</code> .
SN17	Parameter configurations are not supported for If and Fcn blocks. The Simulink Design Verifier software ignores any parameter configurations that you specify for these blocks.

Embedded MATLAB™ Subset Support

The following table lists only the Embedded MATLAB™ library functions for which the Simulink® Design Verifier™ software provides no support or limited support. See “Embedded MATLAB Function Library Reference” for the complete listing of available functions.

Function	Support Notes
Arithmetic Operator Functions	
mldivide (\)	Supports only scalar arguments.
mpower (^)	Supports only integer exponents.
mrdivide (/)	Supports only scalar arguments.
power (.^)	Supports only integer exponents.
Casting Functions	
char	Not supported.
typecast	Not supported.
Complex Number Functions	
complex	Not supported.
imag	Not supported.
Error Handling Functions	
assert	Supported, but does not behave like a Proof Objective block.
Exponential Functions	
exp	Not supported.
expm	Not supported.
expm1	Not supported.
log	Not supported.
log2	Not supported.
log10	Not supported.
log1p	Not supported.
nextpow2	Not supported.

Function	Support Notes
nthroot	Not supported.
reallog	Not supported.
realpow	Not supported.
realsqrt	Not supported.
sqrt	Not supported.
Filtering and Convolution Functions	
detrend	Not supported.
Fixed-Point Toolbox™ Functions	
complex	Not supported.
isfinite	Supported, but returns logical 1 (true) for infinite or NaN values.
isinf	Supported, but always returns logical 0 (false).
isnan	Supported, but always returns logical 0 (false).
Interpolation and Computational Geometry	
cart2pol	Not supported.
cart2sph	Not supported.
pol2cart	Not supported.
sph2cart	Not supported.
Matrix and Array Functions	
angle	Not supported.
cond	Not supported.
det	Not supported.
eig	Not supported.
inv	Not supported.
invhilb	Not supported.
isfinite	Supported, but returns logical 1 (true) for infinite or NaN values.

Function	Support Notes
isinf	Supported, but always returns logical 0 (false).
isnan	Supported, but always returns logical 0 (false).
logspace	Not supported.
lu	Not supported.
norm	Supported only when invoked using the syntax $\text{norm}(A, p)$ where p is either 1 or inf.
normest	Not supported.
pinv	Not supported.
planerot	Not supported.
qr	Not supported.
rank	Not supported.
rcond	Not supported.
subspace	Not supported.
Polynomial Functions	
poly	Not supported.
polyfit	Not supported.
Signal Processing Functions	
chol	Not supported.
fft	Not supported.
fftshift	Not supported.
ifft	Not supported.
ifftshift	Not supported.
sosfilt	Not supported.
svd	Not supported.

Function	Support Notes
Special Values	
rand	Not supported.
randn	Not supported.
Specialized Math	
beta	Not supported.
betainc	Not supported.
betaln	Not supported.
ellipke	Not supported.
erf	Not supported.
erfc	Not supported.
erfcinv	Not supported.
erfcx	Not supported.
erfinv	Not supported.
expint	Not supported.
gamma	Not supported.
gammainc	Not supported.
gammaln	Not supported.
Statistical Functions	
std	Not supported.
String Functions	
char	Not supported.
ischar	Not supported.
Structure Functions	
struct	Not supported.
isstruct	Not supported.
Trigonometric Functions	

Function	Support Notes
acos	Not supported.
acosd	Not supported.
acosh	Not supported.
acot	Not supported.
acotd	Not supported.
acoth	Not supported.
acsc	Not supported.
acscd	Not supported.
acsch	Not supported.
asec	Not supported.
asecd	Not supported.
asech	Not supported.
asin	Not supported.
asinh	Not supported.
atan	Not supported.
atan2	Not supported.
atand	Not supported.
atanh	Not supported.
cos	Not supported.
cosd	Not supported.
cosh	Not supported.
cot	Not supported.
cotd	Not supported.
coth	Not supported.
csc	Not supported.
cscd	Not supported.

Function	Support Notes
csch	Not supported.
hypot	Not supported.
sec	Not supported.
secd	Not supported.
sech	Not supported.
sin	Not supported.
sind	Not supported.
sinh	Not supported.
tan	Not supported.
tand	Not supported.
tanh	Not supported.

analysis model

The target model for a Simulink® Design Verifier™ analysis. If you select an atomic subsystem for analysis, the analysis model is generated by extracting the subsystem to a new model.

assumption

A property that is assumed to be true during a property proof. The proof result holds only when the assumption is true.

block replacement rule

A rule that is registered with the Simulink® Design Verifier™ software and defines how instances of specific blocks will be replaced by an alternate implementation. The software uses M-code to define when and how to apply a block replacement rule (see Chapter 3, “Working with Block Replacements”).

condition coverage

Measures the percentage of the total number of logic conditions associated with logical model objects that the simulation actually exercised. See “Using Model Coverage” in the *Simulink® Verification and Validation™ User’s Guide*.

constraint

A property that is forced to be true during test case generation.

counterexample

A test case that demonstrates a property violation.

coverage objective

A test objective that defines when a coverage point results in a particular outcome.

coverage point

A decision, condition, or MCDC expression associated with a model object. Each coverage point has a fixed number of mutually exclusive outcomes.

decision coverage

Measures the percentage of the total number of simulation paths through model objects that the simulation actually traversed. See “Using Model Coverage” in the *Simulink® Verification and Validation™ User’s Guide*.

floating-point approximation

The process of approximating floating-point numbers using rational numbers (i.e., fractions whose numerator and denominator are small integers). The Simulink® Design Verifier™ software performs floating-point approximations during its analysis. It can generate invalid test cases that result from numerical differences. For example, given a sufficiently large floating-point number x , the expression $x == (x+1)$ will be true; however, this expression will never hold if x is a rational number.

invalid test case

A test case that does not satisfy its objectives.

Modified Condition/Decision Coverage (MCDC)

Measures the independence of logical block inputs and transition conditions associated with logical model objects during the simulation. See “Using Model Coverage” in the *Simulink® Verification and Validation™ User’s Guide*.

nonlinear arithmetic

A computation in the model that cannot be expressed as a combination of mutually exclusive linear expressions. Nonlinear arithmetic can affect a property or test objective, and it can cause the analysis to return an error. In this case, you should apply simplifying approximations and abstractions.

property

A logical expression of the signals and data values, within a model, that is intended to be proven true during simulation. Properties evaluate at specific points in the model.

property violation

The condition during a simulation when a property is false.

test case

A sequence of numeric values and input data time that you input to a model during its simulation.

test harness

A model that runs test cases on an analysis model.

test objective

A logical expression of the signals and data values, within a model, that is intended to be true at least once in the resulting test case during simulation. Test objectives evaluate at specific points in the model.

Test Objective block

The block that you add to a model to define test objectives. In the block mask, define test objectives as values or ranges that an input signal must satisfy during a test case.

unsatisfiable test objective

The status of a test objective that indicates a test case cannot be generated for the specified approximations. This includes floating-point approximations and maximum-step limitations specified in the **Test Generation** pane of the Configuration Parameters dialog box.

validated property

The status of a property that indicates no counterexample exists, subject to floating-point approximations and the settings specified in the **Property Proving** pane of the Configuration Parameters dialog box.

Examples

Use this list to find examples in the documentation.

Working with Block Replacements

- “Constructing Replacement Blocks” on page 3-7
- “Writing Block Replacement Rules” on page 3-10
- “Configuring Block Replacements” on page 3-15

Specifying Parameter Configurations

- “Constructing the Example Model” on page 4-8
- “Parameterizing the Constant Block” on page 4-11
- “Specifying a Parameter Configuration” on page 4-12
- “Analyzing the Example Model” on page 4-13
- “Simulating the Test Cases” on page 4-16

Generating Test Cases

- “Constructing the Example Model” on page 6-5
- “Checking Compatibility of the Example Model” on page 6-6
- “Configuring Test Generation Options” on page 6-10
- “Analyzing the Example Model” on page 6-13
- “Customizing Test Generation” on page 6-21
- “Reanalyzing the Example Model” on page 6-25

Proving Properties of a Model

- “Constructing the Example Model” on page 7-5
- “Instrumenting the Example Model” on page 7-10
- “Configuring Property Proving Options” on page 7-13
- “Analyzing the Example Model” on page 7-15
- “Customizing the Example Proof” on page 7-23
- “Reanalyzing the Example Model” on page 7-25

B

- block replacements
 - configuration 3-15
 - example 3-7
 - execution 3-16
 - factory defaults 3-3
 - introduction 3-2
 - template 3-6
- block support
 - limitations 2-3
 - summary 12-2

C

- configuration parameters
 - block replacements 5-6
 - Block Replacements pane 11-10
 - Apply block replacements 11-11
 - File path of the output model 11-13
 - List of block replacement rules 11-12
 - Design Verifier 5-5
 - Design Verifier pane 11-3
 - Display unsatisfiable test objectives 11-6
 - Make output file names unique by adding a suffix 11-8
 - Maximum analysis time 11-5
 - Mode 11-4
 - Output directory 11-7
 - parameters 5-8
 - Parameters pane 11-15
 - Apply parameters 11-16
 - Parameter configuration file 11-17
 - property proving 5-11
 - Property Proving pane 11-27
 - Assertion blocks 11-28
 - Maximum violation steps 11-31
 - Proof assumptions 11-29
 - Strategy 11-30
 - report 5-14

- Report pane 11-43
 - Display report 11-47
 - Generate report of the results 11-44
 - Include screen shots and plots 11-46
 - Report file name 11-45
- results 5-12
- Results pane 11-33
 - Data file name 11-37
 - Harness model file name 11-35
 - Include expected output values 11-38
 - Randomize data that does not affect outcome 11-40
 - Save test data to file 11-36
 - Save test harness as model 11-34
- summary 11-48
- test generation 5-9
- Test Generation pane 11-19
 - Maximum test case steps 11-23
 - Model coverage objectives 11-20
 - Test conditions 11-21
 - Test objectives 11-22
 - Test suite optimization 11-24

E

- Embedded MATLAB library functions
 - limitations 2-8
- Embedded MATLAB subset support
 - summary 13-2

M

- model compatibility
 - checking 2-10

P

- parameter configurations
 - example 4-7
 - introduction 4-2
 - syntax 4-4

- template 4-3
- Proof Assumption block 10-2
- Proof Objective block 10-8
- property proofs
 - example 7-4
 - introduction 7-2
 - Stateflow actions 7-2
 - workflow 7-3

S

- Simulink Design Verifier
 - model parameters 11-48
 - running demo 1-6
 - workflow 1-17
- Simulink Design Verifier data files
 - anatomy 8-23
 - simulation 8-28
- Simulink Design Verifier options
 - saving 5-16
 - viewing 5-2
- Simulink Design Verifier report
 - table of contents 8-8
- Simulink Design Verifier reports
 - approximations 8-22
 - block replacements summary 8-14
 - summary 8-9
 - test cases/counterexamples 8-19
 - test/proof objectives 8-14
 - title 8-8
- sldvblockreplacement function 9-2
- sldvcompat function 9-3

- sldvextract function 9-5
- sldvgencov function 9-6
- sldvharnessmerge function 9-7
- sldvoptions function 9-8
- sldvrun function 9-15
- sldvruntest function 9-17
- system requirements 1-3

T

- test case generation
 - example 6-4
 - introduction 6-2
 - Stateflow actions 6-2
 - workflow 6-3
- Test Condition block 10-13
- test harness models
 - anatomy 8-2
 - simulation 8-6
- Test Objective block 10-19

U

- unsupported features
 - Embedded MATLAB subset 2-7
 - Simulink 2-3
 - Stateflow 2-5

V

- Verification Subsystem block 10-24