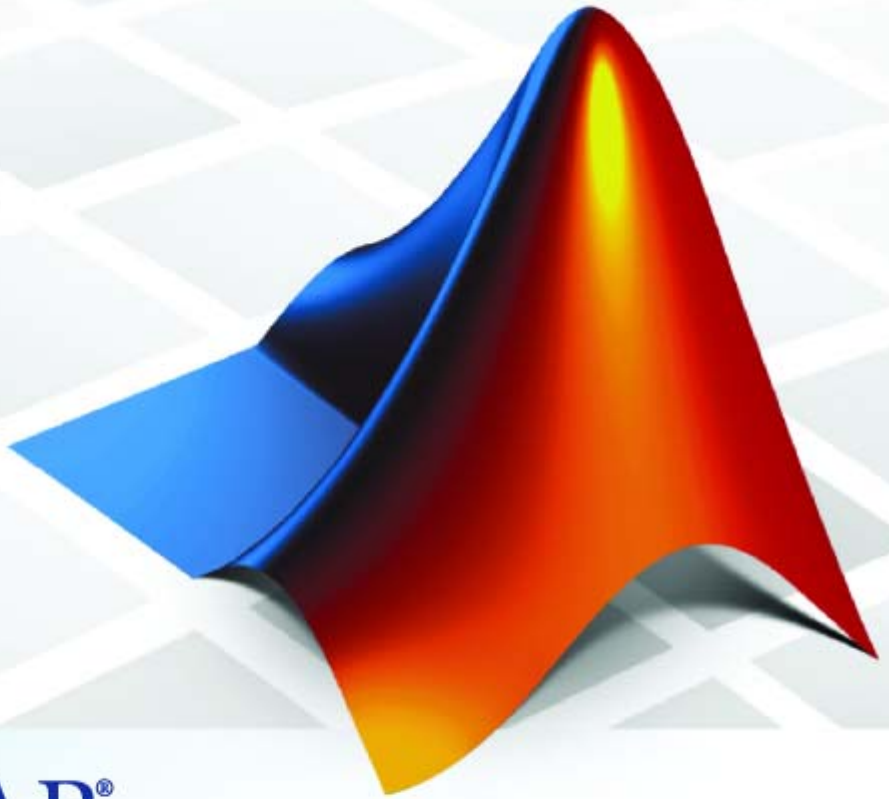


# Simulink® Design Verifier 1

## 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 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, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, SimBiology, SimHydraulics, and xPC TargetBox are registered trademarks and SimEvents is a trademark of The MathWorks, Inc.

Prover, Prover Technology, Prover Plug-In and the Prover logo are trademarks or registered trademarks of Prover Technology AB in Sweden, the United States and in other countries.

Other product or brand names are 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](http://www.mathworks.com/patents) for more information.

## Acknowledgment

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





## Acknowledgment

## Getting Started

# 1

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

## Ensuring Compatibility with Simulink Design Verifier

### 2

<b>Unsupported Simulink Features</b> .....	<b>2-2</b>
List of Unsupported Simulink Features .....	<b>2-2</b>
Limitations of Simulink Block Support .....	<b>2-2</b>
<b>Unsupported Stateflow Features</b> .....	<b>2-3</b>
<b>Checking Model Compatibility</b> .....	<b>2-5</b>

## Working with Block Replacements

### 3

<b>About Block Replacements</b> .....	<b>3-2</b>
<b>Built-In Block Replacements</b> .....	<b>3-3</b>
<b>Template for Block Replacement Rules</b> .....	<b>3-6</b>
<b>Creating Custom Block Replacements</b> .....	<b>3-7</b>
Constructing Replacement Blocks .....	<b>3-7</b>
Writing Block Replacement Rules .....	<b>3-10</b>
<b>Executing Block Replacements</b> .....	<b>3-14</b>
Configuring Block Replacements .....	<b>3-14</b>
Replacing Blocks in a Model .....	<b>3-15</b>

## Specifying Parameter Configurations

### 4

<b>About Parameter Configurations</b> .....	<b>4-2</b>
---	------------

<b>Template for Parameter Configurations</b> .....	<b>4-3</b>
<b>Defining Parameter Configurations</b> .....	<b>4-4</b>
<b>Parameter Configuration Example</b> .....	<b>4-7</b>
Constructing the Example Model .....	<b>4-7</b>
Parameterizing the Constant Block .....	<b>4-9</b>
Specifying a Parameter Configuration .....	<b>4-11</b>
Analyzing the Example Model .....	<b>4-12</b>
Simulating the Test Cases .....	<b>4-14</b>

## Configuring Simulink Design Verifier

### 5

<b>Viewing Simulink Design Verifier Options</b> .....	<b>5-2</b>
<b>Configuring Simulink Design Verifier Options</b> .....	<b>5-5</b>
Design Verifier Pane .....	<b>5-5</b>
Block Replacements Pane .....	<b>5-6</b>
Parameters Pane .....	<b>5-8</b>
Test Generation Pane .....	<b>5-9</b>
Property Proving Pane .....	<b>5-10</b>
Results Pane .....	<b>5-12</b>
Report Pane .....	<b>5-14</b>
<b>Saving Simulink Design Verifier Options</b> .....	<b>5-15</b>

## Generating Test Cases

### 6

<b>About Test Case Generation</b> .....	<b>6-2</b>
<b>Basic Workflow for Generating Test Cases</b> .....	<b>6-3</b>

<b>Generating Test Cases Example</b> .....	<b>6-4</b>
Constructing the Example Model .....	<b>6-4</b>
Checking Compatibility of the Example Model .....	<b>6-6</b>
Configuring Test Generation Options .....	<b>6-9</b>
Analyzing the Example Model .....	<b>6-12</b>
Customizing Test Generation .....	<b>6-20</b>
Reanalyzing the Example Model .....	<b>6-24</b>

## Proving Properties of a Model

# 7

<b>About Property Proofs</b> .....	<b>7-2</b>
<b>Basic Workflow for Proving Model Properties</b> .....	<b>7-3</b>
<b>Proving Model Properties Example</b> .....	<b>7-4</b>
Constructing the Example Model .....	<b>7-5</b>
Checking Compatibility of the Example Model .....	<b>7-6</b>
Instrumenting the Example Model .....	<b>7-9</b>
Configuring Property Proving Options .....	<b>7-12</b>
Analyzing the Example Model .....	<b>7-14</b>
Customizing the Example Proof .....	<b>7-22</b>
Reanalyzing the Example Model .....	<b>7-24</b>

## Reviewing the Results

# 8

<b>Exploring Test Harness Models</b> .....	<b>8-2</b>
Anatomy of a Test Harness .....	<b>8-2</b>
Simulating the Test Harness .....	<b>8-5</b>
<b>Understanding Simulink Design Verifier Reports</b> .....	<b>8-6</b>
Front Matter .....	<b>8-6</b>
Summary Chapter .....	<b>8-7</b>
Block Replacements Summary Chapter .....	<b>8-12</b>
Test/Proof Objectives Chapter .....	<b>8-12</b>



Test Cases / Counterexamples Chapter .....	8-17
Approximations Chapter .....	8-20
<b>Examining Simulink Design Verifier Data Files .....</b>	<b>8-21</b>
Anatomy of the sldvData Structure .....	8-21
Simulating Models with Simulink Design Verifier Data Files .....	8-25

## Analyzing Large Models and Improving Performance

# A

<b>How Simulink Design Verifier Works .....</b>	<b>A-2</b>
<b>Sources of Model Complexity in Simulink Design     Verifier .....</b>	<b>A-5</b>
<b>Handling Models with Large Numbers of Inputs .....</b>	<b>A-6</b>
<b>Reducing Complexity from Floating-Point Operations     and Nonlinear Arithmetic .....</b>	<b>A-7</b>
<b>Partitioning Inputs and Generating Tests     Incrementally .....</b>	<b>A-9</b>
<b>Handling Models with Large State Spaces .....</b>	<b>A-11</b>
<b>Handling Problems with Counters and Timers .....</b>	<b>A-12</b>
<b>Special Strategies for Proving Properties of Larger     Models .....</b>	<b>A-13</b>

**Functions — Alphabetical List**

**9**

**Blocks — Alphabetical List**

**10**

**Configuration Parameters**

**11**

**Simulink Block Support**

**12**

**Glossary**

**Examples**

**B**

<b>Working with Block Replacements .....</b>	<b>B-2</b>
<b>Specifying Parameter Configurations .....</b>	<b>B-2</b>
<b>Generating Test Cases .....</b>	<b>B-2</b>
<b>Proving Properties of a Model .....</b>	<b>B-2</b>





# Getting Started

---

What Is Simulink Design Verifier? (p. 1-2)	Overview of the product
Before You Begin (p. 1-3)	Other products you need or might want to use with Simulink Design Verifier
Starting Simulink Design Verifier (p. 1-4)	Accessing the Simulink Design Verifier library
Running a Demo Model (p. 1-6)	Analyzing a simple demo model with Simulink Design Verifier
Basic Workflow for Using Simulink Design Verifier (p. 1-17)	Overview of the basic workflow
Learning More (p. 1-18)	Where to find more information

## **What Is Simulink Design Verifier?**

Simulink Design Verifier is a product that extends Simulink by performing exhaustive formal analyses of your models to confirm that they behave correctly.

Simulink Design Verifier 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

### What You Need to Know

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

To learn more about these topics, see the following:

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

### Required Products

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

- MATLAB®
- Simulink
- Simulink Verification and Validation

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

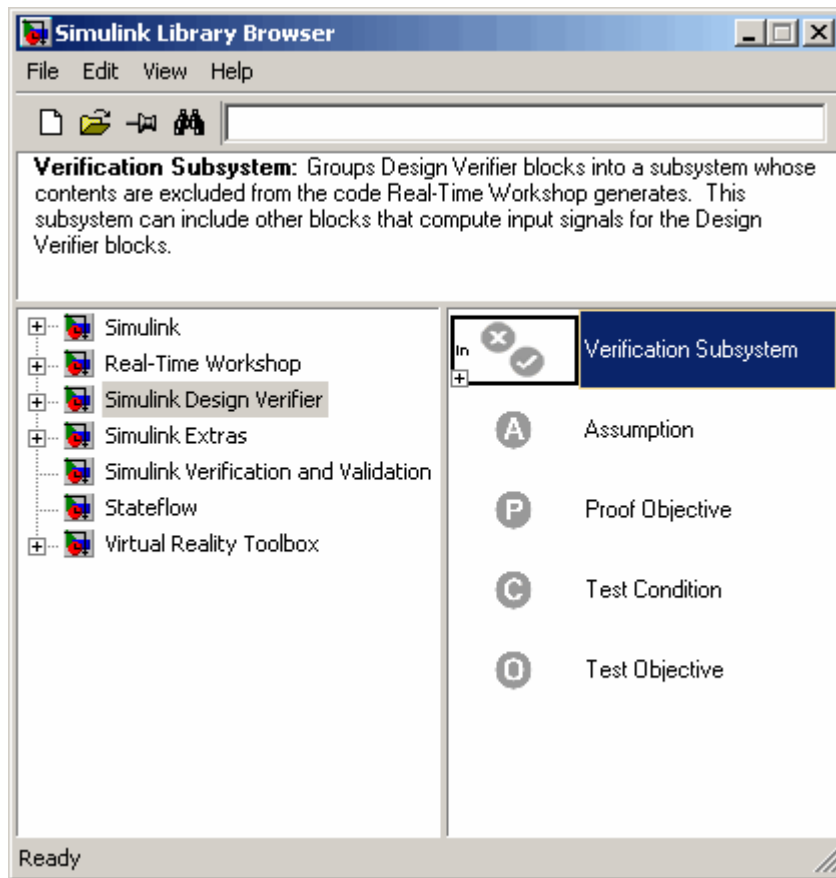
- Stateflow

## Starting Simulink Design Verifier

Simulink Design Verifier is part of your MATLAB installation.

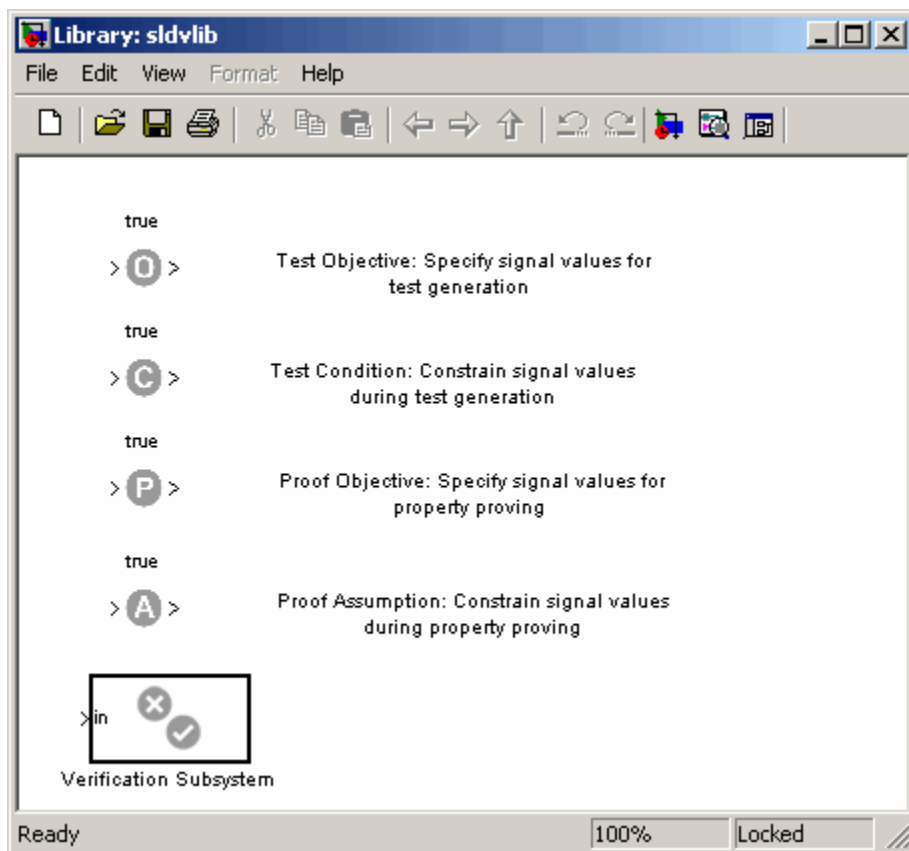
To open the Simulink Design Verifier block library:

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



- On any other platform, type `sldvlib` at the MATLAB prompt to display the Simulink Design Verifier library.





## Running a Demo Model

- “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

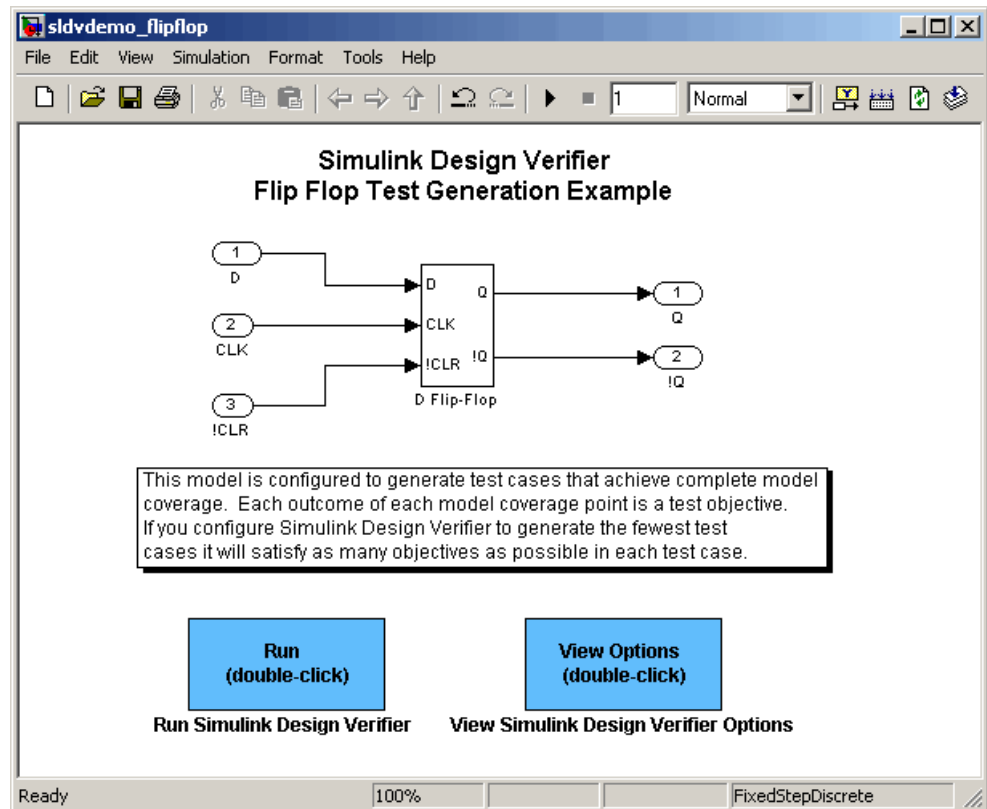
### About This Demo

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

### Opening the Model

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

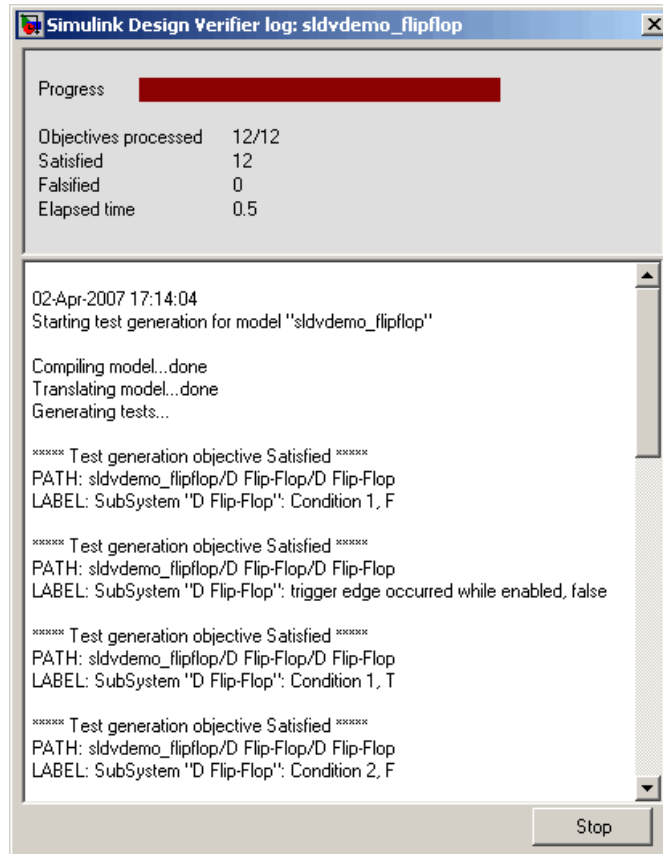
Simulink displays the Flip Flop Test Generation Example model.



## Generating Test Cases

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

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



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

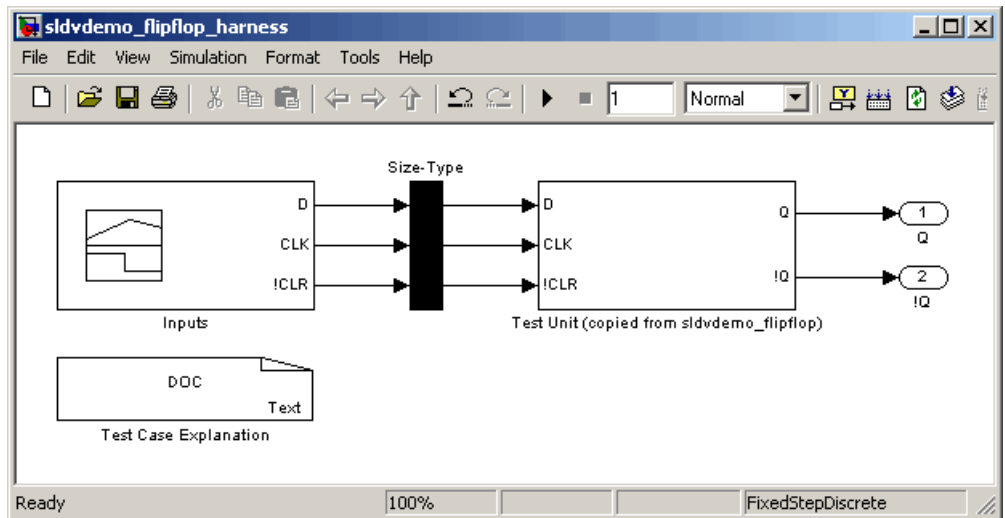
When Simulink Design Verifier completes its analysis, it displays the following items:

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

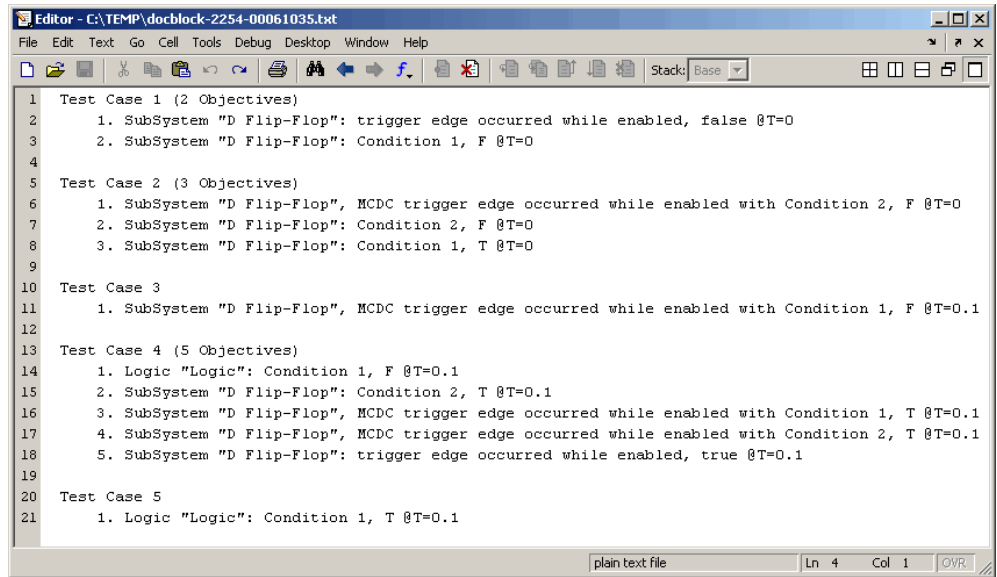
The sections that follow describe each of these items.

## Exploring the Test Harness

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



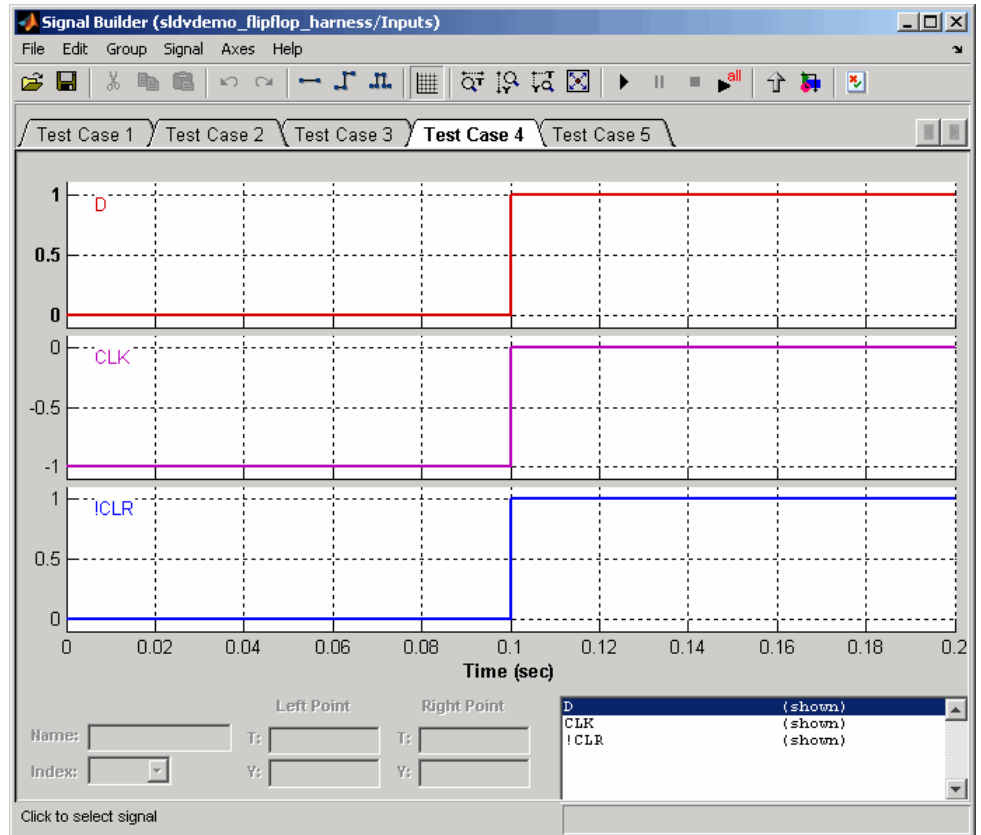
- 1 The block labeled Test Case Explanation is a DocBlock that documents the test cases Simulink Design Verifier generated. Double-click the Test Case Explanation block to view a description of each test case in terms of the objectives that it satisfies.



```
Editor - C:\TEMP\docblock-2254-00061035.txt
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1 Test Case 1 (2 Objectives)
2   1. SubSystem "D Flip-Flop": trigger edge occurred while enabled, false @T=0
3   2. SubSystem "D Flip-Flop": Condition 1, F @T=0
4
5 Test Case 2 (3 Objectives)
6   1. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with Condition 2, F @T=0
7   2. SubSystem "D Flip-Flop": Condition 2, F @T=0
8   3. SubSystem "D Flip-Flop": Condition 1, T @T=0
9
10 Test Case 3
11   1. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with Condition 1, F @T=0.1
12
13 Test Case 4 (5 Objectives)
14   1. Logic "Logic": Condition 1, F @T=0.1
15   2. SubSystem "D Flip-Flop": Condition 2, T @T=0.1
16   3. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with Condition 1, T @T=0.1
17   4. SubSystem "D Flip-Flop", MCDC trigger edge occurred while enabled with Condition 2, T @T=0.1
18   5. SubSystem "D Flip-Flop": trigger edge occurred while enabled, true @T=0.1
19
20 Test Case 5
21   1. Logic "Logic": Condition 1, T @T=0.1
plain text file Ln 4 Col 1 OVR
```

- 2** The block labeled Test Unit is a Subsystem block that contains a copy of the original model Simulink Design Verifier analyzed. Double-click the Test Unit block to view its contents and confirm that it is a copy of the Flip Flop Test Generation Example model.
- 3** The block labeled Inputs is a Signal Builder block that contains the test case signals Simulink Design Verifier 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 4.

The Signal Builder dialog box displays the signal values for Test Case 4.




In Test Case 4 at 0.1 seconds,

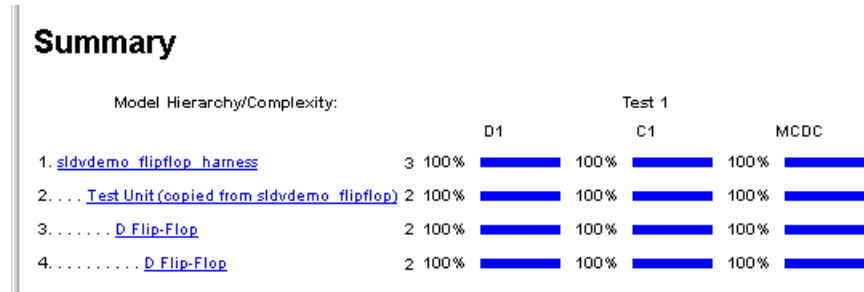
- The D signal transitions from 0 to 1.
- The CLK signal transitions from -1 to 0.
- The !CLR signal transitions from 0 to 1.

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

- 5 To confirm that Simulink Design Verifier achieved complete model coverage, simulate the test harness using all the test cases. In the Signal

Builder dialog box, click the **Run all** button .

Simulink simulates the test harness using all the test cases, collects model coverage information, and displays a coverage report with the following summary:



The coverage report indicates Simulink Design Verifier generated test cases that achieve complete coverage for the Flip Flop Test Generation Example model.

## Interpreting the Simulink Design Verifier Report

Simulink Design Verifier creates an HTML report that summarizes its analysis results. 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/Proof Objectives](#)
  - [Status](#)
  - [sldvdemo\\_flipflop](#)
  - [D Flip-Flop](#)
- [3. Test Cases / Counterexamples](#)
  - [Test Case 1](#)
  - [Test Case 2](#)
  - [Test Case 3](#)
  - [Test Case 4](#)
  - [Test Case 5](#)
- [4. Approximations](#)

1 In the **Table of Contents**, click Summary.

The report displays its Summary chapter, as shown here.



## Chapter 1. Summary

### Input Model

File: c:\matlab\toolbox\sldv\sldvdemos\sldvdemo\_flipflop.mdl  
 Version: 1.12  
 Time Stamp: Sat Mar 3 02:53:36 2007  
 Author:

### Analysis Information

Design Verifier Version: 1.0  
 Total Analysis Time: 0.46 secs  
 Status: Completed normally  
[Approximations:](#) 1  
[Objectives Satisfied:](#) 12  
 Objectives Proven Unsatisfiable: 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/Proof Objectives chapter:

## Chapter 2. Test/Proof Objectives

### Table of Contents

[Status](#)

[sldvdemo\\_flipflop](#)

[D Flip-Flop](#)

### Status

**Table 2.1. Objectives Satisfied**

#:	Type	Model Item	Description
<a href="#">1</a>	Condition	<a href="#">Logic</a>	Logic "Logic", Condition 1, T
<a href="#">2</a>	Condition	<a href="#">Logic</a>	Logic "Logic", Condition 1, F
<a href="#">3</a>	Condition	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", Condition 1, T
<a href="#">4</a>	Condition	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", Condition 1, F
<a href="#">5</a>	Condition	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", Condition 2, T
<a href="#">6</a>	Condition	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", Condition 2, F
<a href="#">7</a>	Decision	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", trigger edge occurred while enabled, F
<a href="#">8</a>	Decision	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", trigger edge occurred while enabled, T
<a href="#">9</a>	Mcadc	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", trigger edge occurred while enabled with Condition 1, T
<a href="#">10</a>	Mcadc	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", trigger edge occurred while enabled with Condition 1, F
<a href="#">11</a>	Mcadc	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", trigger edge occurred while enabled with Condition 2, T
<a href="#">12</a>	Mcadc	<a href="#">D Flip-Flop</a>	SubSystem "D Flip-Flop", trigger edge occurred while enabled with Condition 2, F

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

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

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

## sldvdemo\_flipflop

Objectives of: D Flip-Flop

#:	Status	Test Cases	Description
3	Satisfied	<a href="#">TC 2</a>	Condition 1, T
4	Satisfied	<a href="#">TC 1</a>	Condition 1, F
5	Satisfied	<a href="#">TC 4</a>	Condition 2, T
6	Satisfied	<a href="#">TC 2</a>	Condition 2, F
7	Satisfied	<a href="#">TC 1</a>	trigger edge occurred while enabled, F
8	Satisfied	<a href="#">TC 4</a>	trigger edge occurred while enabled, T
9	Satisfied	<a href="#">TC 4</a>	trigger edge occurred while enabled with Condition 1, T
10	Satisfied	<a href="#">TC 3</a>	trigger edge occurred while enabled with Condition 1, F
11	Satisfied	<a href="#">TC 4</a>	trigger edge occurred while enabled with Condition 2, T
12	Satisfied	<a href="#">TC 2</a>	trigger edge occurred while enabled with Condition 2, 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 4](#).

The report displays its Test Case 4 section under the Test Cases / Counterexamples chapter, as shown here.

## Test Case 4

### Summary

Length: 0.2 Seconds (3 sample periods)

Objective Count: 5

### Objectives Reached At:

Step	Time	Objectives
2	0.1	2 5 8 9 11

### Generated Input Data.

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

This section provides details about a test case that Simulink Design Verifier generated. For example, Test Case 4 satisfies five model coverage objectives. In this test case, the following signal values achieve objectives 2, 5, 8, 9, and 11:

- The D signal transitions from 0 to 1 at 0.1 seconds.
- The CLK signal transitions from -1 to 0 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 4, and the Test Case Explanation block lists five objectives that Test Case 4 achieves (see “Exploring the Test Harness” on page 1-9).

## Basic Workflow for Using Simulink Design Verifier

The Simulink Design Verifier User's Guide 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 Simulink Design Verifier"
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"
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


- “Next Step” on page 1-18
- “Product Help” on page 1-18
- “The MathWorks Online” on page 1-18

### Next Step

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

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 MATLAB, 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 Simulink Design Verifier

---

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

Unsupported Simulink Features  
(p. 2-2)

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

Unsupported Stateflow Features  
(p. 2-3)

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

Checking Model Compatibility  
(p. 2-5)

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

## Unsupported Simulink Features

- “List of Unsupported Simulink Features” on page 2-2
- “Limitations of Simulink Block Support” on page 2-2

### List of Unsupported Simulink Features

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

Feature Not Supported	Remarks
Variable-step solvers	Simulink Design Verifier supports only fixed-step solvers (see “Choosing a Fixed-Step Solver” in <i>Using Simulink</i> ).
Complex signals	Simulink Design Verifier supports only real signals (for contrast, see “Complex Signals” in <i>Using Simulink</i> ).
Fixed-point data types	Simulink Design Verifier supports only the built-in data types that Simulink recognizes (see “Data Types Supported by Simulink” in <i>Using Simulink</i> ).

### Limitations of Simulink Block Support

Simulink Design Verifier provides various levels of support for Simulink blocks. That is, Simulink Design Verifier 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 Simulink Design Verifier. Similarly, specify only the block parameters that Simulink Design Verifier recognizes for blocks that it partially supports. See Chapter 12, “Simulink Block Support” for a list of Simulink blocks and details regarding whether Simulink Design Verifier provides support.

## Unsupported Stateflow Features

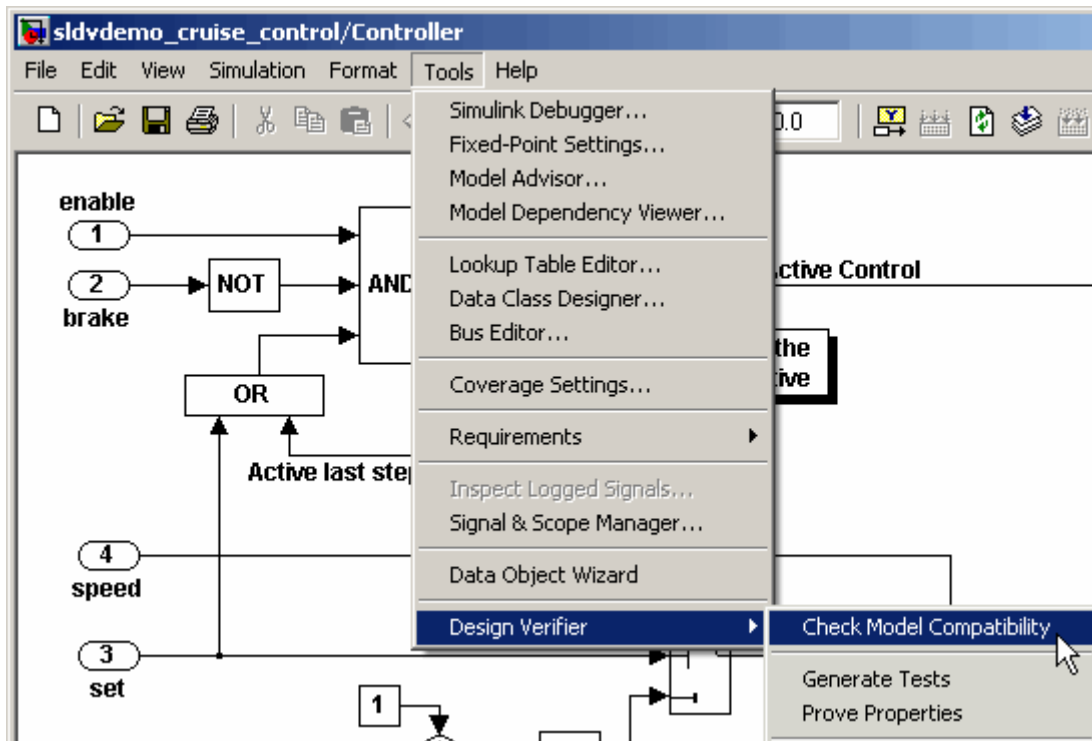
Simulink Design Verifier does not support the following Stateflow features. Avoid using these unsupported Stateflow features in models that you analyze with Simulink Design Verifier.

Feature Not Supported	Remarks
m1 namespace operator, m1 function, m1 expressions	Simulink Design Verifier does not support calls to MATLAB functions or access to MATLAB workspace variables, which Stateflow allows (see “Using MATLAB Functions and Data in Actions” in the <i>Stateflow and Stateflow Coder User’s Guide</i> ).
Embedded MATLAB functions	Simulink Design Verifier does not support Embedded MATLAB functions, which Stateflow allows (see “Using Embedded MATLAB Functions” in the <i>Stateflow and Stateflow Coder User’s Guide</i> ).
C math functions	Simulink Design Verifier does not support calls to C math functions, which Stateflow allows (see “Calling C Functions in Actions” in the <i>Stateflow and Stateflow Coder User’s Guide</i> ).
Recursion	Simulink Design Verifier does not support recursive functions, which Stateflow allows you to implement using graphical functions (see “Using Functions to Extend Actions” in the <i>Stateflow and Stateflow Coder User’s Guide</i> ). Also, Simulink Design Verifier does not support recursion that Stateflow allows you to implement using a combination of event broadcasts and function calls.
Fixed-point data types	Simulink Design Verifier does not support fixed-point data types, which Stateflow allows (see “Using Fixed-Point Data in Stateflow” in the <i>Stateflow and Stateflow Coder User’s Guide</i> ).

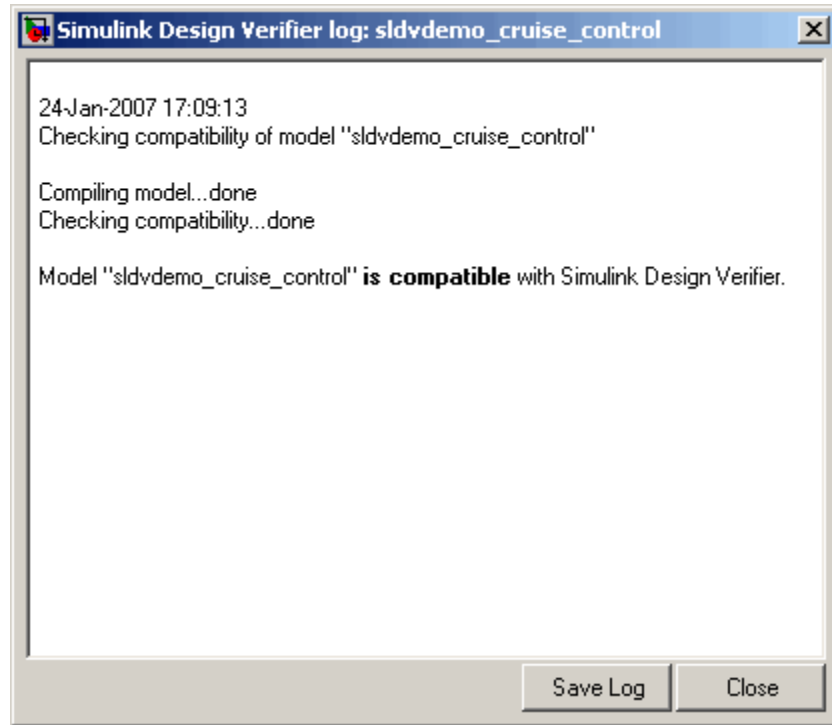
<b>Feature Not Supported</b>	<b>Remarks</b>
Custom C or C++ code	Simulink Design Verifier does not support custom C or C++ code, which Stateflow allows (see “Integrating Custom Code with Stateflow Targets” in the <i>Stateflow and Stateflow Coder User’s Guide</i> ).
Machine-parented data and events	Simulink Design Verifier does not support machine-parented data and events (i.e., defined at the level of the Stateflow machine in the Stateflow hierarchy), which Stateflow allows (see “Defining Events and Data” in the <i>Stateflow and Stateflow Coder User’s Guide</i> ).

## Checking Model Compatibility

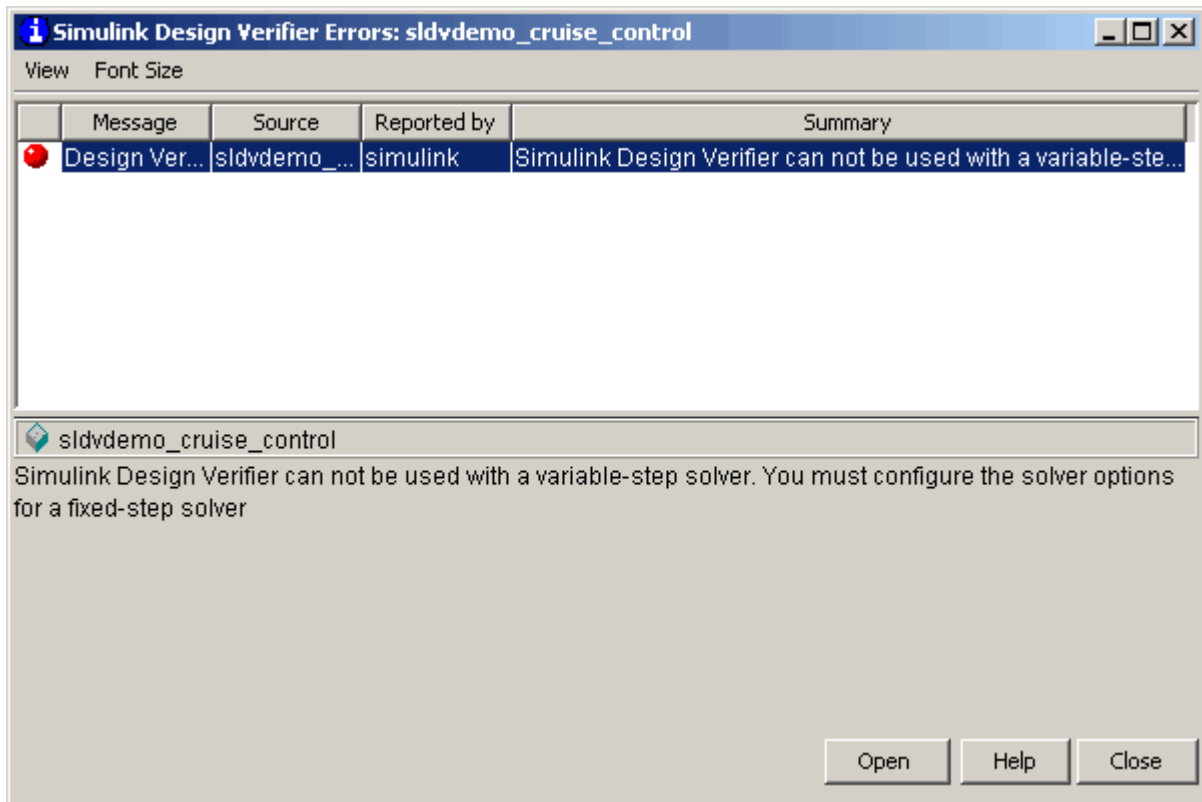
Simulink Design Verifier provides a mechanism that checks whether your model is compatible for analysis. To check the compatibility of your model, from the **Tools** menu of your Simulink model, select **Design Verifier > Check Model Compatibility**.



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



Otherwise, Simulink Design Verifier 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, Simulink Design Verifier 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** Simulink Design Verifier 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 Chapter 9, “Functions — Alphabetical List” for more information.





# Working with Block Replacements

---

Simulink Design Verifier 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. 3-14)	Describes how to execute block replacements.

## About Block Replacements

Simulink Design Verifier 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, Simulink Design Verifier 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 Simulink Design Verifier.

Simulink Design Verifier 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

Simulink Design Verifier 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 Simulink Design Verifier. 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 <b>Vector of input values</b> 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 <b>Number of inputs</b> 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 <b>Number of inputs</b> 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 <b>Number of inputs</b> 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 <b>Number of inputs</b> 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 <b>Index Option</b> 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 <b>Input port size</b> and <b>Index mode</b> 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 <b>Index Option</b> 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 <b>Input port size, Output size, and Index mode</b> parameters.

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

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

---

**Note** Simulink Design Verifier 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, Simulink Design Verifier 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

This section demonstrates how to create custom block replacements in Simulink Design Verifier. The process consists of two tasks, constructing a replacement block and writing a block replacement rule. The following sections guide you through the process of creating a custom block replacement:

Constructing Replacement Blocks (p. 3-7)	Describes how to construct replacement blocks.
Writing Block Replacement Rules (p. 3-10)	Describes how to write block replacement rules.

### Constructing Replacement Blocks

Simulink Design Verifier 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 whose block names specify default values (e.g., In1 and Out1).

---

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

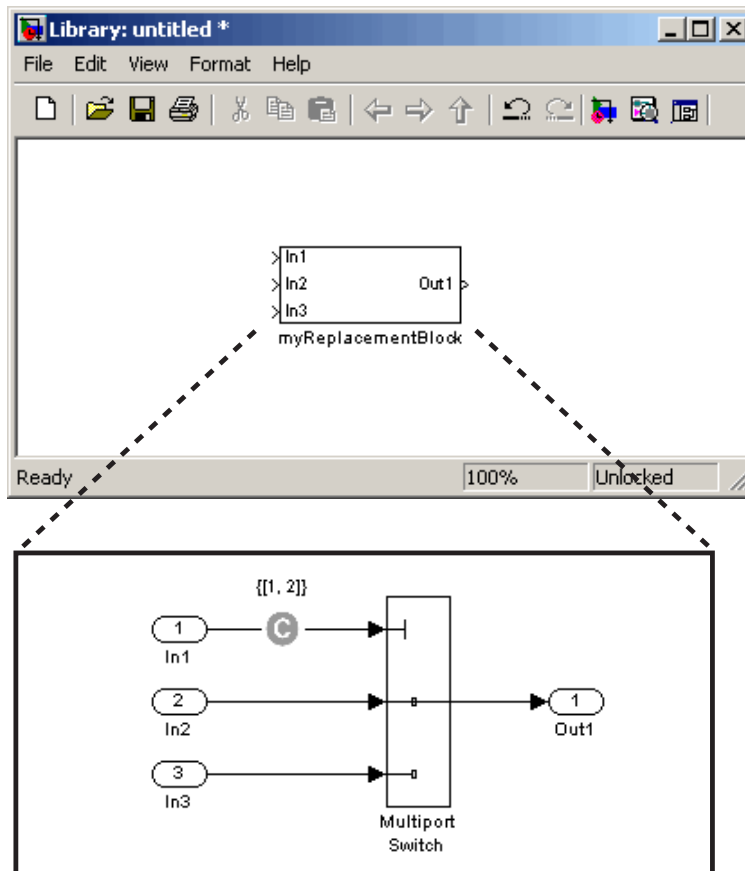
---

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  $\{[1, 2]\}$

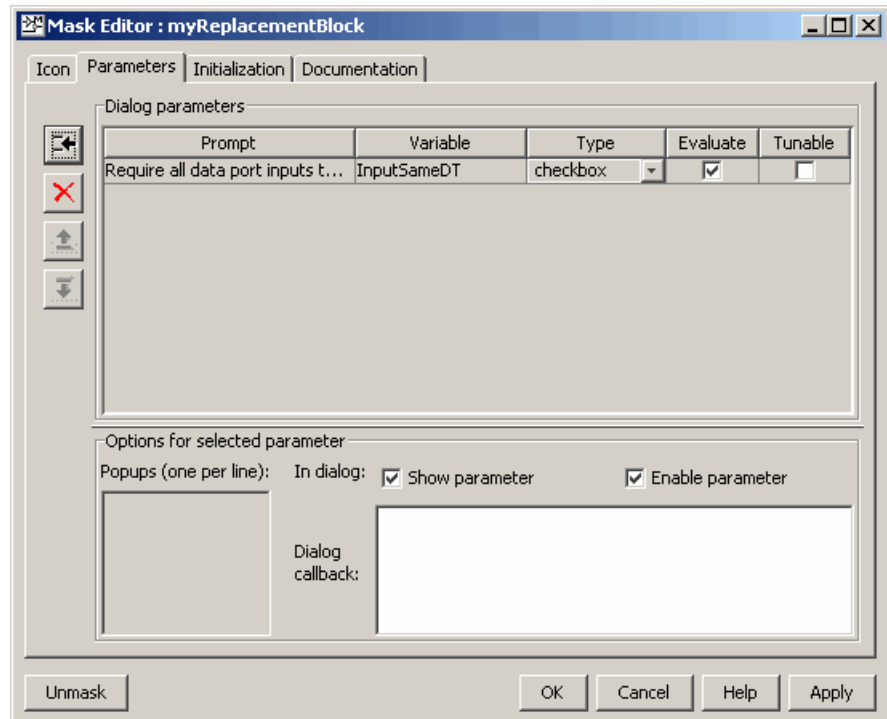


- 3** Create a mask for your subsystem (see “Masking a Subsystem” in *Using Simulink*).

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(gcb, '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

Simulink Design Verifier 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, Output, 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
%Simulink Design Verifier
%
% 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 Simulink Design Verifier uses when replacing blocks by specifying a value for the `rule.replacementMode` object. Valid values include:
- **Normal** — When using this rule, Simulink Design Verifier replaces blocks with a copy of the subsystem specified by the `rule.replacementPath` object.
  - **ConfigurableSubSystem** — When using this rule, Simulink Design Verifier 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 Simulink Design Verifier replaces blocks in your model.

The following example instructs Simulink Design Verifier 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 Simulink Design
% Verifier 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 3-14).

## Executing Block Replacements

You can execute block replacements from the MATLAB command line or an M-file program, or from the Simulink GUI. The following sections describe how to configure block replacement options, execute block replacements, and interpret the output that block replacements generate.

Configuring Block Replacements (p. 3-14)	Describes how to configure block replacement options.
Replacing Blocks in a Model (p. 3-15)	Describes how to execute block replacements in your model and interpret the output messages.

### 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 checking 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. You can specify multiple rules as a list delimited by spaces, commas, or carriage returns. Simulink

Design Verifier executes the rules in the order that you list them. For example, to execute a subset of the factory default rules (see “Built-In Block Replacements” on page 3-3) 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** Simulink Design Verifier replaces a block in your model only once. If multiple rules apply to the same block, Simulink Design Verifier 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 Simulink Design Verifier 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, “Functions — Alphabetical List” for more information.

## Replacing Blocks in a Model

After enabling the **Apply block replacements** option (see “Configuring Block Replacements” on page 3-14), 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.

When performing block replacements, Simulink Design Verifier copies your model and replaces blocks in the copy, leaving your original model unaltered. Upon completing its analysis, Simulink Design Verifier generates a report

that displays information about the block replacements it executed (see “Understanding Simulink Design Verifier Reports” on page 8-6).

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, Simulink Design Verifier 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 3-14). 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, Simulink Design Verifier displays information about the block replacements that it performed. For example, the following message indicates that Simulink Design Verifier 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

---

Simulink Design Verifier 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

Simulink Design Verifier 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 Simulink Design Verifier 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, Simulink Design Verifier 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

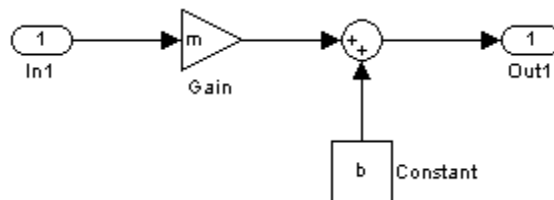
This section describes how to define parameter configurations and outlines the required syntax for their definition.

### 1 Define parameter configurations in an M-file function.

Simulink Design Verifier 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, Simulink Design Verifier 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, Simulink Design Verifier 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

To understand how to specify parameter configurations for use with Simulink Design Verifier, you build a simple Simulink model and generate test cases that achieve complete decision coverage. The following sections guide you through the process of completing this example:

Constructing the Example Model  
(p. 4-7)

Guides you through Task 1 of the example, in which you construct the example model.

Parameterizing the Constant Block  
(p. 4-9)

Guides you through Task 2 of the example, in which you specify a variable as the value of a Constant block parameter.

Specifying a Parameter Configuration  
(p. 4-11)

Guides you through Task 3 of the example, in which you constrain the value of the variable that the Constant block specifies.

Analyzing the Example Model  
(p. 4-12)

Guides you through Task 4 of the example, in which you generate test cases for your model and interpret the results.

Simulating the Test Cases (p. 4-14)

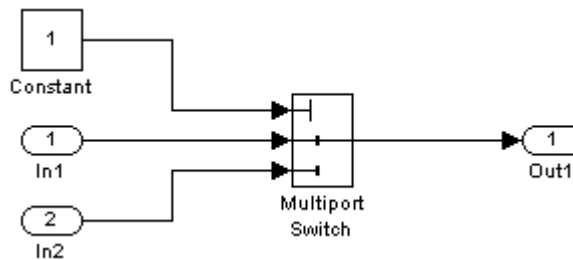
Guides you through Task 5 of the example, in which you simulate the test cases and measure the resulting decision coverage.

### Constructing the Example Model

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

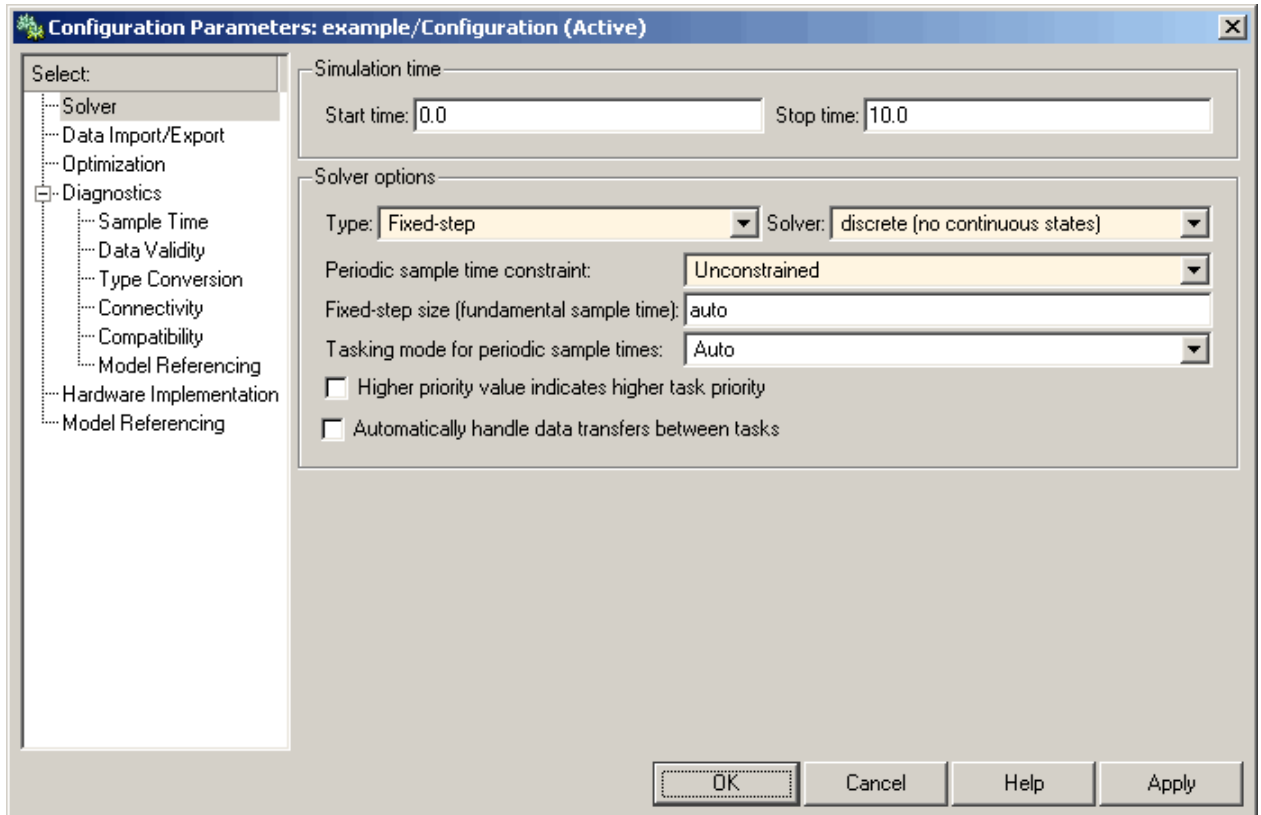
- 1 Create an empty Simulink model (see “Creating an Empty Model” in *Using Simulink* for help with this step).

- 2 Copy the following blocks into your empty model window (see “Adding Blocks” 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**.  
Simulink displays the Configuration Parameters dialog box.
- 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 4-9.

## Parameterizing the Constant Block

This section presents Task 2 of the process that describes how to specify parameter configurations in Simulink Design Verifier. In this task, you

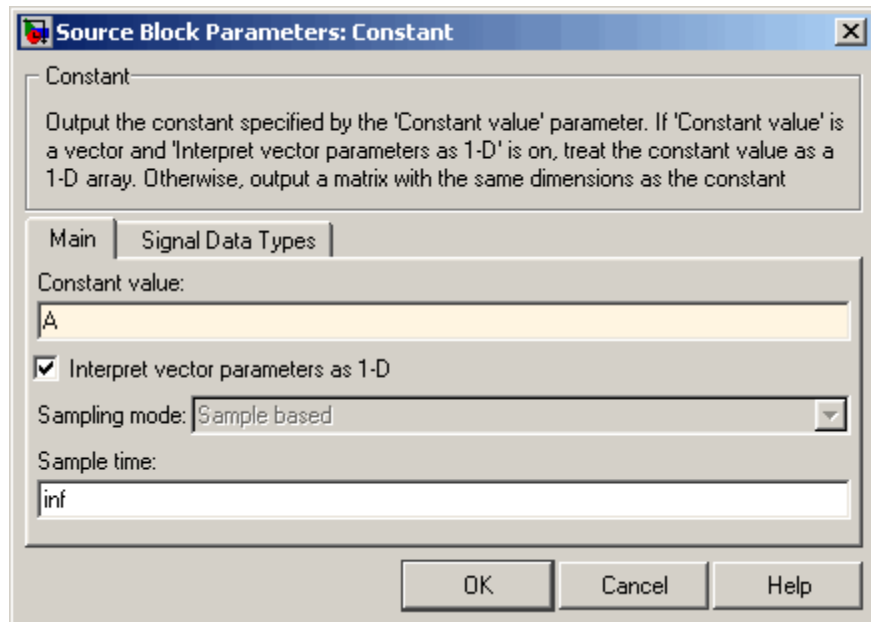
parameterize the Constant block in the model that you created in the previous task (see “Constructing the Example Model” on page 4-7). In particular, you specify a variable as the value of the Constant block’s **Constant value** parameter. To complete this task, perform the following steps:

- 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. Simulink 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 4-11.

## Specifying a Parameter Configuration

This section presents Task 3 of the process that describes how to specify parameter configurations in Simulink Design Verifier. 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 4-9). To complete this task, perform the following steps:

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

Simulink Design Verifier 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.

Simulink Design Verifier 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 4-12.

### Analyzing the Example Model

This section presents Task 4 of the process that describes how to specify parameter configurations in Simulink Design Verifier. 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 4-11). Simulink Design Verifier 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**.

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

- Simulink Design Verifier report — Simulink Design Verifier displays an HTML report named `param_example_report.html`.
- Test harness — Simulink Design Verifier 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 Simulink Design Verifier 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 / Counterexamples** 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 4-14.

### Simulating the Test Cases

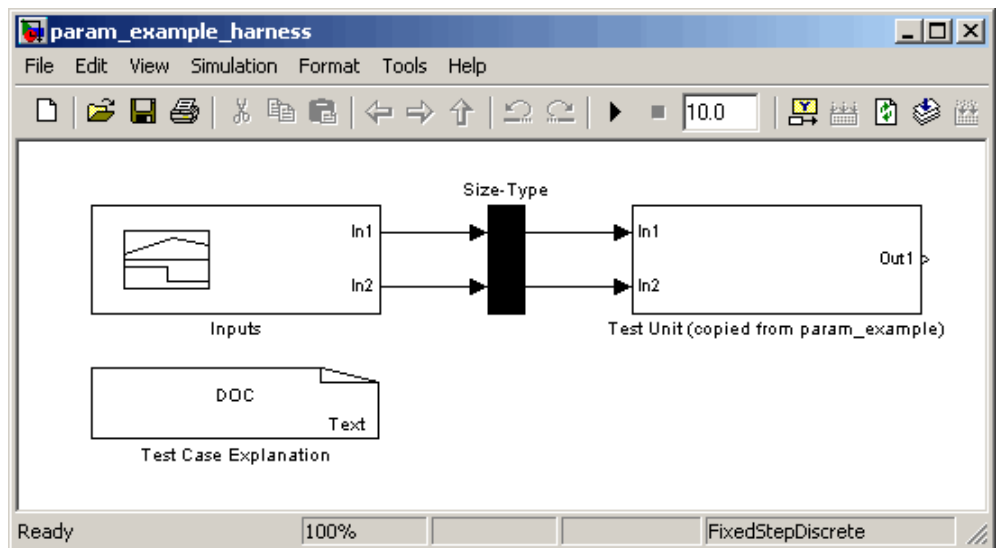
This section presents Task 5 of the process that describes how to specify parameter configurations in Simulink Design Verifier. In this final task,



you simulate the test cases that Simulink Design Verifier generated in the previous task (see “Simulating the Test Cases” on page 4-14). Also, you review the coverage report that results from the simulation. To complete this task, perform the following steps:

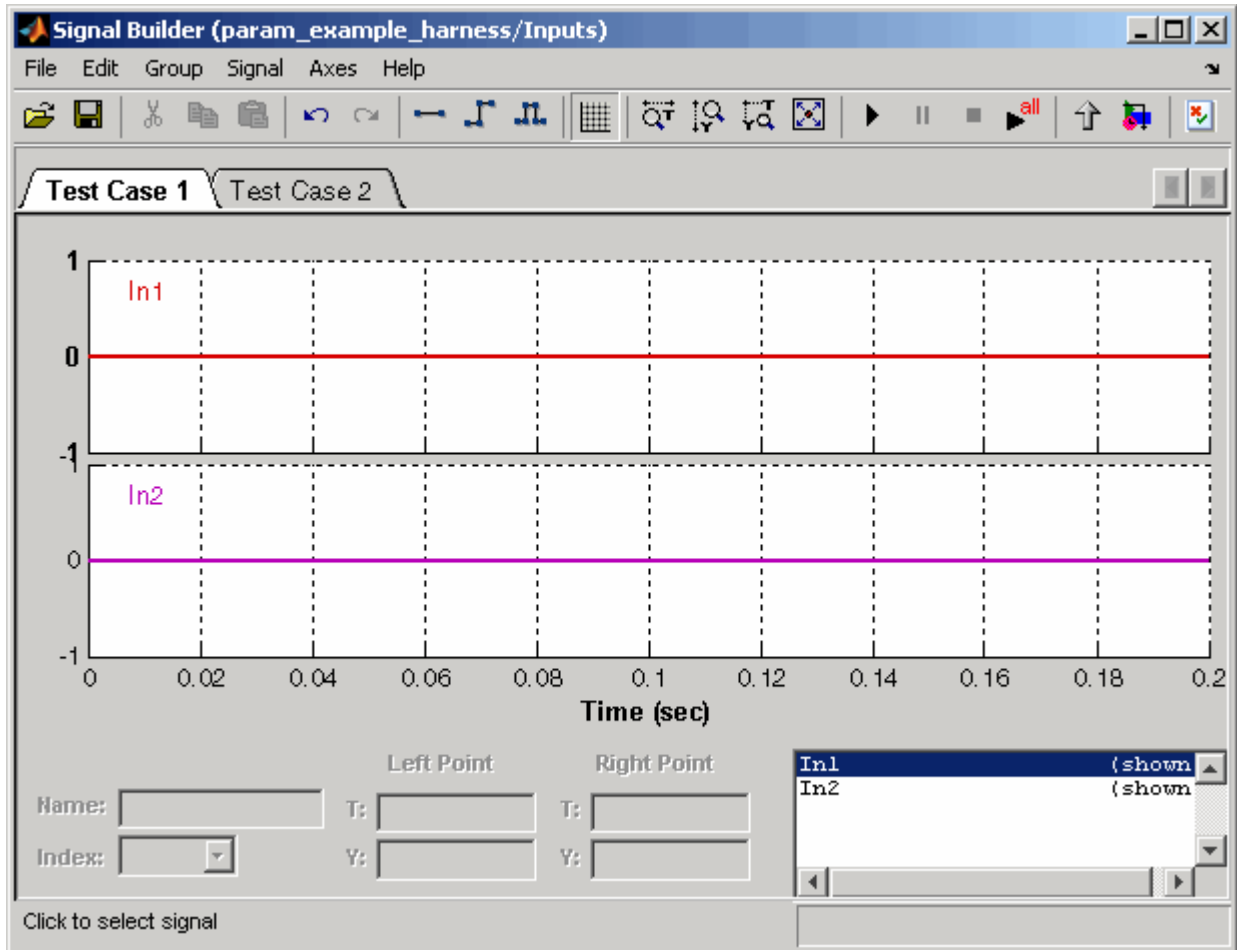
- 1 Open the test harness model named `param_example_harness.mdl` (if it is not already open).


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 .

Simulink 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. <a href="#">param_example_harness</a>	2	100%	
2. . . . <a href="#">Test Unit (copied from param_example)</a>	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

---

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 the options that control Simulink Design Verifier.

Configuring Simulink Design Verifier Options (p. 5-5)

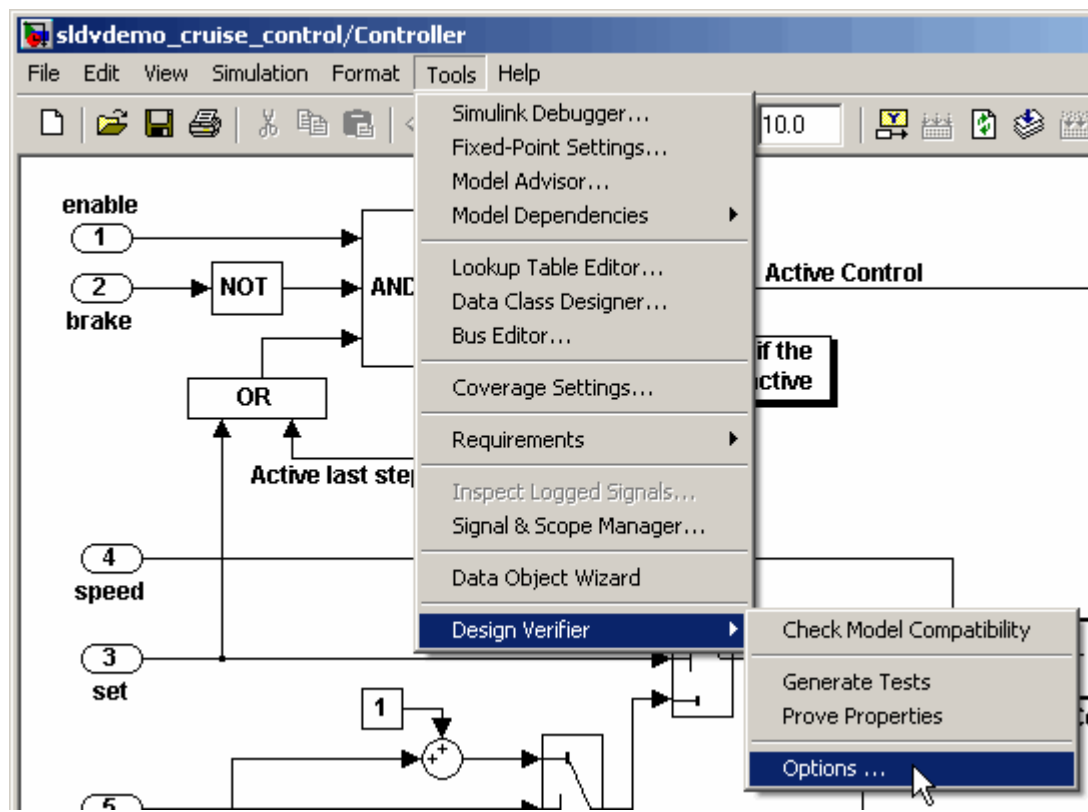
Describes the options that control Simulink Design Verifier.

Saving Simulink Design Verifier Options (p. 5-15)

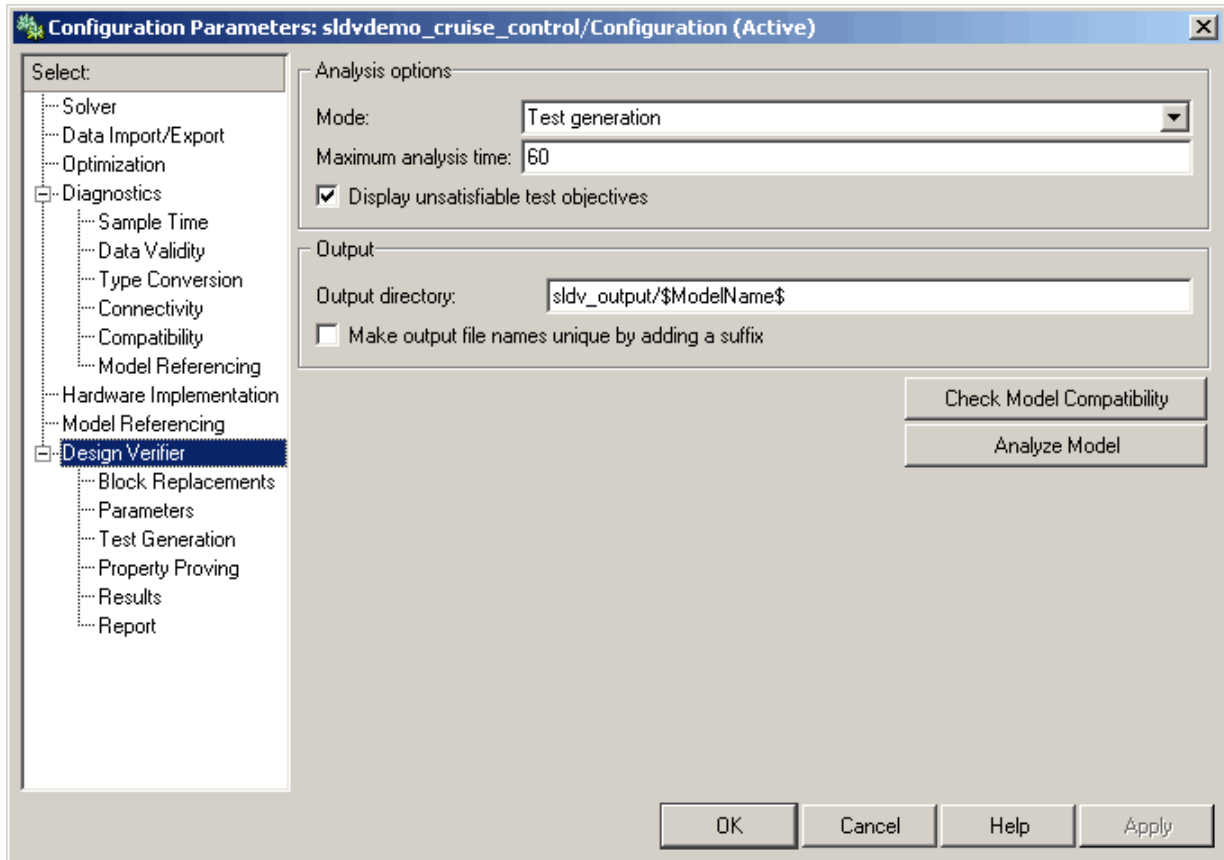
Discusses how Simulink Design Verifier saves its options.

## Viewing Simulink Design Verifier Options

Simulink Design Verifier 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**.



Simulink Design Verifier 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 the “Running Simulations” chapter of *Using Simulink* 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, Simulink Design Verifier 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 Simulink Design Verifier options associated with the Simulink model *system*:

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

See `sldvoptions` in Chapter 9, “Functions — Alphabetical List” for more information.



## Configuring Simulink Design Verifier Options

This section describes the options that control Simulink Design Verifier. Groupings of options appear on panes of the Configuration Parameters dialog box. See the following sections for information on how to set specific Simulink Design Verifier options associated with those panes:

- “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 5-10
- “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 pane with the following settings:

- 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

Buttons at the bottom right: Check Model Compatibility, 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 Simulink Design Verifier analyzes Simulink models. It contains the following controls.

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

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

**Display unsatisfiable test objectives.** If selected, this option causes Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier to append an incremental numeric suffix to output file names. Selecting this option prevents Simulink Design Verifier from overwriting existing files that have the same name.

### Block Replacements Pane

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

Block replacements

Apply block replacements

List of block replacement rules (in order of priority):

Output model

File path of the output model: \_\_\_\_\_

## 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 Simulink Design Verifier 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 Simulink Design Verifier processes before analyzing the model. This option is accessible only if **Apply block replacements** is selected. Simulink Design Verifier 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 3-14).

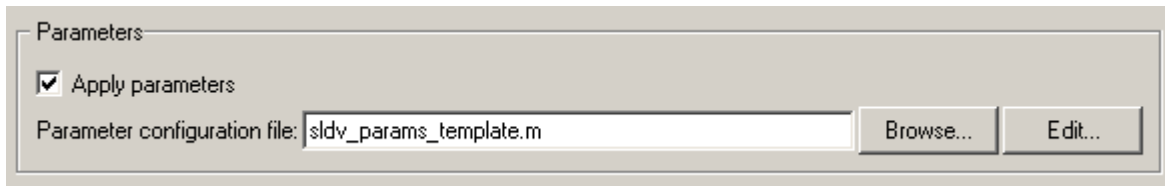
The default value is <FactoryDefaultRules>. If you specify the default value, Simulink Design Verifier uses its factory default block replacement rules (see “Built-In Block Replacements” on page 3-3).

**File path of the output model.** Specifies a directory to which Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier to use parameter configurations when analyzing a model. 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.

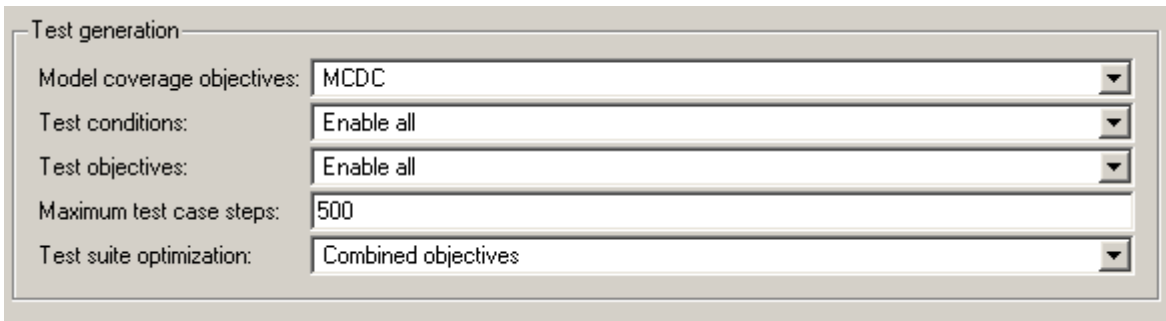
---

**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 Simulink Design Verifier 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 Simulink Design Verifier 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.

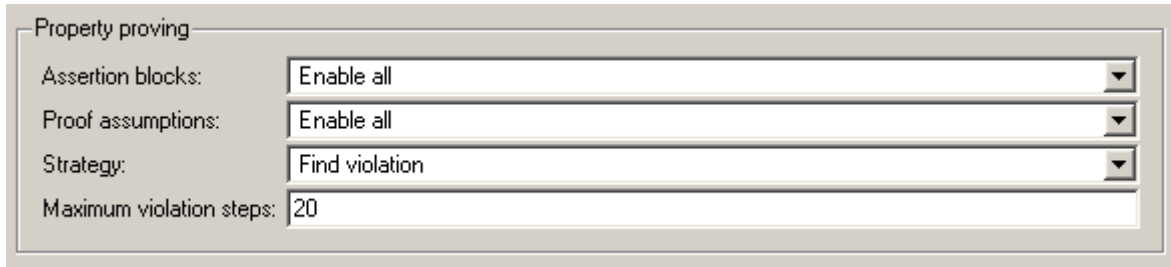
**Maximum test case steps.** Specifies the maximum number of simulation steps Simulink Design Verifier takes when attempting to satisfy a test objective.

**Test suite optimization.** This option allows you to specify whether each of the test cases generated by Simulink Design Verifier maps to a single test objective or multiple test objectives. Select one of the following options:

- **Combined objectives** — Generates test cases that can address more than one test objective.
- **Individual objectives** — Generates test cases that each address only one test objective.

### Property Proving Pane

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



Property proving

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 Simulink Design Verifier uses when proving properties. Select one of the following options:

- **Find violation** — If this strategy is selected, Simulink Design Verifier 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, Simulink Design Verifier performs property proofs.
- **Prove with violation detection** — This strategy combines the **Find violation** and **Prove** strategies. If selected, Simulink Design Verifier 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.

**Maximum violation steps.** Specifies the maximum number of simulation steps over which Simulink Design Verifier searches for property violations. Simulink Design Verifier 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 Simulink Design Verifier handles the results that it generates.

The screenshot shows a configuration dialog with two sections: "Harness model options" and "Data file options".

**Harness model options**

- Save test harness as model
- Harness model file name:

**Data file options**

- Save test data to file
- Data file name:



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

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

### **Harness model options**

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

**Save test harness as model.** If selected, this option causes Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier handles the MAT-file it produces. It contains the following controls.

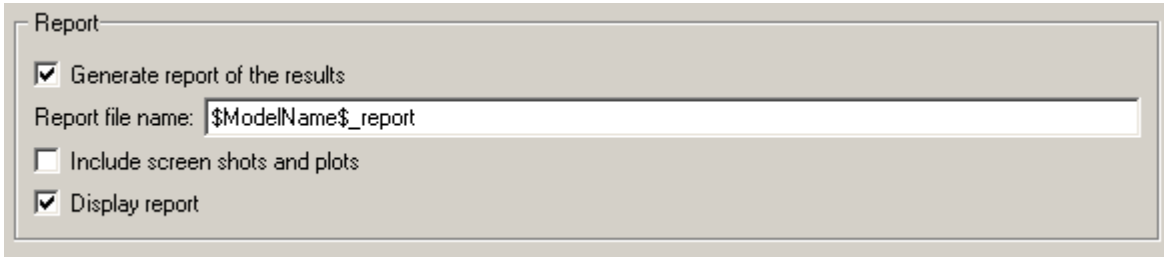
**Save test data to file.** If selected, this option causes Simulink Design Verifier to save the test data it generates to a MAT-file. Enabling this option provides access to the **Data file name** option.

**Data file name.** Specifies a file name with which Simulink Design Verifier 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.

### Report Pane

The **Report** pane allows you to specify options that control how Simulink Design Verifier 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 Simulink Design Verifier to save the HTML report it generates. 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 Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier 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**

Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier.
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

Simulink Design Verifier 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, Simulink Design Verifier performs a formal analysis of your model. After completing its analysis, Simulink Design Verifier 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.

Simulink Design Verifier 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 the analysis Simulink Design Verifier conducts. For more information about these blocks, see Chapter 10, "Blocks — Alphabetical List".

Simulink Design Verifier 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 Simulink Design Verifier (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 6-20).
- 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 6-9).
- 4** Execute the Simulink Design Verifier analysis and review its results (for examples, see “Analyzing the Example Model” on page 6-12 and “Reanalyzing the Example Model” on page 6-24).

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

## Generating Test Cases Example

To understand the test generation capabilities of Simulink Design Verifier, you build a simple Simulink model and generate test cases by completing a series of incremental tasks. The following sections guide you through the process of completing this example:

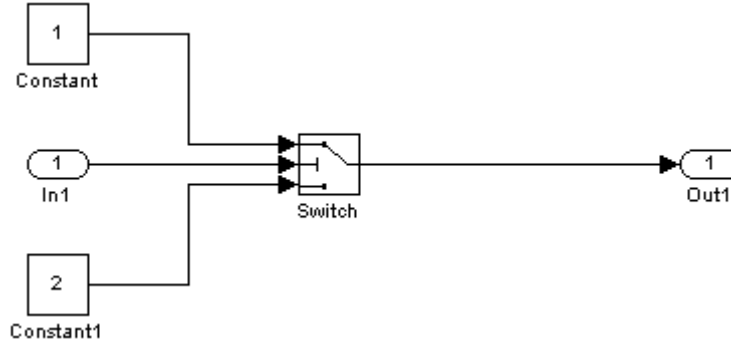
Constructing the Example Model (p. 6-4)	Guides you through Task 1 of the test generation example, in which you construct the example model.
Checking Compatibility of the Example Model (p. 6-6)	Guides you through Task 2 of the test generation example, in which you ensure your model's compatibility with Simulink Design Verifier.
Configuring Test Generation Options (p. 6-9)	Guides you through Task 3 of the test generation example, in which you configure Simulink Design Verifier to generate tests.
Analyzing the Example Model (p. 6-12)	Guides you through Task 4 of the test generation example, in which you generate test cases for your model and interpret the results.
Customizing Test Generation (p. 6-20)	Guides you through Task 5 of the test generation example, in which you add a Test Condition block to customize test generation.
Reanalyzing the Example Model (p. 6-24)	Guides you through Task 6 of the test generation example, in which you generate test cases for your modified model and interpret the results.

### Constructing the Example Model

This section presents Task 1 of the process that describes how to generate test cases with Simulink Design Verifier. 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 the Simulink documentation for help with this step).
- 2 Copy the following blocks into your empty model window (see “Adding Blocks” in the Simulink documentation for help with this step):
  - An Inport block to initiate the input signal, from the Sources library
  - 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 the Blocks” 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).

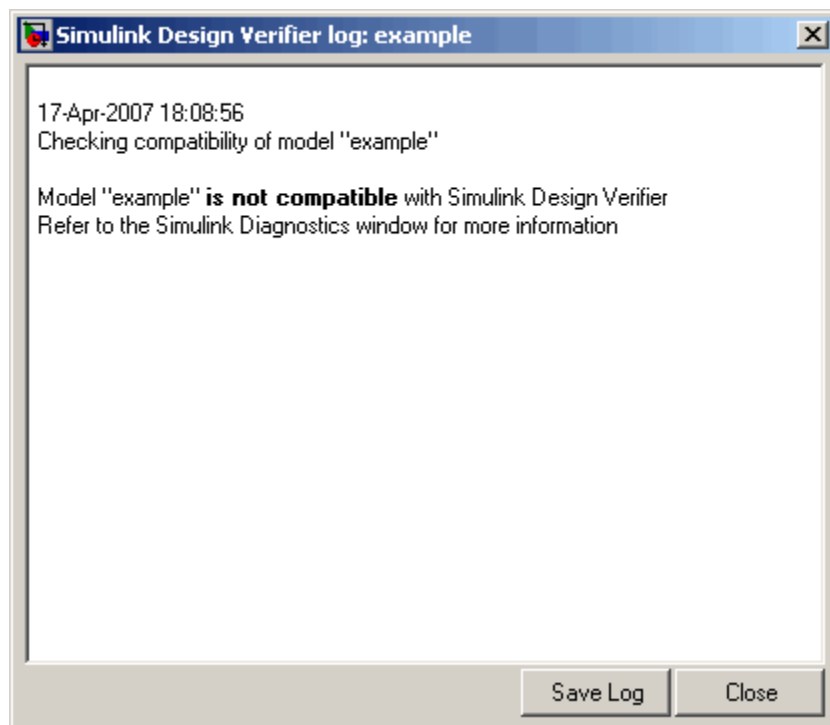
**What to do next:** Now you are ready to begin Task 2 of this example, “Checking Compatibility of the Example Model” on page 6-6.

## Checking Compatibility of the Example Model

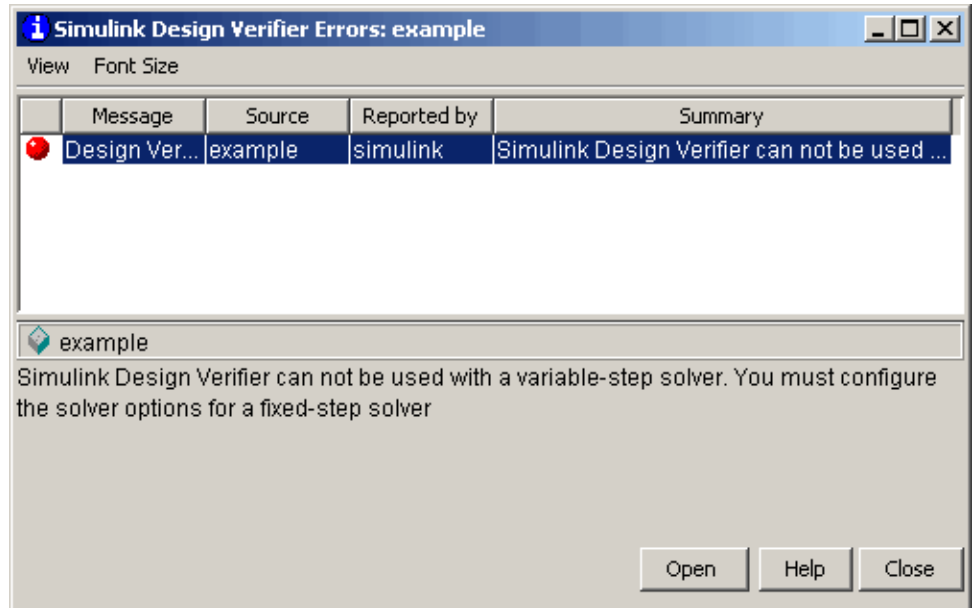
This section presents Task 2 of the process that describes how to generate test cases with Simulink Design Verifier. In this task, you ensure that a model is compatible for use with Simulink Design Verifier. Specifically, you check the compatibility of the simple Simulink model that you created in the previous task (see “Constructing the Example Model” on page 6-4). To complete this task, perform the following steps:

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

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



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



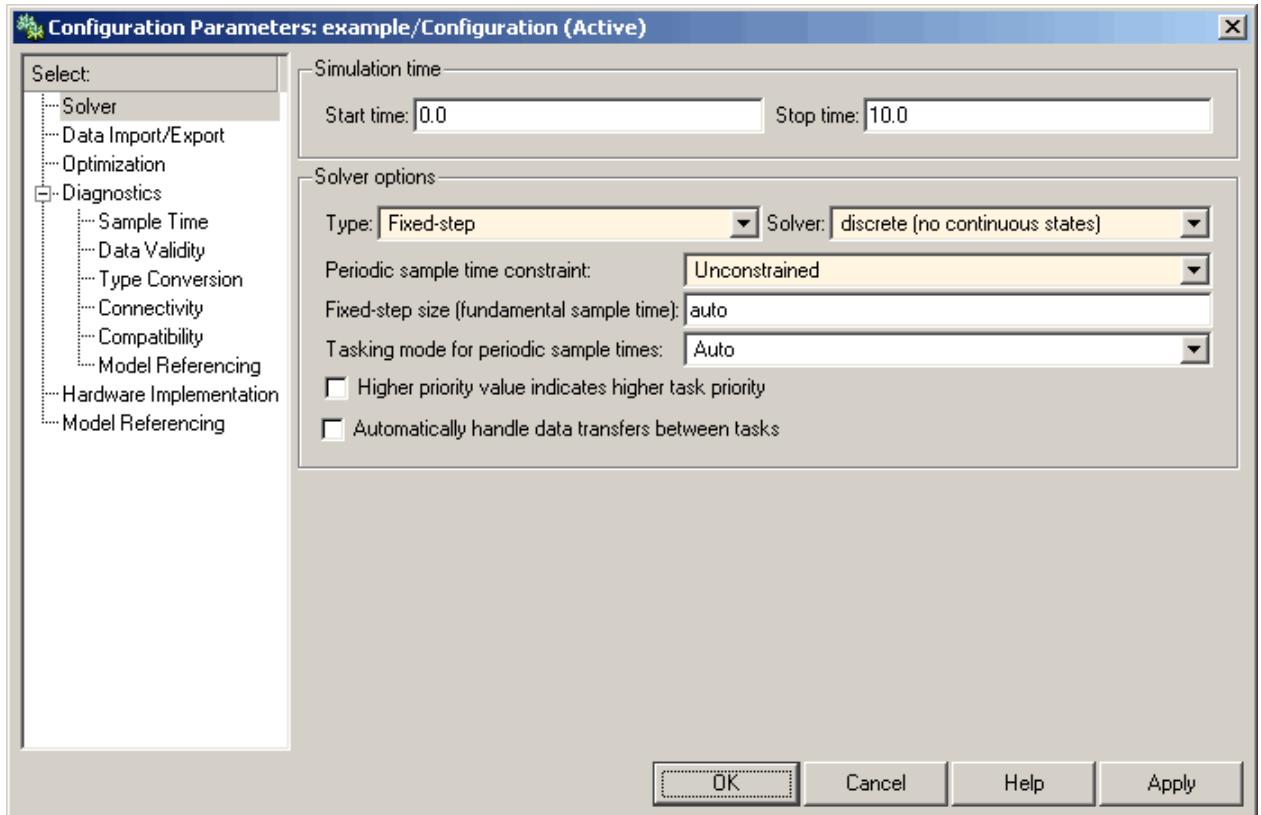
The error message informs you that Simulink Design Verifier does not support variable-step solvers. To work around this incompatibility, you must use a fixed-step solver.

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

Simulink displays the Configuration Parameters dialog box.

- 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).

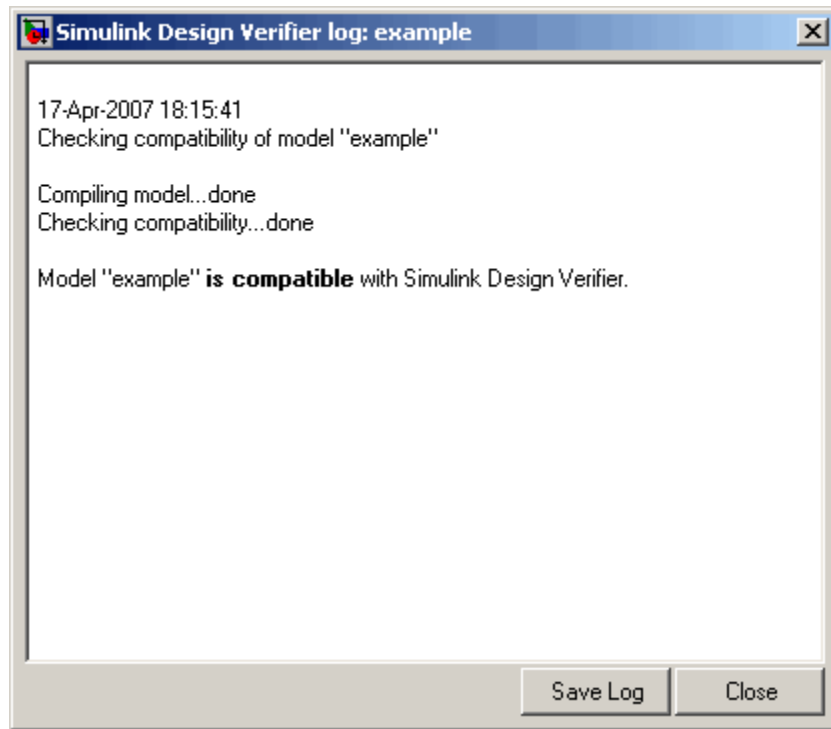
The Configuration Parameters dialog box appears as follows:



**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**.

Simulink Design Verifier 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 6-9.

## Configuring Test Generation Options

This section presents Task 3 of the process that describes how to generate test cases with Simulink Design Verifier. In this task, you configure Simulink Design Verifier 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 6-4). 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).

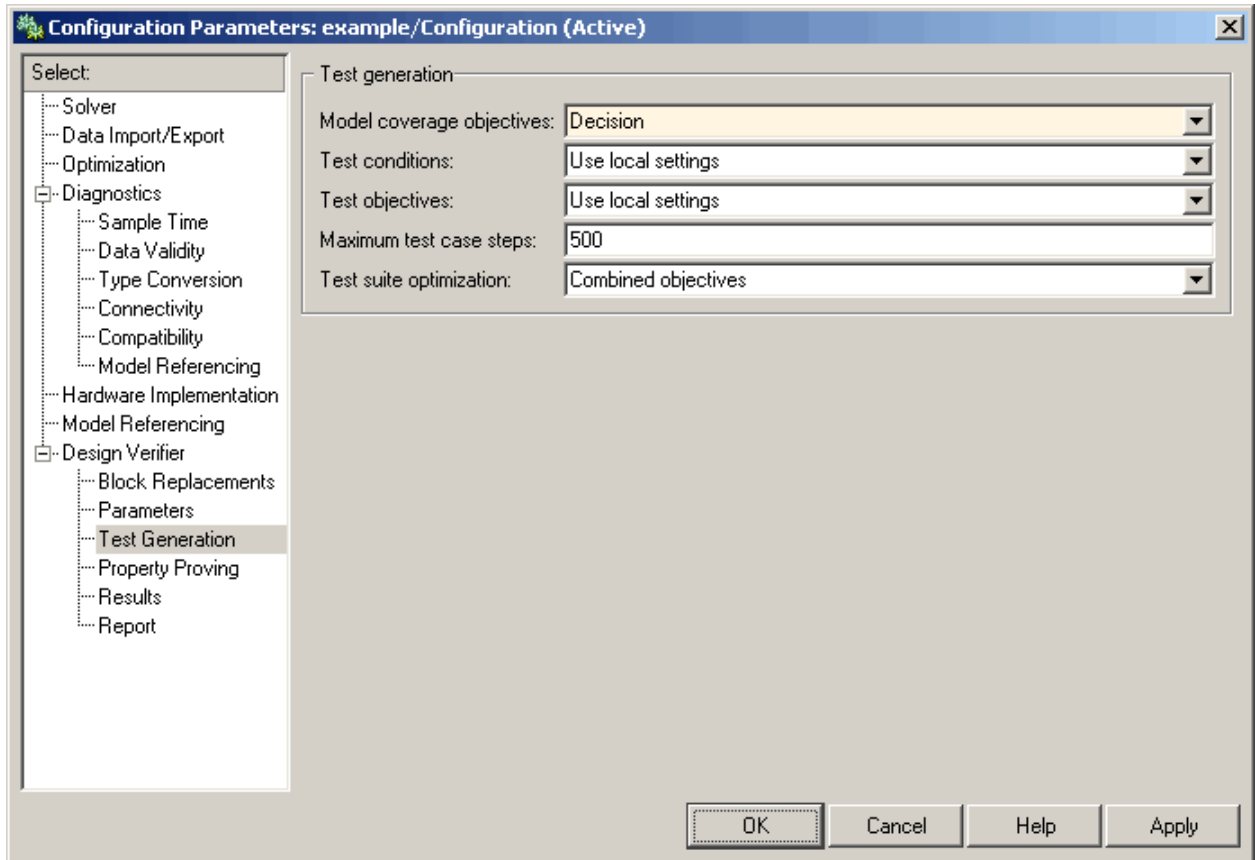
Simulink Design Verifier displays its options in the Configuration Parameters dialog box.

- 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 Simulink Design Verifier 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 6-12.

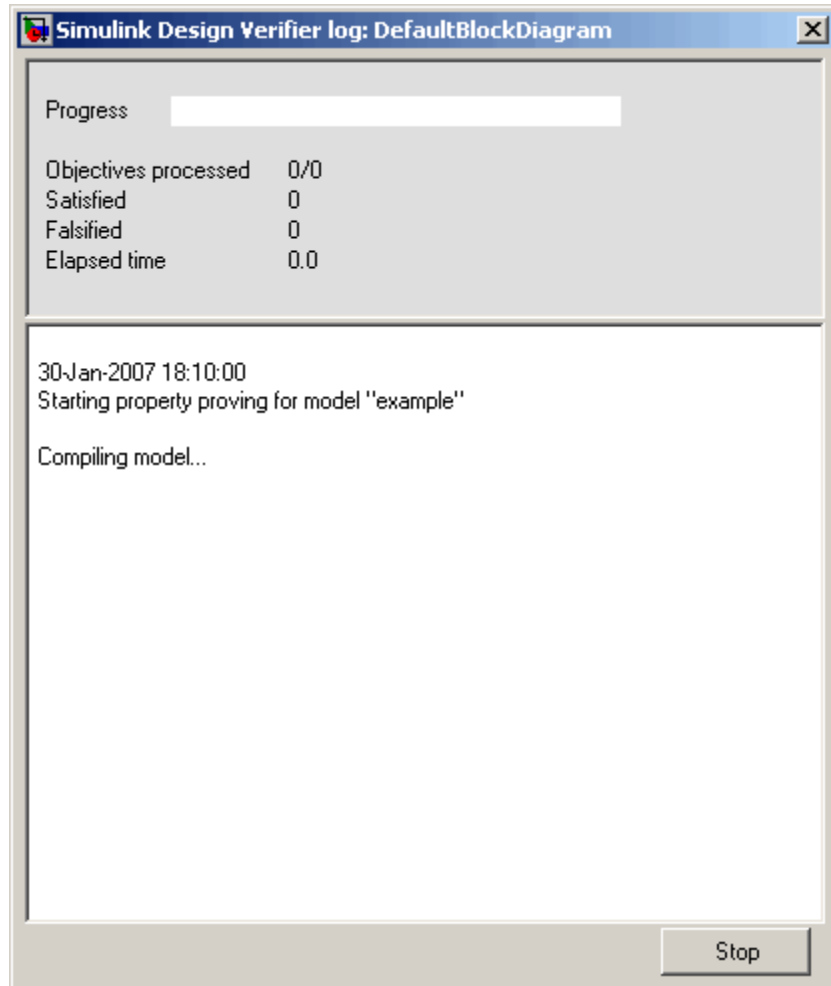
### Analyzing the Example Model

This section presents Task 4 of the process that describes how to generate test cases with Simulink Design Verifier. In this task, you execute the Simulink Design Verifier analysis, which you configured in the previous task (see “Configuring Test Generation Options” on page 6-9). Simulink Design Verifier 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**.

Simulink Design Verifier begins analyzing your model to generate test cases. During its analysis, Simulink Design Verifier 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 Simulink Design Verifier completes its analysis, it displays the following items:

- Simulink Design Verifier report — Simulink Design Verifier displays an HTML report named `example_report.html`.
- Test harness — Simulink Design Verifier 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/Proof Objectives](#)
  - [Status](#)
  - [example](#)
- [3. Test Cases / Counterexamples](#)
  - [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: Tue Apr 17 18:08:49 2007  
 Author: scowan

### Analysis Information

Design Verifier Version: 1.0  
 Total Analysis Time: 0.1 secs  
 Status: Completed normally  
[Approximations:](#) 1  
[Objectives Satisfied:](#) 2  
 Objectives Proven 0  
 Unsatisfiable: 0  
 Objectives Undecided: 0  
 Objectives Producing Errors: 0

The Summary chapter provides an overview of the analysis results. In particular, Simulink Design Verifier 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/Proof Objectives chapter.

**Table 2.1. Objectives Satisfied**

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

This table lists the test objectives that Simulink Design Verifier satisfied. Specifically, it describes the test objectives that provide decision coverage for a Switch block. You can locate the model item by clicking Switch; Simulink Design Verifier 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	<a href="#">TC 2</a>	false (output is from 3rd input port)
2	Satisfied	<a href="#">TC 1</a>	true (output is from 1st input port)

This table informs you that Simulink Design Verifier 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:

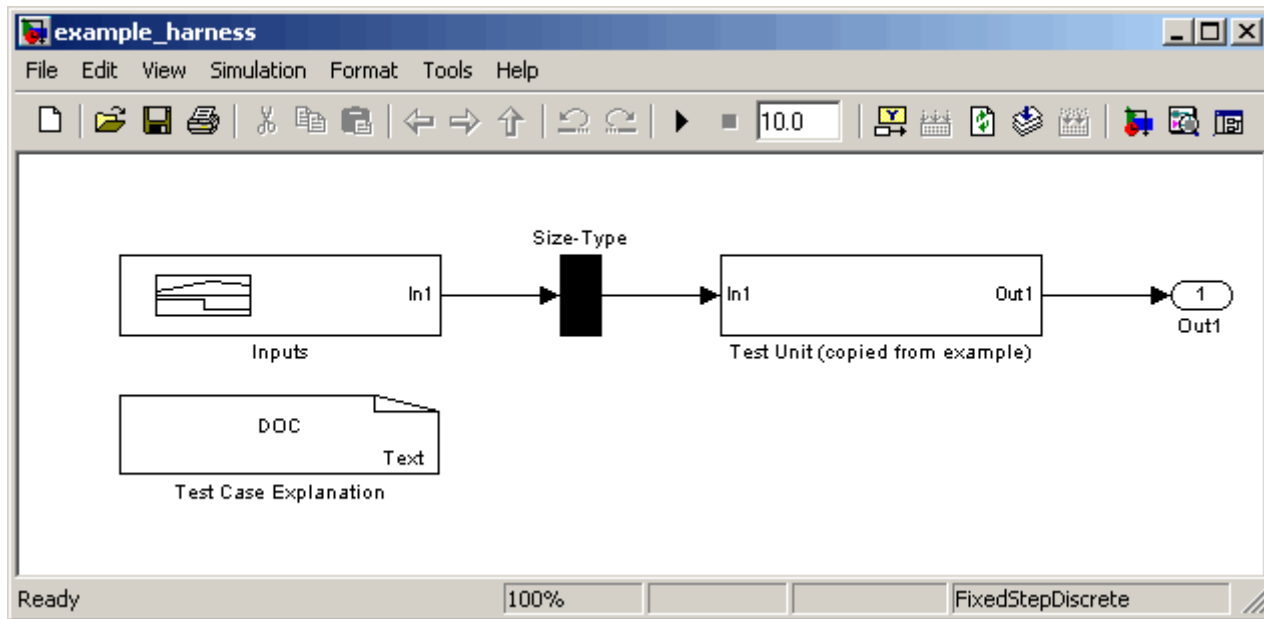
Step	Time	Objectives
1	0	<a href="#">1</a>

### Generated Input Data.

<b>Time 0</b>	
<b>Step 1</b>	
In1	-1

This section provides details about a test case that Simulink Design Verifier 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, Simulink Design Verifier 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 Simulink Design Verifier generates.

---

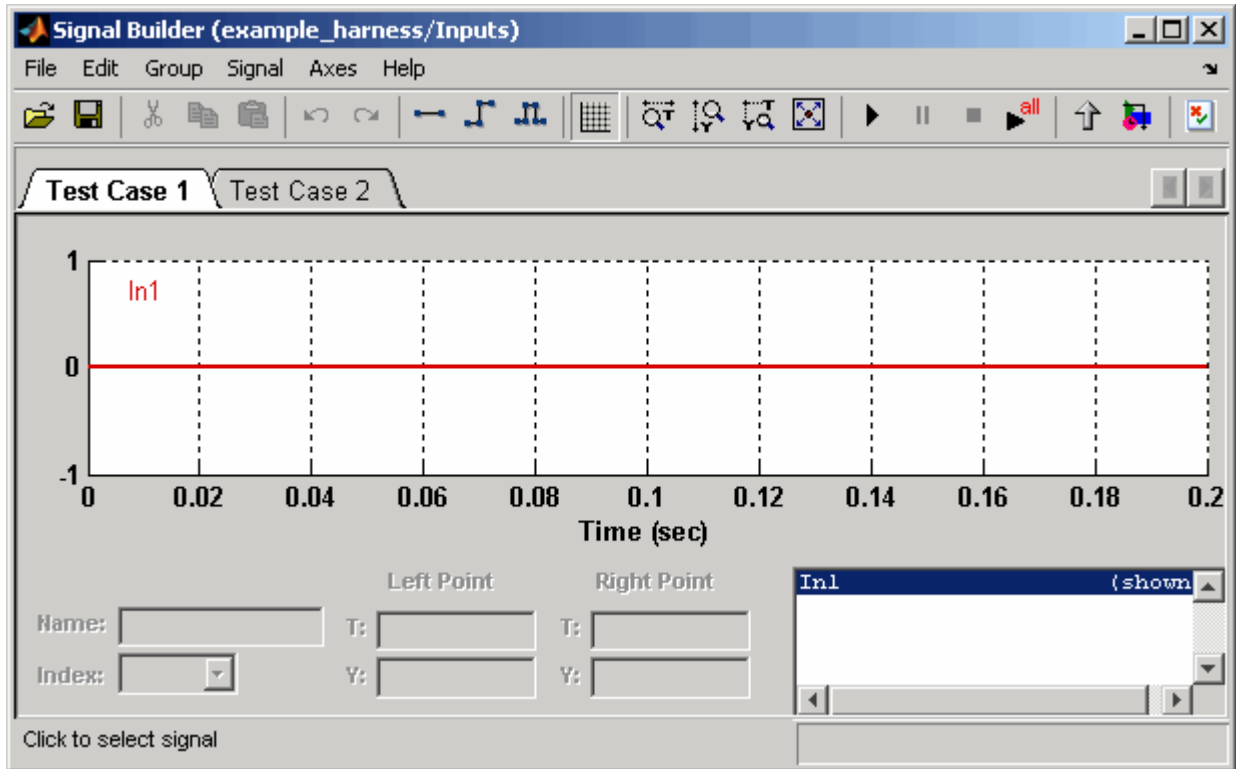
**Note** See the *Simulink Reference* for more information about interacting with blocks such as the Signal Builder, Subsystem, and DocBlock.


---

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

- 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 .

Simulink 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. <a href="#">example_harness</a>	2	100% 
2. . . . <a href="#">Test Unit (copied from example)</a>	1	100% 

The coverage report indicates Simulink Design Verifier 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 6-20.

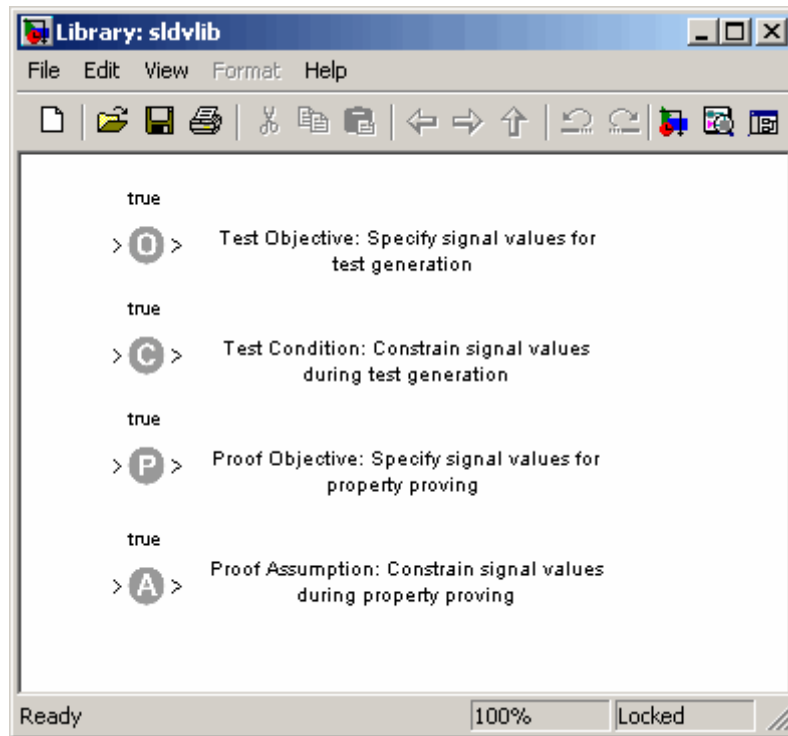
### Customizing Test Generation

This section presents Task 5 of the process that describes how to generate test cases with Simulink Design Verifier. 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 6-12). 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`.

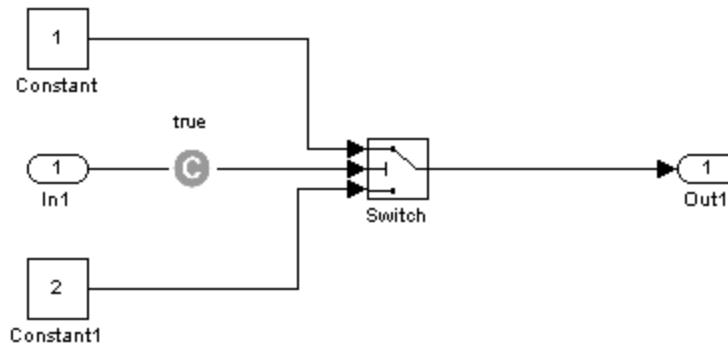
The Simulink Design Verifier library appears.





- 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:

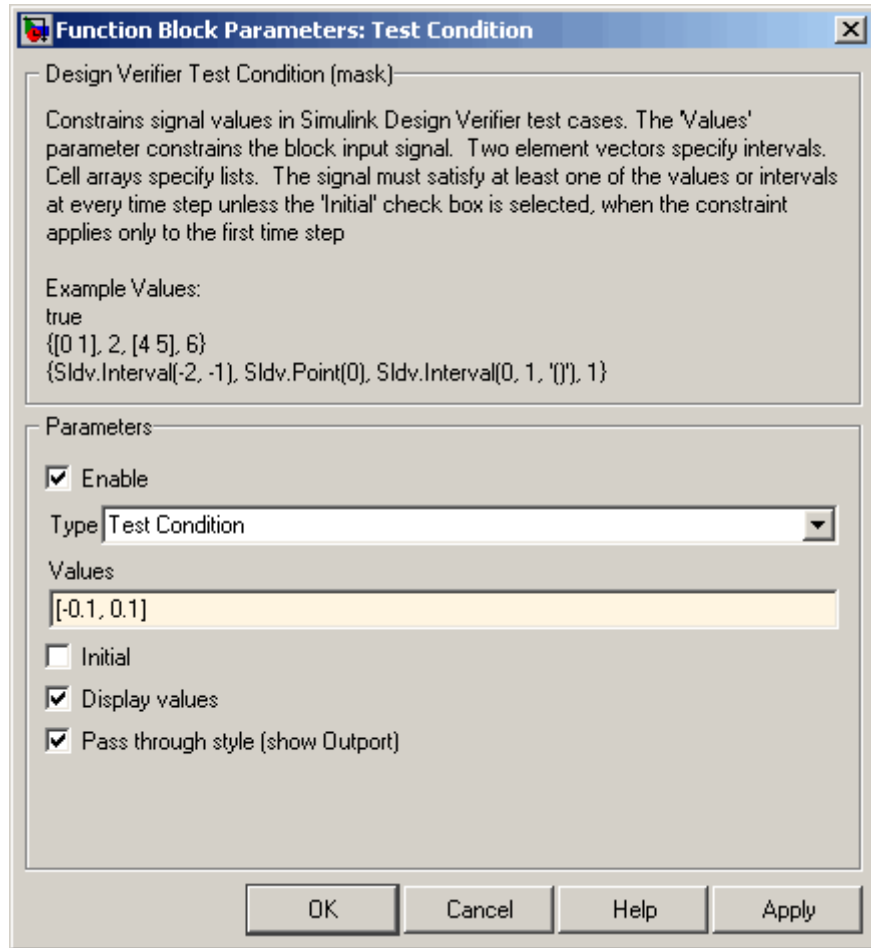


- 4 Double-click the Test Condition block in your model to access its attributes.

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, Simulink Design Verifier will constrain the signal values entering the Switch block control port to the specified interval.

The Test Condition block parameter dialog box appears.



**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 6-24.

## Reanalyzing the Example Model

This section presents Task 6 of the process that describes how to generate test cases with Simulink Design Verifier. 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 6-20). 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 6-12). To complete this task, perform the following steps:

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

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

When Simulink Design Verifier 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: Tue Apr 17 19:20:04 2007  
Author: scowan

### Analysis Information

Design Verifier Version: 1.0  
Total Analysis Time: 0.11 secs  
Status: Completed normally  
[Approximations:](#) 1  
[Objectives Satisfied:](#) 2  
Objectives Proven 0  
Unsatisfiable: 0  
Objectives Undecided: 0  
Objectives Producing Errors: 0

The Summary chapter indicates that Simulink Design Verifier 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/Proof Objectives chapter.

Table 2.1. Objectives Satisfied

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

With the following active constraints:

Name	Constraint
<a href="#">Test Condition</a>	[-0.1, 0.1]

This table lists the test objectives that Simulink Design Verifier satisfied. It also identifies any active constraints that Simulink Design Verifier 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	<a href="#">TC 2</a>	false (output is from 3rd input port)
2	Satisfied	<a href="#">TC 1</a>	true (output is from 1st input port)

This table informs you that Simulink Design Verifier 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	<a href="#">1</a>

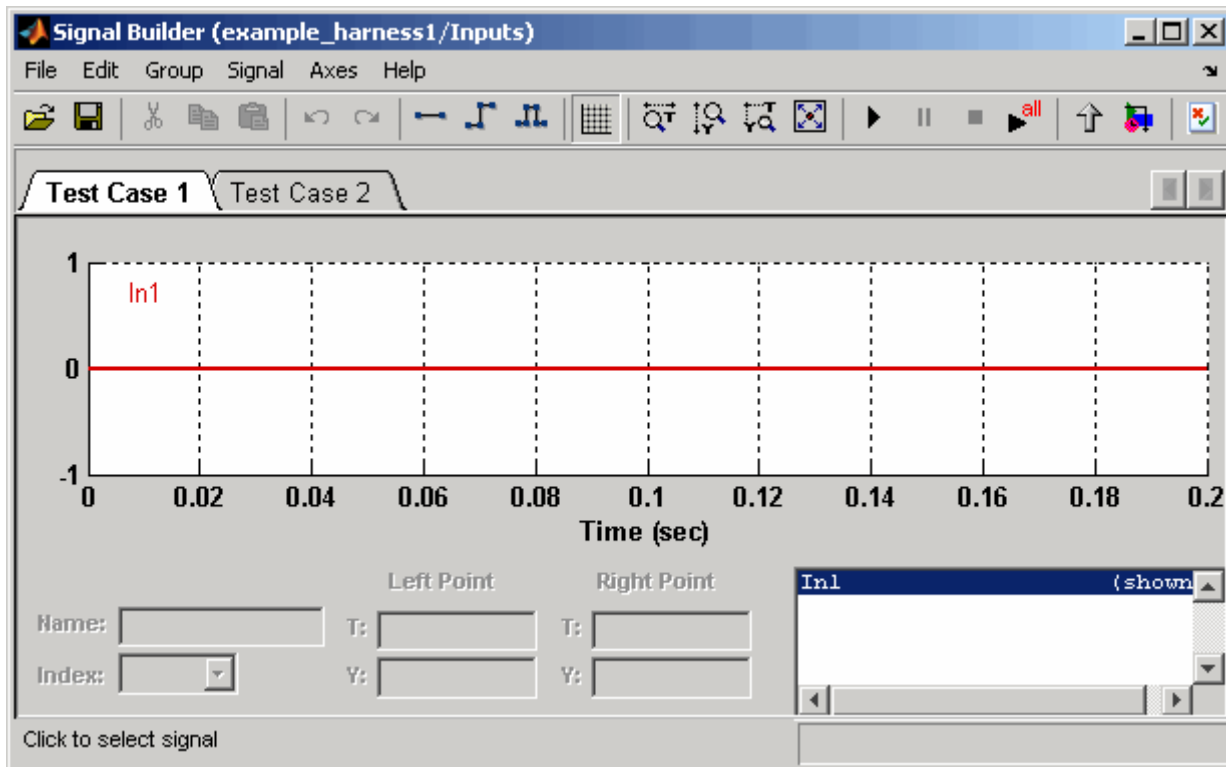
### Generated Input Data.


<b>Time 0</b>	
<b>Step 1</b>	
In1	-0.05

This section provides details about a test case that Simulink Design Verifier 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  $[-0.1, 0.1]$ , Simulink Design Verifier 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 .

Simulink 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. <a href="#">example_harness1</a>	3 100% 
2. ... <a href="#">Test Unit (copied from another example)</a>	2 100% 



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



# Proving Properties of a Model

---

This chapter describes how you can use Simulink Design Verifier 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 Simulink Design Verifier.

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

Simulink Design Verifier 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 Simulink Design Verifier to prove whether such properties are valid. Simulink Design Verifier performs a formal analysis of your model to prove or disprove the specified properties. If Simulink Design Verifier disproves a property, it provides a counterexample that demonstrates a property violation.

Simulink Design Verifier 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 Simulink Design Verifier will prove. Use the Proof Assumption block to constrain the values of a signal during the proof Simulink Design Verifier conducts. For more information about these blocks, refer to Chapter 10, “Blocks — Alphabetical List”.

---

**Note** Blocks from the Model Verification library in Simulink 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.

---

Simulink Design Verifier 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:

```
sldv.prove(expr, "{values}")  
sldv.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 Simulink Design Verifier (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 7-9 and “Customizing the Example Proof” on page 7-22).
- 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 7-12).
- 4** Execute the Simulink Design Verifier analysis and review its results (for examples, see “Analyzing the Example Model” on page 7-14 and “Reanalyzing the Example Model” on page 7-24).

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

## Proving Model Properties Example

To understand the property proving capabilities of Simulink Design Verifier, build a simple Simulink model and prove a property by completing a series of incremental tasks. The following sections guide you through the process of completing this example:

Constructing the Example Model  
(p. 7-5)

Guides you through Task 1 of the example proof, in which you construct the example model.

Checking Compatibility of the Example Model (p. 7-6)

Guides you through Task 2 of the example proof, in which you ensure your model's compatibility with Simulink Design Verifier.

Instrumenting the Example Model  
(p. 7-9)

Guides you through Task 3 of the example proof, in which you add a Proof Objective block to your model to prepare for its proof.

Configuring Property Proving Options (p. 7-12)

Guides you through Task 4 of the example proof, in which you configure Simulink Design Verifier to prove properties.

Analyzing the Example Model  
(p. 7-14)

Guides you through Task 5 of the example proof, in which you prove a property of your model and interpret the results.

Customizing the Example Proof  
(p. 7-22)

Guides you through Task 6 of the example proof, in which you add a Proof Assumption block to customize the proof.

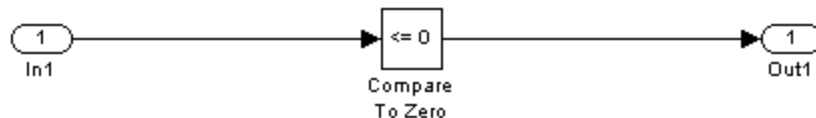
Reanalyzing the Example Model  
(p. 7-24)

Guides you through Task 7 of the example proof, in which you prove a property of your modified model and interpret the results.

## Constructing the Example Model

This section presents Task 1 of the process that describes how to implement an example proof with Simulink Design Verifier. 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 the Simulink documentation for help with this step).
- 2 Copy the following blocks into your empty model window (see “Adding Blocks” in the Simulink documentation for help with this step):
  - An Inport block to initiate the input signal, from the Sources library
  - 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 the Blocks” 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).

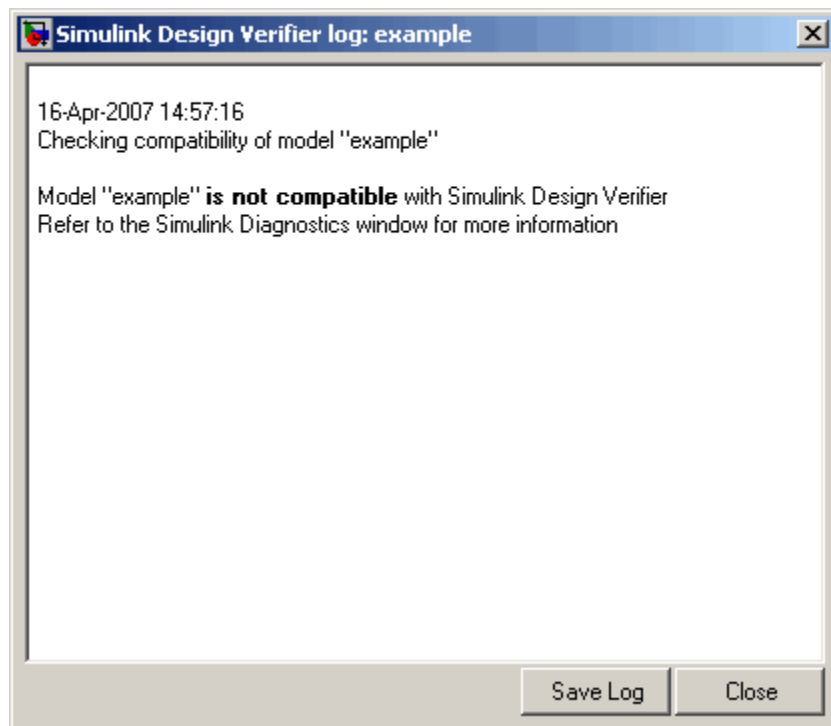
**What to do next:** Now you are ready to begin Task 2 of this example, “Checking Compatibility of the Example Model” on page 7-6.

### Checking Compatibility of the Example Model

This section presents Task 2 of the process that describes how to implement an example proof with Simulink Design Verifier. In this task, you ensure that a model is compatible for use with Simulink Design Verifier. 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:

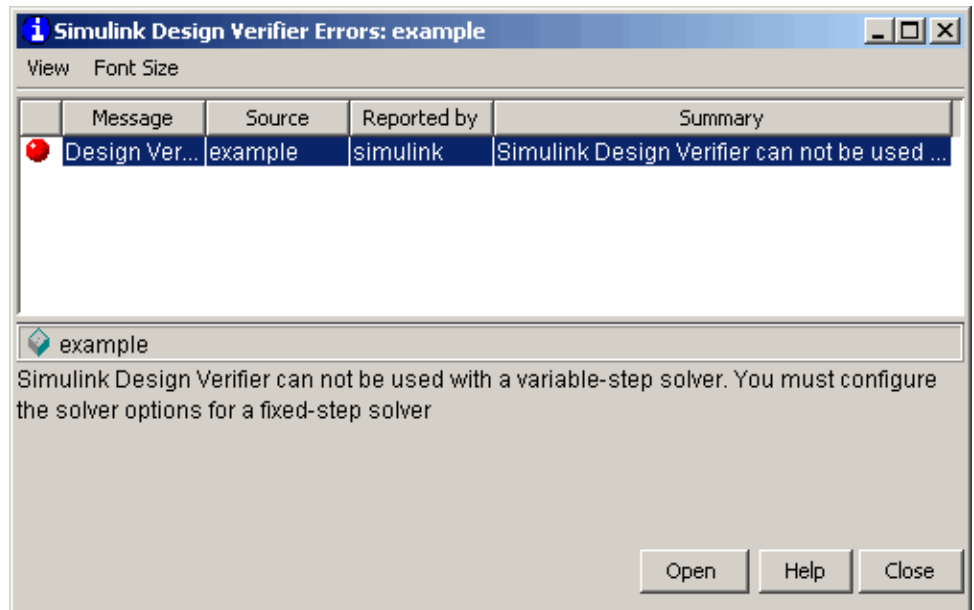
- 1 In your Simulink model window, select **Tools > Design Verifier > Check Model Compatibility**.

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



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





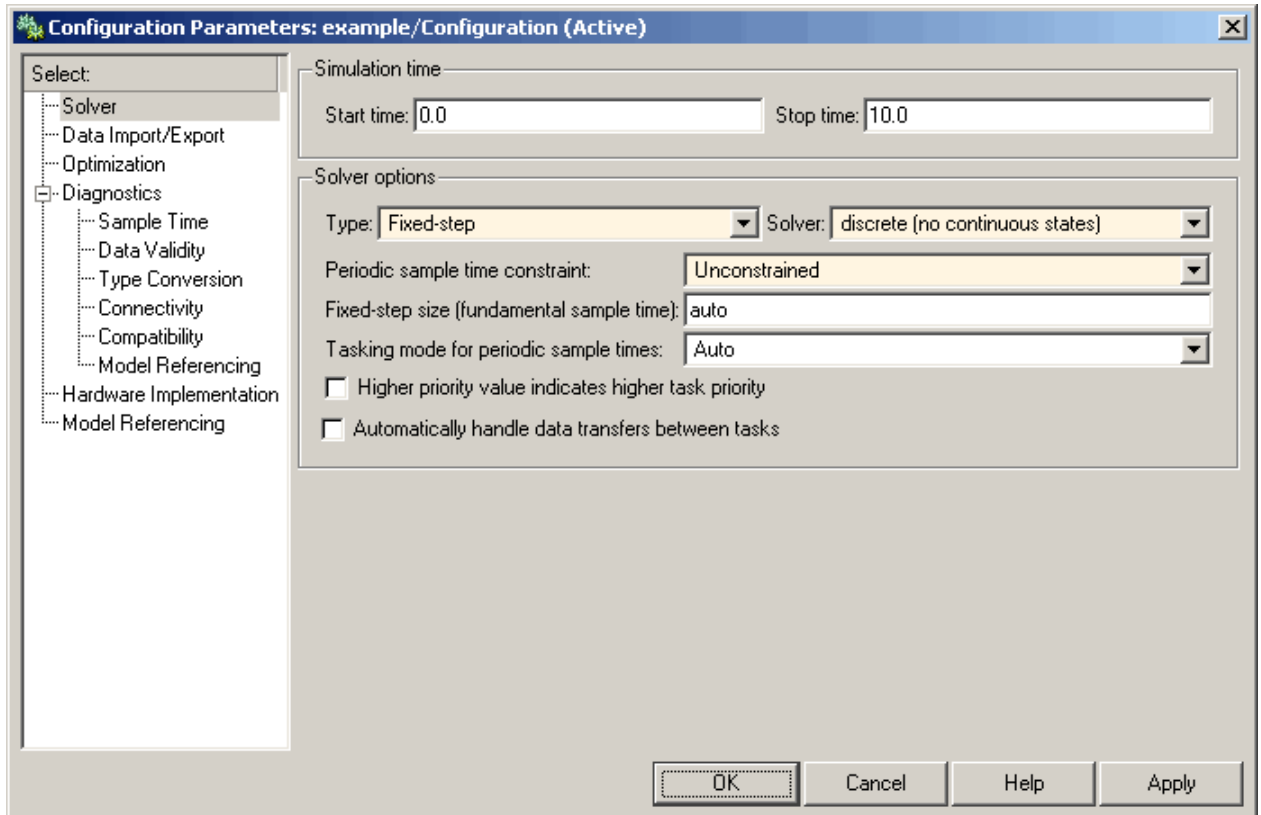
The error message informs you that Simulink Design Verifier does not support variable-step solvers. To work around this incompatibility, you must use a fixed-step solver.

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

Simulink displays the Configuration Parameters dialog box.

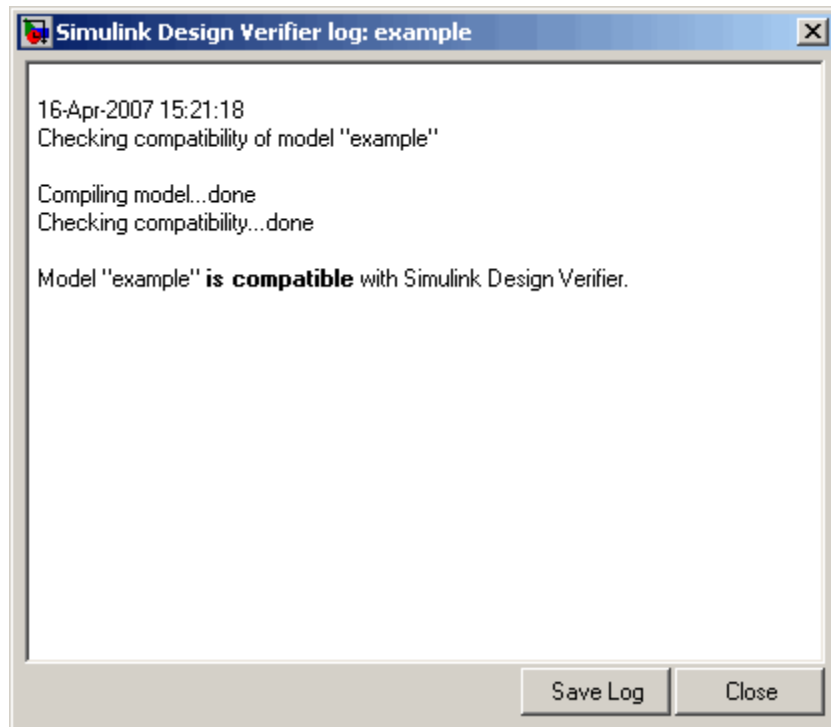
- 3 In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Solver** category (if not already selected). Under **Solver options** on the right side, set the **Type** option to Fixed-step, and then set the **Solver** option to discrete (no continuous states).

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**.

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



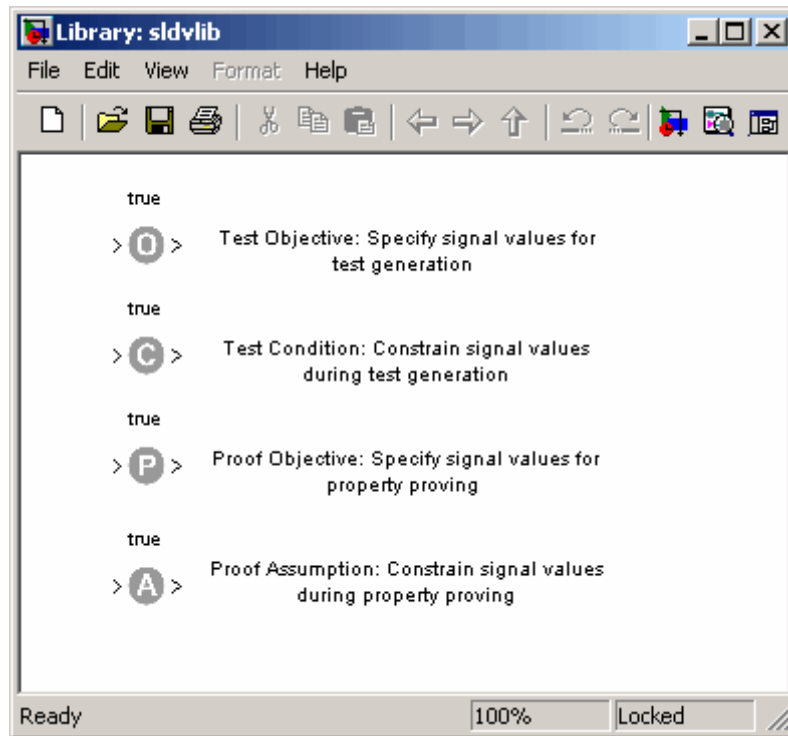
**What to do next:** Now you are ready to begin Task 3 of this example, “Instrumenting the Example Model” on page 7-9.

## Instrumenting the Example Model

This section presents Task 3 of the process that describes how to implement an example proof with Simulink Design Verifier. In this task, you prepare a model so that you can prove its properties with Simulink Design Verifier. 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:

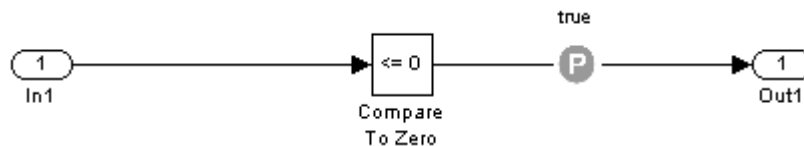
- 1 In the MATLAB Command Window, enter `sldvlib`.

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:

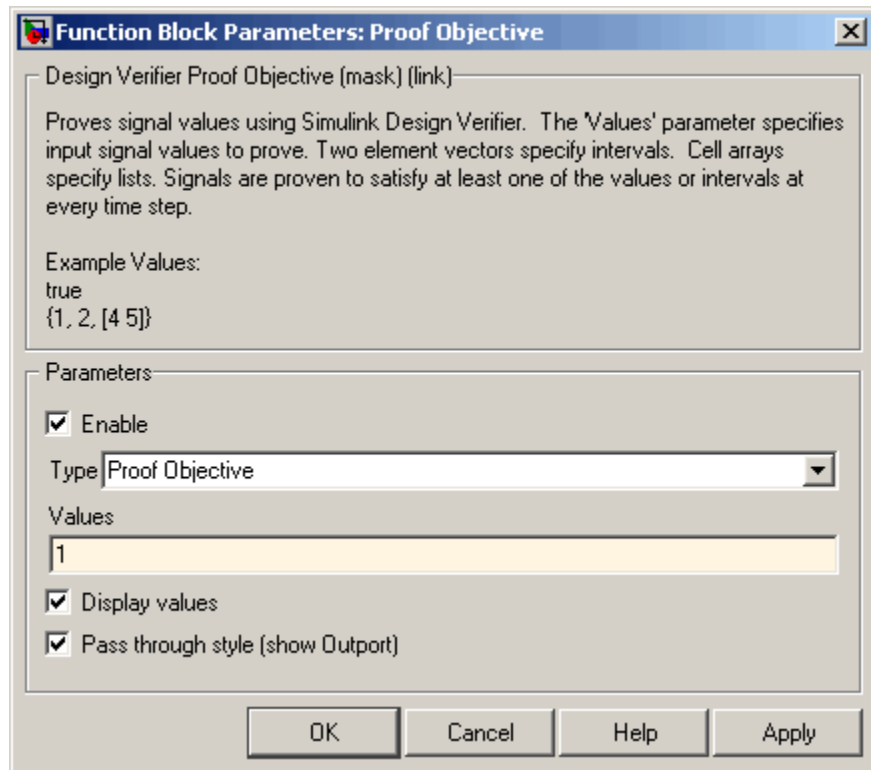


- 4** Double-click the Proof Objective block in your model to access its attributes.

The Proof Objective block parameter dialog box appears.

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

The Proof Objective block parameter dialog box appears.



- 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 7-12.

### Configuring Property Proving Options

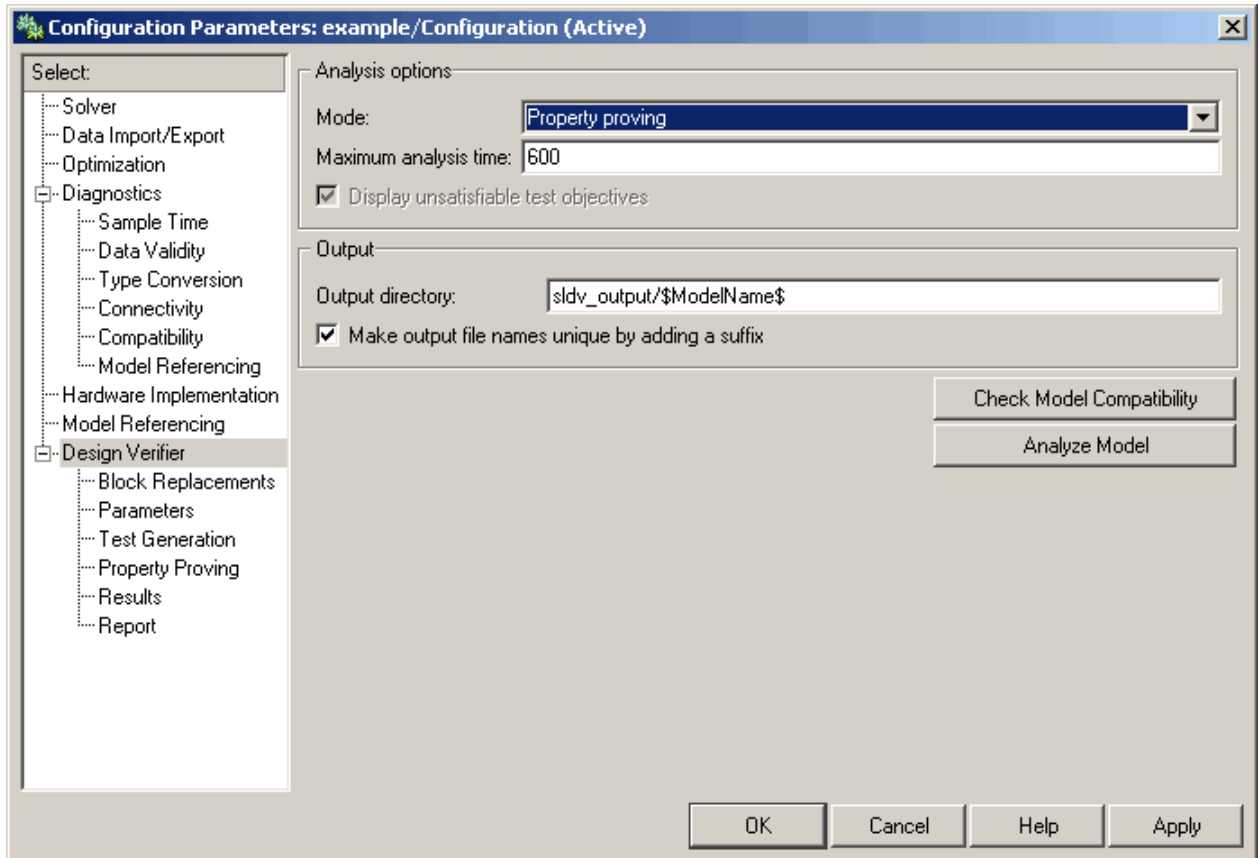
This section presents Task 4 of the process that describes how to implement an example proof with Simulink Design Verifier. In this task, you configure Simulink Design Verifier to prove properties of the simple Simulink model that you instrumented in the previous task (see “Instrumenting the Example Model” on page 7-9). 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).

Simulink Design Verifier 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 Simulink Design Verifier proves properties of your model. See “Property Proving Pane” on page 5-10 for more information.

---

**What to do next:** Now you are ready to begin Task 5 of this example, “Analyzing the Example Model” on page 7-14.

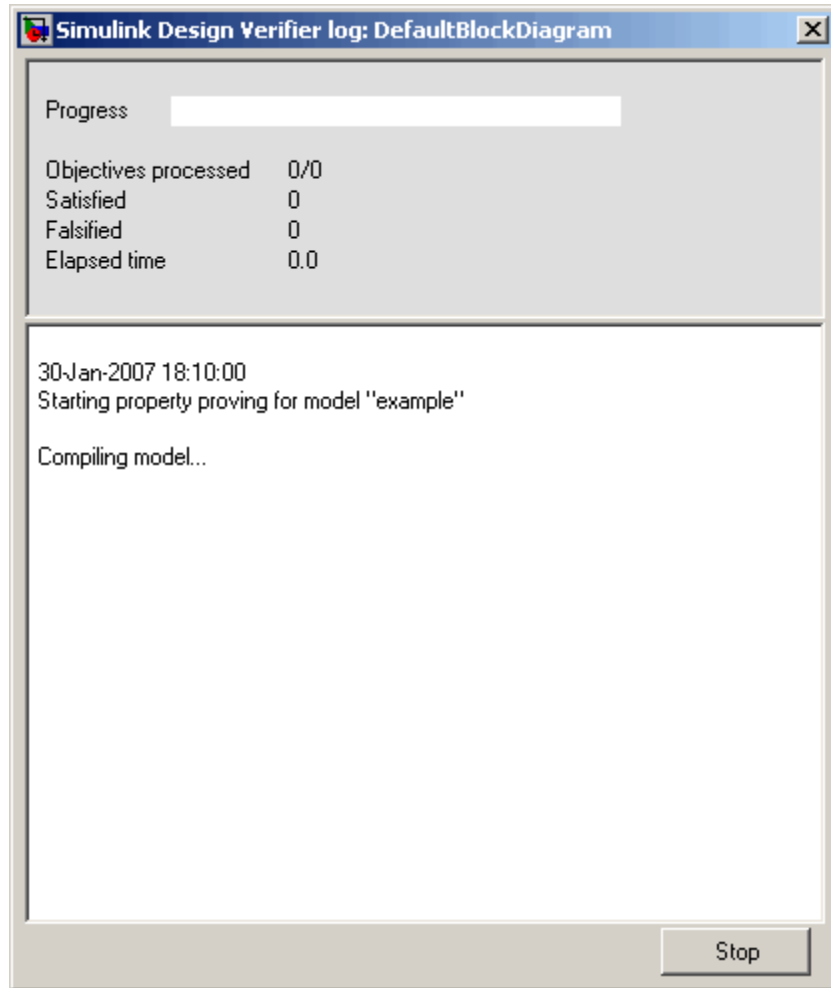
### Analyzing the Example Model

This section presents Task 5 of the process that describes how to implement an example proof with Simulink Design Verifier. In this task, you execute the Simulink Design Verifier analysis, which you configured in the previous task (see “Configuring Property Proving Options” on page 7-12). Simulink Design Verifier 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**.

Simulink Design Verifier begins analyzing your model to prove its properties. During its analysis, Simulink Design Verifier 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 Simulink Design Verifier completes its analysis, it displays the following items:

- Simulink Design Verifier report — Simulink Design Verifier displays an HTML report named `example_report.html`.
- Test harness — Simulink Design Verifier 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/Proof Objectives](#)
  - [Status](#)
  - [example](#)
- [3. Test Cases / 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: 1.3  
 Time Stamp: Mon Apr 16 16:58:59 2007  
 Author: scowan

### Analysis Information

Design Verifier Version: 1.0  
 Total Analysis Time: 0.03 secs  
 Status: Completed normally  
[Approximations:](#) 1  
 Objectives Proven Valid: 0  
[Objectives Falsified with Counterexamples:](#) 1  
 Objectives Undecided: 0  
 Objectives Producing Errors: 0

The Summary chapter provides an overview of the analysis results. In particular, Simulink Design Verifier 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 Test/Proof Objectives chapter.

**Table 2.1. Objectives Falsified with Counterexamples**

#:	Type	Model Item	Description
<a href="#">1</a>	Custom Proof Objective	<a href="#">Proof Objective</a>	Proof Objective "Proof Objective" : 1

This table lists the proof objectives that Simulink Design Verifier disproved using a counterexample it generated. You can locate the objective in your model window by clicking **Proof Objective**; Simulink Design Verifier 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	<a href="#">TC 1</a>	: 1

This table informs you that Simulink Design Verifier 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:

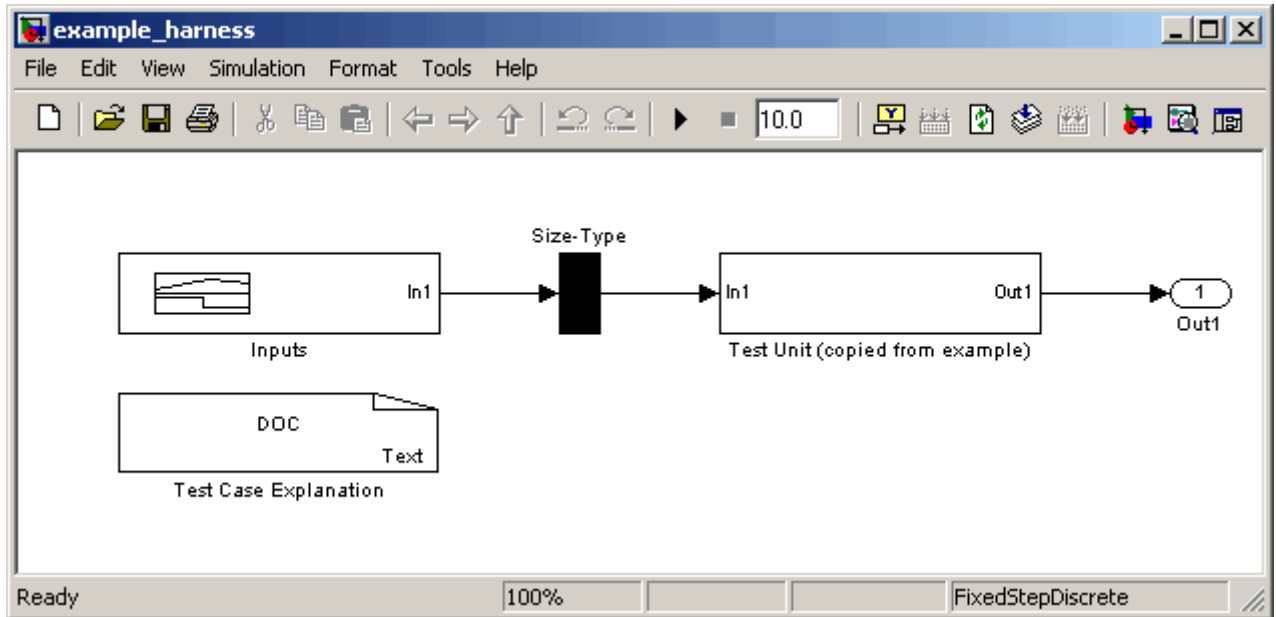
Objectives Falsified
1

### Generated Input Data.

<b>Time</b>	<b>0</b>
<b>Step</b>	<b>1</b>
In1	255

This section provides details about the counterexample that Simulink Design Verifier 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 Simulink Design Verifier generates.

---

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

---

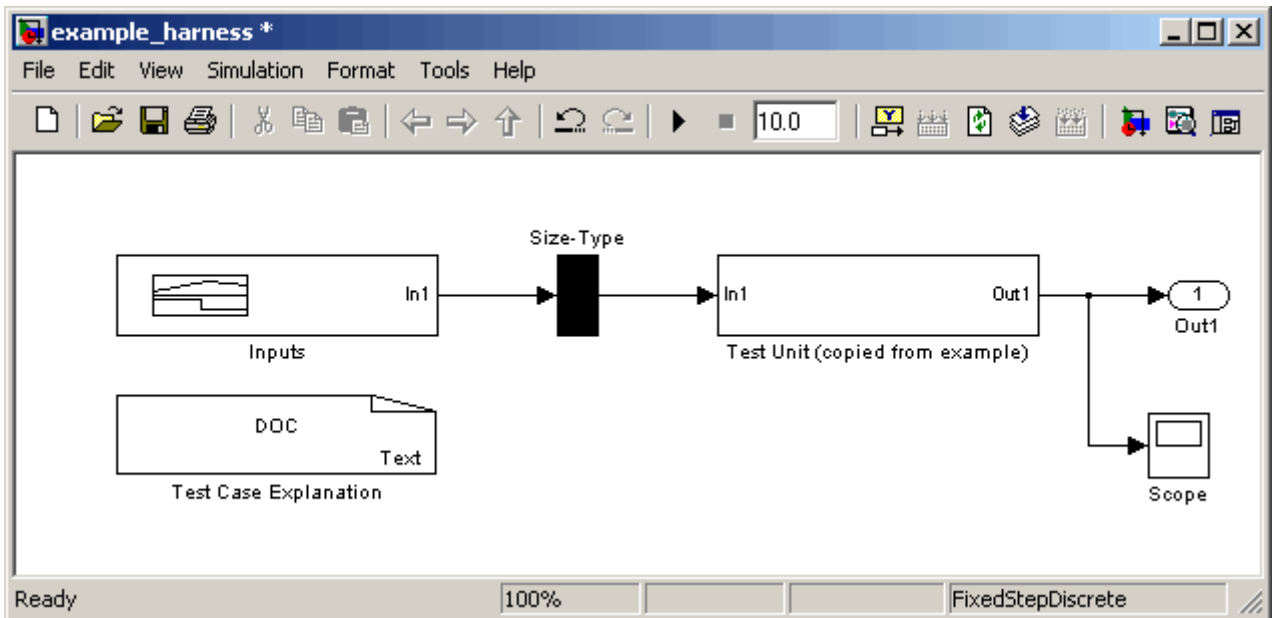
You can simulate the harness model to observe the counterexample that falsifies the proof objective in your model:

- 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:

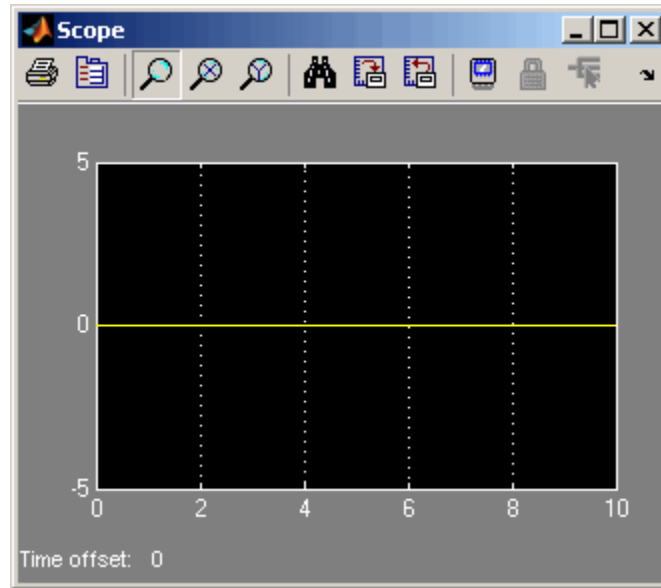


- d** In your harness model window, select **Simulation > Start** to begin the simulation.

Simulink 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 7-9) 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 7-22.

### Customizing the Example Proof

This section presents Task 6 of the process that describes how to implement an example proof with Simulink Design Verifier. In this task, you modify the simple Simulink model whose proof objective Simulink Design Verifier disproved in the previous task (see “Analyzing the Example Model” on page 7-14). Specifically, you customize the proof by adding and configuring a Proof Assumption block. To complete this task, perform the following steps:

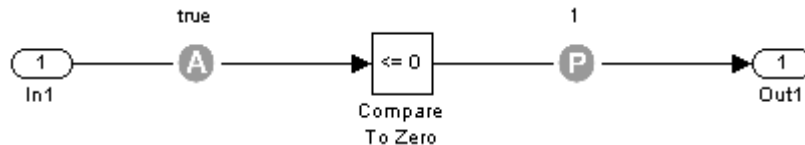


- 1 If the Simulink Design Verifier library is not already open, type `sldvlib` in the MATLAB Command Window.

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:

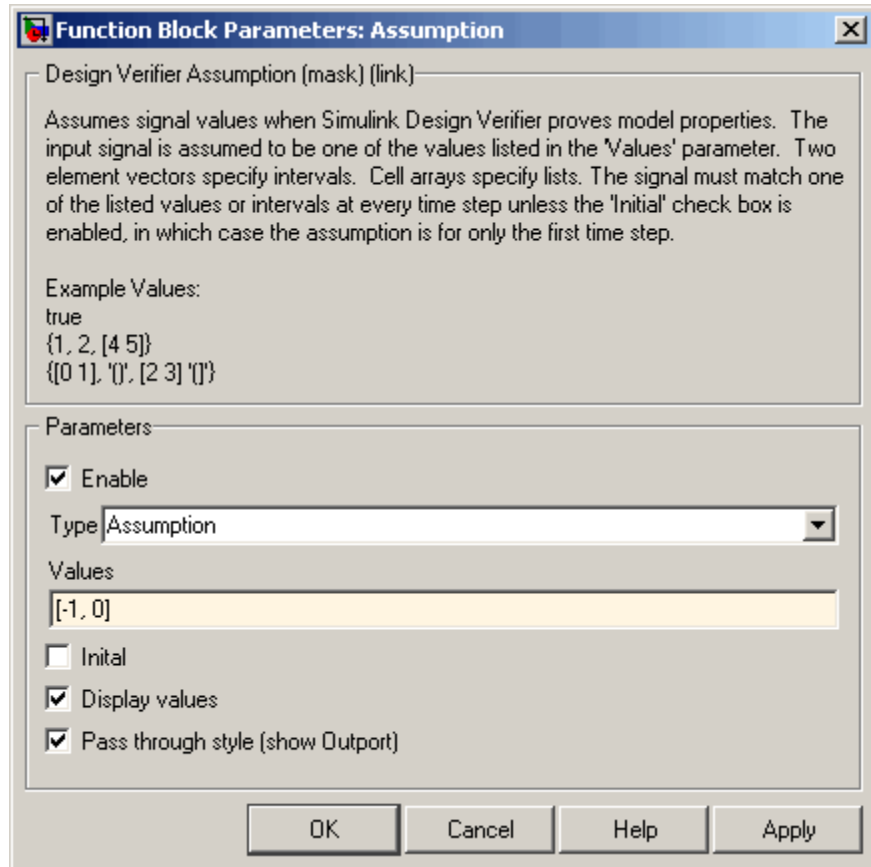


- 4 Double-click the Proof Assumption block in your model to access its attributes.

The Proof Assumption block parameter dialog box appears.

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

The Proof Assumption block parameter dialog box appears.



- 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 7-24.

### Reanalyzing the Example Model

This section presents Task 7 of the process that describes how to implement an example proof with Simulink Design Verifier. 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 7-22). 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 7-14). To complete this task, perform the following steps:

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

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

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

---

**Note** If Simulink Design Verifier 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: 1.4  
Time Stamp: Mon Apr 16 19:55:08 2007  
Author: scowan

### Analysis Information

Design Verifier Version: 1.0  
Total Analysis Time: 0.1 secs  
Status: Completed normally  
[Approximations:](#) 1  
[Objectives Proven Valid:](#) 1  
Objectives Falsified with Counterexamples: 0  
Objectives Undecided: 0  
Objectives Producing Errors: 0

The Summary chapter indicates that Simulink Design Verifier 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 Test/Proof Objectives chapter.

**Table 2.1. Objectives Proven Valid**

#:	Type	Model Item	Description
<a href="#">1</a>	Custom Proof Objective	<a href="#">Proof Objective</a>	Proof Objective "Proof Objective" : 1

With the following active constraints:

Name	Constraint
<a href="#">Assumption</a>	[-1, 0]

This table lists the proof objectives that Simulink Design Verifier 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 Simulink Design Verifier 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], Simulink Design Verifier was able to prove that this interval contains no values that are greater than zero, thereby satisfying the proof objective.



# Reviewing the Results

---

Simulink Design Verifier produces several artifacts after it analyzes your model. Depending on the analysis, Simulink Design Verifier 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. 8-6)

Describes the different parts of a  
Simulink Design Verifier report.

Examining Simulink Design Verifier  
Data Files (p. 8-21)

Describes the contents of a Simulink  
Design Verifier data file.

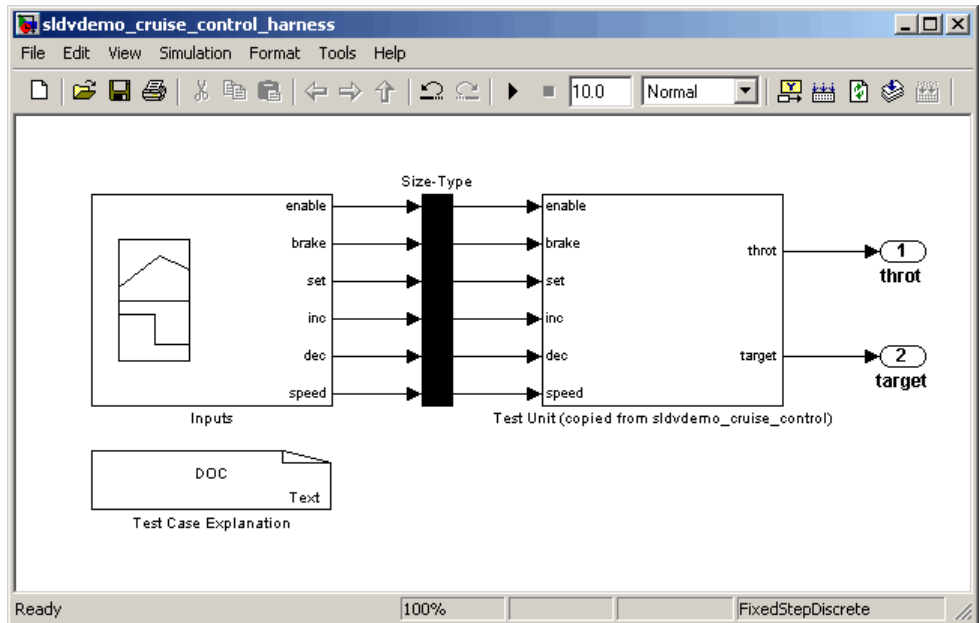
## Exploring Test Harness Models

Simulink Design Verifier generates a test harness model after it completes its analysis. If Simulink Design Verifier's **Mode** parameter specifies Test generation, the harness model contains test cases that achieve test objectives. Otherwise, Simulink Design Verifier's **Mode** parameter specifies Property proving and the harness model contains counterexamples that falsify proof objectives. The following sections describe the contents of a harness model and explain how to simulate it.

- “Anatomy of a Test Harness” on page 8-2
- “Simulating the Test Harness” on page 8-5

### Anatomy of a Test Harness

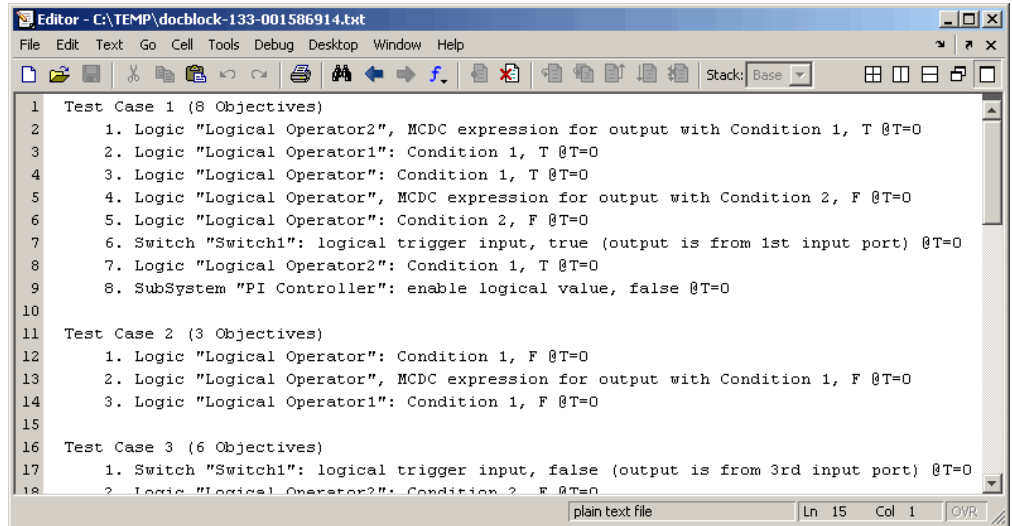
When Simulink Design Verifier completes its analysis, it produces a test harness model that looks like this:



The harness model contains the following items:



- **Test Case Explanation** — This DocBlock documents the test cases or counterexamples that Simulink Design Verifier 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 cases and their objectives. The text is as follows:

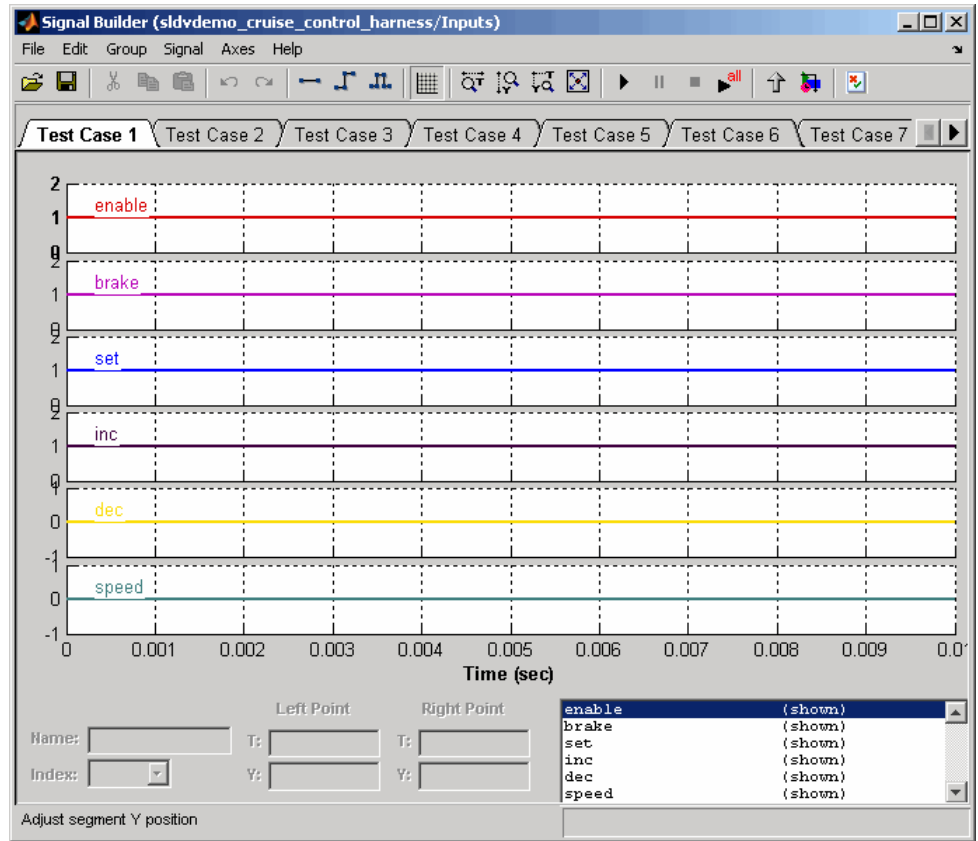
```

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 indicates "plain text file", "Ln 15", "Col 1", and "OVR".

- **Inputs** — This Signal Builder block contains signals that comprise the test cases or counterexamples that Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier generates. Using the test harness model, you can simulate

- A single test case or counterexample
- All test cases, for which Simulink collects model coverage information


To simulate a single test case or counterexample:

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

Simulink displays the Signal Builder dialog box.

- 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 .

Simulink simulates the test harness model using the signals associated with the selected test case or counterexample.

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

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

Simulink displays the Signal Builder dialog box.

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

Simulink simulates the test harness model using all test cases, 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

Simulink Design Verifier generates an HTML report that contains the following sections:

- “Front Matter” on page 8-6
- “Summary Chapter” on page 8-7
- “Block Replacements Summary Chapter” on page 8-12
- “Test/Proof Objectives Chapter” on page 8-12
- “Test Cases / Counterexamples Chapter” on page 8-17
- “Approximations Chapter” on page 8-20

### **Front Matter**

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

# Simulink Design Verifier Report

## sldvdemo\_flipflop

scowan

05-Apr-2007 14:02:39

---

### Table of Contents

- [1. Summary](#)
- [2. Test/Proof Objectives](#)
  - [Status](#)
  - [sldvdemo\\_flipflop](#)
  - [D Flip-Flop](#)
- [3. Test Cases / Counterexamples](#)
  - [Test Case 1](#)
  - [Test Case 2](#)
  - [Test Case 3](#)
  - [Test Case 4](#)
  - [Test Case 5](#)
- [4. Approximations](#)

### List of Tables

- 2.1. [Objectives Satisfied](#)

The title section lists the following information:

- Model or subsystem name that Simulink Design Verifier analyzed
- User name associated with the current MATLAB session
- Date and time that Simulink Design Verifier 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 8-8

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

### Input Model

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

<b>Input Model</b>	
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 Simulink Design Verifier 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 Simulink Design Verifier generates test cases for a model:

**Analysis Information**

Design Verifier Version:	1.0
Total Analysis Time:	1.57 secs
Status:	Completed normally
<a href="#">Approximations:</a>	1
<a href="#">Objectives Satisfied:</a>	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 Simulink Design Verifier
- Total time Simulink Design Verifier spent to complete its analysis
- Completion status of the Simulink Design Verifier analysis
- Total number of different approximation schemes Simulink Design Verifier used in its analysis (see “Approximations Chapter” on page 8-20)
- Total number of test or proof objectives for which Simulink Design Verifier was unable to decide an outcome
- Total number of test or proof objectives that produced errors

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

- Total number of test objectives that Simulink Design Verifier satisfied
- Total number of test objectives that Simulink Design Verifier satisfied without generating test cases
- Total number of test objectives that Simulink Design Verifier determined to be unsatisfiable

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

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

See “Test/Proof Objectives Chapter” on page 8-12 for more information related to the status of test and proof objectives.

### Output Files

The Output Files section provides information about the artifacts Simulink Design Verifier 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 8-21)
- Path and file name of the Simulink Design Verifier report

### Options

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



**Options.**

<b>Parameter</b>	<b>Setting</b>
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 Simulink Design Verifier executed. It appears only if Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier 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 8-13
- “Model Hierarchy” on page 8-16

## 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 Simulink Design Verifier was unable to determine an outcome in the allotted time. In this case, either Simulink Design Verifier 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
<a href="#">3</a>	Decision	<a href="#">Switch</a>	Switch "Switch", trigger >= threshold, F

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

**Table 2.4. Objectives Producing Errors**

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

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

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

**Table 2.1. Objectives Proven Unsatisfiable**

#:	Type	Model Item	Description
<a href="#">9</a>	Custom Test Objective	<a href="#">Test Objective</a>	Test Objective "Test Objective":1

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

**Table 2.2. Objectives Satisfied**

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

- **Objectives Satisfied - No Test Case** — Lists test objectives that Simulink Design Verifier satisfied without generating test cases. In this case, you might have specified a test objective on a signal whose value Simulink Design Verifier cannot control; or Simulink Design Verifier 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
<a href="#">1</a>	Custom Test Objective	<a href="#">Test Objective</a>	Test Objective "Test Objective": 0.9
<a href="#">2</a>	Custom Test Objective	<a href="#">Test Objective</a>	Test Objective "Test Objective": 1.1

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

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

Table 2.1. Objectives Proven Valid

#:	Type	Model Item	Description
<a href="#">1</a>	Custom Proof Objective	<a href="#">Proof Objective1</a>	Proof Objective "Proof Objective" : 1

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

Table 2.1. Objectives Falsified with Counterexamples

#:	Type	Model Item	Description
<a href="#">2</a>	Custom Proof Objective	<a href="#">Proof Objective</a>	Proof Objective "Proof Objective" : 1

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

Table 2.1. Objectives Falsified - No Counterexample

#:	Type	Model Item	Description
<a href="#">1</a>	Custom Proof Objective	<a href="#">Proof Objective</a>	Proof Objective "Proof Objective" : 0.9
<a href="#">2</a>	Custom Proof Objective	<a href="#">Proof Objective</a>	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 Simulink Design Verifier 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	<a href="#">TC 1</a>	input < 0, F
2	Satisfied	<a href="#">TC 3</a>	input < 0, T

Objectives of: Switch

#:	Status	Test Cases	Description
3	Undecidable	n/a	trigger >= threshold, F
4	Satisfied	<a href="#">TC 1</a>	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	<a href="#">TC 2</a>	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 of proof objective

## Test Cases / Counterexamples Chapter

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

- “Test Cases” on page 8-17
- “Counterexamples” on page 8-19

### Test Cases

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

## Test Case 1

### Summary

Length: 0.2 Seconds (3 sample periods)

Objective Count: 4

### Objectives Reached At:

Step	Time	Objectives
2	0.1	<a href="#">1</a> <a href="#">4</a> <a href="#">5</a> <a href="#">10</a>

### 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.

---



## Counterexamples

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

---

## Approximations Chapter

The Approximations chapter provides an overview of the approximations that Simulink Design Verifier 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 Simulink Design Verifier used during its analysis of the model.

---

**Note** Review the analysis results carefully when Simulink Design Verifier 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 Simulink Design Verifier 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

Simulink Design Verifier 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 Simulink Design Verifier gathers and produces during its analysis of a model. Although Simulink Design Verifier 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.

The following sections describe the contents of the `sldvData` structure and explain a method for simulating your model using a data file.

- “Anatomy of the `sldvData` Structure” on page 8-21
- “Simulating Models with Simulink Design Verifier Data Files” on page 8-25

### Anatomy of the `sldvData` Structure

When Simulink Design Verifier 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).

Simulink Design Verifier 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')
```

MATLAB 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`.

MATLAB displays the following field names that constitute the structure:

```
sldvData =

    ModelObjects: [1x2 struct]
    Objectives: [1x12 struct]
    TestCases: [1x5 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:

- “ModelObjects Field” on page 8-22
- “Objectives Field” on page 8-23
- “TestCases Field” on page 8-23

### 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
<code>descr</code>	String specifying the full path to a model object, including objects in a Stateflow chart.
<code>s1Path</code>	String specifying the full path to a Simulink model object.
<code>sfObjType</code>	String specifying the type of a Stateflow object, e.g., S for state and T for transition.
<code>sfObjNum</code>	Integer representing the unique identifier of a Stateflow object.
<code>objectives</code>	Vector of integers representing the indices of objectives associated with a model object.
<code>handle</code>	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>
portdimensions	<p>Cell array of vectors specifying the dimensions of signals in a test case or counterexample.</p>

## 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 Simulink Design Verifier 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, “Functions — Alphabetical List” for more information.





# Analyzing Large Models and Improving Performance

---

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

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

- “How Simulink Design Verifier Works” on page A-2
- “Sources of Model Complexity in Simulink Design Verifier” 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
- “Special Strategies for Proving Properties of Larger Models” on page A-13

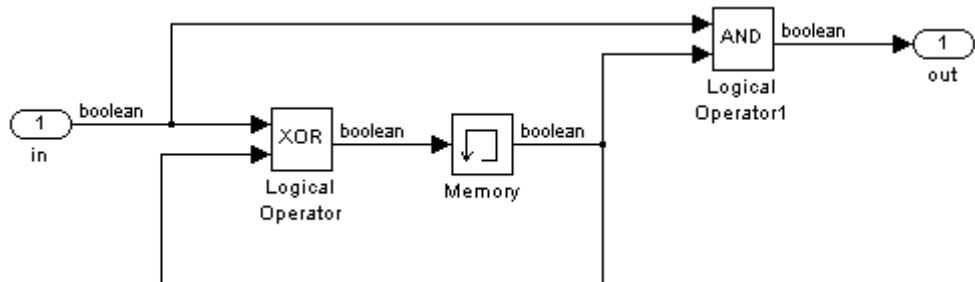
## How Simulink Design Verifier Works

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

Once Simulink Design Verifier 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 Simulink Design Verifier 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, Simulink Design Verifier 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.

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

## **Sources of Model Complexity in Simulink Design Verifier**

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 Simulink Design Verifier.

## 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, Simulink Design Verifier 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 Simulink Design Verifier, as occurs with multiplication or division unless one of the multiply operands or the divisor is a constant.

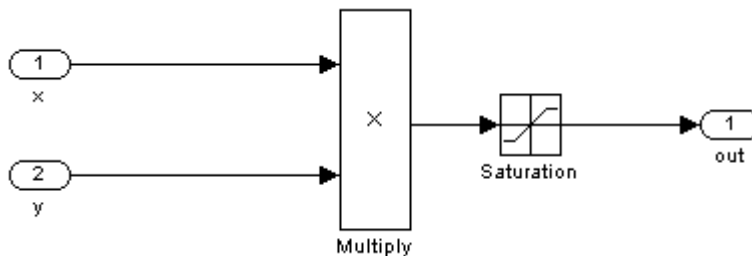
Input complexity can also result from certain cast operations. For example, casting a `double` to an `int8` can introduce a nonlinearity in certain situations.

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

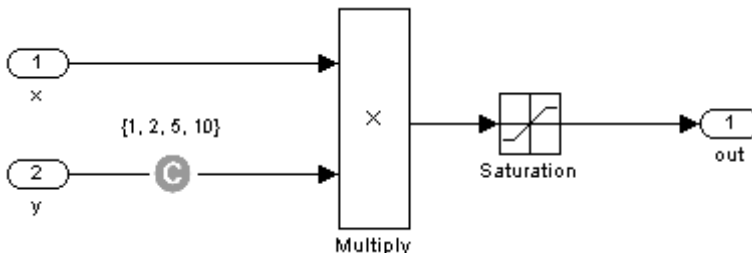
## Reducing Complexity from Floating-Point Operations and Nonlinear Arithmetic

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:

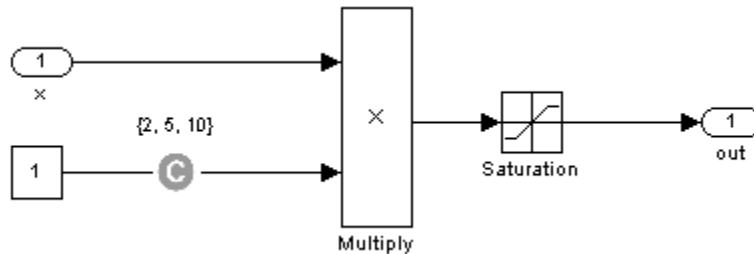


Simulink Design Verifier generates errors when attempting to satisfy the upper and lower limits of the Saturation block. You can work around these errors by restricting one of the inputs to be a range of values. For example, if you restrict the second input ( $y$ ) to be either 1, 2, 5, or 10, Simulink Design Verifier 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 with 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.



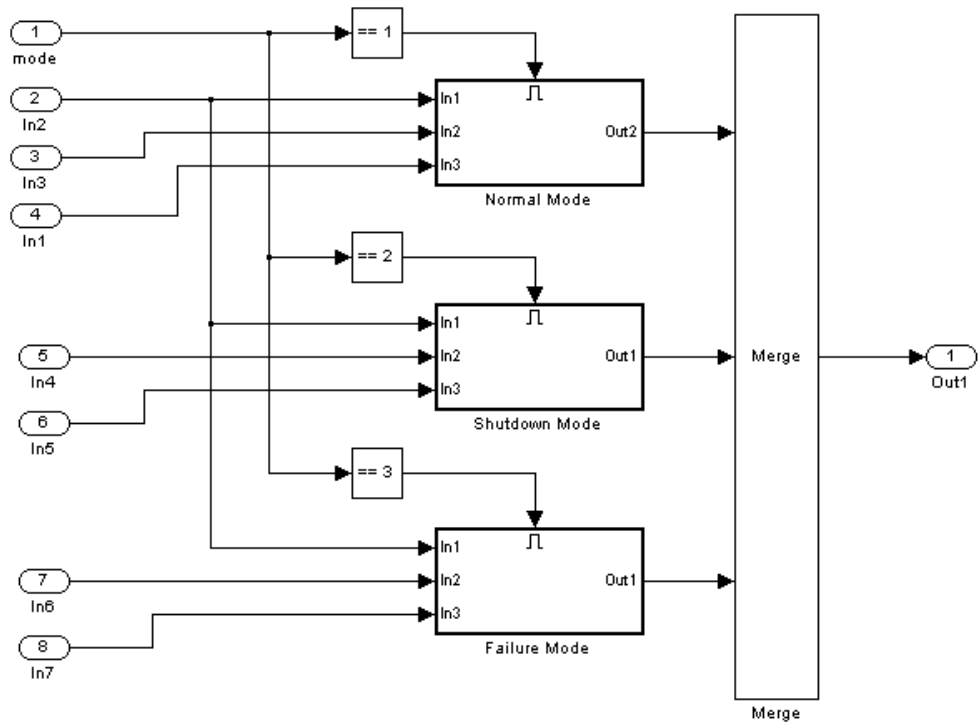
## Partitioning Inputs and Generating Tests Incrementally

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 Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier. 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 variables in the design 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. Simulink Design Verifier 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 Simulink Design Verifier 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 Simulink Design Verifier. 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.

## Special Strategies for Proving Properties of Larger 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 quickly proven, but otherwise a successful proof requires that Simulink Design Verifier search through all the reachable configurations of your model, even the ones that are only reached 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 within Simulink Design Verifier 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 Simulink Design Verifier 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 5-10 for more information.

An effective strategy for proving properties combines proving and violation searching to get the most benefit from Simulink Design Verifier 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, Simulink Design Verifier should complete the analysis 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, Simulink Design Verifier 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.

# Functions — Alphabetical List

---

# sldvblockreplacement

---

**Purpose** Replace model blocks to support Simulink Design Verifier

**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 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 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`.

**Synopsis** `sldvoptions`



**Purpose** Check model for compatibility with Simulink Design Verifier

**Syntax**

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

**Description** `status = sldvcompat(model)` returns 1 if `model` is compatible with Simulink Design Verifier and 0 otherwise. When checking for compatibility, Simulink Design Verifier 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 Simulink Design Verifier. The function destroys the temporary model after the compatibility check.

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

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

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

Simulink Design Verifier 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 Simulink Design Verifier:

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

Simulink Design Verifier 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** Run Simulink Design Verifier to extract subsystem contents into new model

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

**Description** `[status, modelH] = sldvextract(blockH)` runs the Simulink Design Verifier on the model that contains the subsystem `blockH` and extracts the contents of `blockH` into a new model. It returns the handle of the new model in `modelH`. The `sldvextract` function returns 1 upon successful completion and 0 otherwise.

# sldvgencov

---

**Purpose** Run Simulink Design Verifier to obtain missing model coverage

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

**Description** `[status, cvdo] = sldvgencov(model, options, startcov)` runs Simulink Design Verifier on the specified model using the `sldvoptions` object specified by `options`. Simulink Design Verifier ignores all model coverage objects that are satisfied in the `cvdata` object specified by `startcov`. It returns 1 for `status` if Simulink Design Verifier 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 Simulink Design Verifier, 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 a 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 <b>Assertion blocks</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}
BlockReplacement	Set by the <b>Apply block replacements</b> option on the <b>Design Verifier &gt; Block Replacements</b> pane of the Configuration Parameters dialog box.	'on'   {'off'}

Parameter	Description	Values
BlockReplacementModel-FileName	Set by the <b>File path of the output model</b> option on the <b>Design Verifier &gt; Block Replacements</b> pane of the Configuration Parameters dialog box.	string {'\$modelName\$_replacement'}
BlockReplacementRules-List	Set by the <b>List of block replacement rules</b> option on the <b>Design Verifier &gt; Block Replacements</b> pane of the Configuration Parameters dialog box.	string {'<FactoryDefaultRules>'}
DataFileName	Set by the <b>Data file name</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration Parameters dialog box.	string {'\$modelName\$_sldvdata'}
DisplayReport	Set by the <b>Display report</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'
DisplayUnsatisfiable-Objectives	Set by the <b>Display unsatisfiable test objectives</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'

# sldvoptions

Parameter	Description	Values
HarnessModelFileName	Set by the <b>Harness model file name</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration	string { '\$ModelName\$_harness' }
MakeOutputFilesUnique	Set by the <b>Make output file names unique by adding a suffix</b> check box on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	{ 'on' }   'off'
MaxProcessTime	Set by the <b>Maximum analysis time</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	double { '600' }
MaxTestCaseSteps	Set by the <b>Maximum test case steps</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	int32 { '500' }
MaxViolationSteps	Set by the <b>Maximum violation steps</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	int32 { '20' }
Mode	Set by the <b>Mode</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	{ 'TestGeneration' }   'PropertyProving'



Parameter	Description	Values
ModelCoverageObjectives	Set by the <b>Model coverage objectives</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	'None'   'Decision'   'ConditionDecision'   {'MCDC'}
OutputDir	Set by the <b>Output directory</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	string {'sldv_output/\$modelName\$'}
Parameters	Set by the <b>Apply parameters</b> option on the <b>Design Verifier &gt; Parameters</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'
ParametersConfigFileName	Set by the <b>Parameter configuration file</b> option on the <b>Design Verifier &gt; Parameters</b> pane of the Configuration Parameters dialog box.	string {'sldv_params_template.m'}
ProofAssumptions	Set by the <b>Proof assumptions</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}

## sldvoptions

Parameter	Description	Values
ProvingStrategy	Set by the <b>Strategy</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	'FindViolation'   {'Prove'}   'ProveWithViolationDetection'
ReportFileName	Set by the <b>Report file name</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	string {'\$ModelName\$_report'}
ReportIncludeGraphics	Set by the <b>Include screen shots and plots</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	'on'   {'off'}
SaveDataFile	Set by the <b>Save test data to file</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'
SaveHarnessModel	Set by the <b>Save test harness as model</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'

Parameter	Description	Values
SaveReport	Set by the <b>Generate report of the results</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	{'on'}   {'off'}
TestConditions	Set by the <b>Test conditions</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}
TestObjectives	Set by the <b>Test objectives</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}
TestSuiteOptimization	Set by the <b>Test suite optimization</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	{'CombinedObjectives'}   'IndividualObjectives'

**See Also**

sldvblockreplacement, sldvcompat, sldvgencov, sldvrun

**Purpose** Run Simulink Design Verifier 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 Simulink Design Verifier on the specified model. Simulink Design Verifier uses the configuration settings associated with `model` (if available); otherwise, Simulink Design Verifier 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 Simulink Design Verifier on the new model. Simulink Design Verifier uses the configuration settings associated with the parent model of `block` (if available); otherwise, Simulink Design Verifier uses its default configuration settings.

`status = sldvrun(model, options)` runs Simulink Design Verifier on the model specified by `model`. Simulink Design Verifier uses the `sldvoptions` object specified by `options`.

`[status, filenames] = sldvrun(model, options)` runs Simulink Design Verifier on the model specified by `model`. This function returns `status` and `filenames`, a structure whose fields list the names of the files that Simulink Design Verifier generates:

- `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.

**See Also**

`sldvcompat`, `sldvgencov`, `sldvoptions`

# sldvruntime

---

**Purpose** Simulate model using test case in Simulink Design Verifier data file

**Syntax**

```
data = sldvruntime(model, sldvDataFile, testIdx)
data = sldvruntime(model, sldvDataFile)
[data, cvdo] = sldvruntime(model, sldvDataFile, testIdx,
    true)
[data, cvdo] = sldvruntime(model, sldvDataFile, [], true)
```

**Description** `data = sldvruntime(model, sldvDataFile, testIdx)` simulates `model` using input signals associated with a single test case that Simulink Design Verifier 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 = sldvruntime(model, sldvDataFile)` simulates `model` using all test cases that the MAT-file, `sldvDataFile`, contains.

`[data, cvdo] = sldvruntime(model, sldvDataFile, testIdx, true)` simulates `model` using the test case that `testIdx` indexes in the MAT-file `sldvDataFile`. Simulink collects model coverage information during the simulation, which the function returns in the `cvdata` object `cvdo`.

`[data, cvdo] = sldvruntime(model, sldvDataFile, [], true)` simulates `model` using all test cases that the MAT-file, `sldvDataFile`, contains. Simulink 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*)

# Blocks — Alphabetical List

---

# Proof Assumption

---

**Purpose** Constrain signal values when proving model properties

**Library** Simulink Design Verifier

## Description



When operating in property proving mode, Simulink Design Verifier 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 Simulink Design Verifier 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** Simulink and Real-Time Workshop® ignore the Proof Assumption block during model simulation and code generation, respectively. Simulink Design Verifier 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:

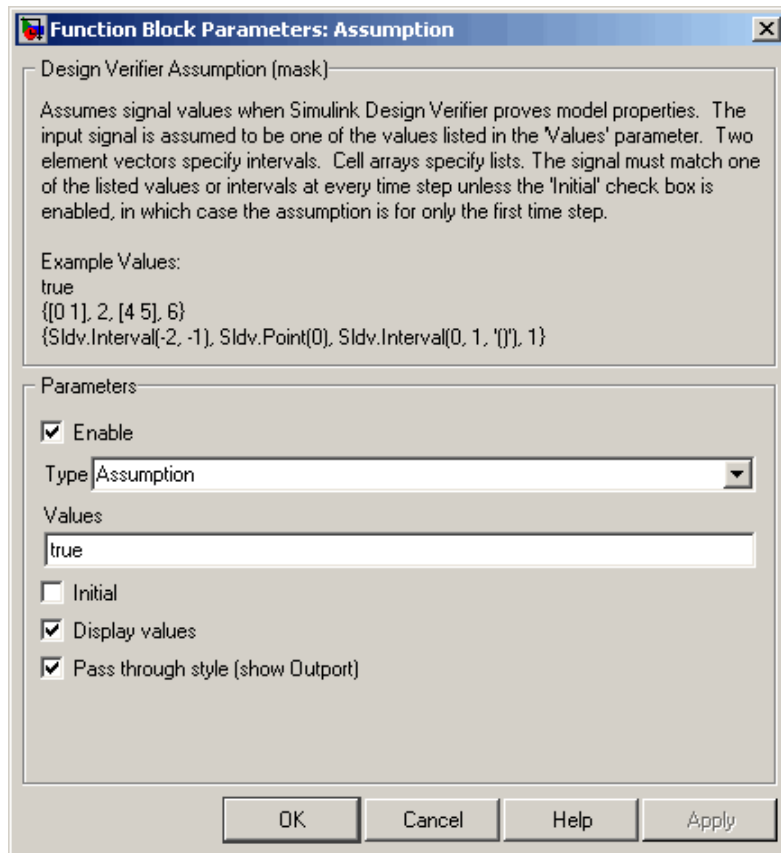
- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

If you specify multiple scalars and intervals for a Proof Assumption block, Simulink Design Verifier combines them using a logical OR operation during the property proof. In this case, Simulink Design Verifier 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 Simulink. For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in the Simulink documentation.

## Parameters and Dialog Box



### Enable

Specify whether the block is enabled. If selected (the default), Simulink Design Verifier uses the block when proving properties of a model. Clearing this option disables the block, that is, causes Simulink Design Verifier 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**

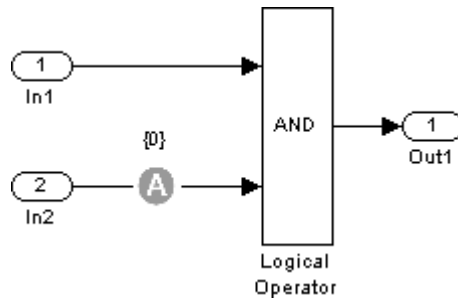
Specify whether the **Values** parameter applies at the beginning of or throughout the entire proof. If selected, the block constrains only the initial value of its input signal at the start of a proof analysis ( $t=0$ ). If not selected (the default), the block constrains its signal value for the entire proof.

## **Display values**

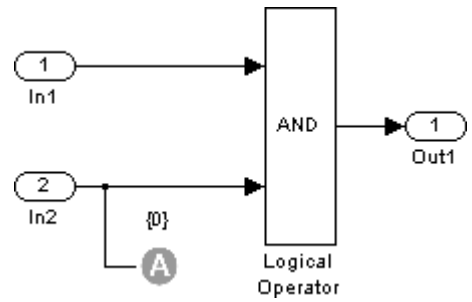
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

## **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, Simulink Design Verifier 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 Simulink Design Verifier 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** Simulink and Real-Time Workshop ignore the Proof Objective block during model simulation and code generation, respectively. Simulink Design Verifier 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

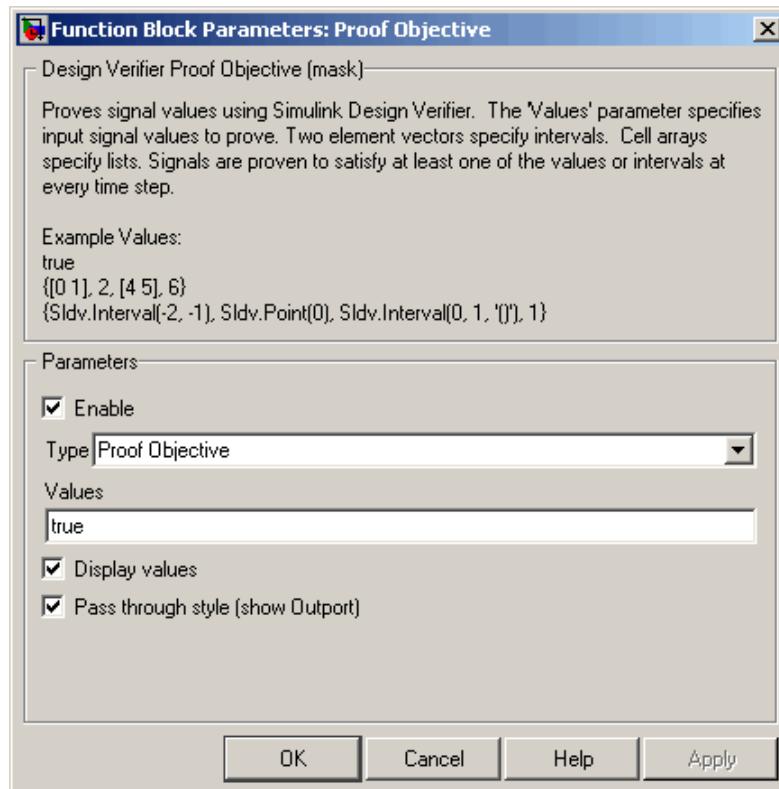
If you specify multiple scalars and intervals for a Proof Objective block, Simulink Design Verifier combines them using a logical OR operation during the property proof. In this case, Simulink Design Verifier considers the entire proof objective to be satisfied if any single scalar or interval is satisfied.

## Data Type Support

The Proof Objective block accepts signals of all built-in data types supported by Simulink. For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in the Simulink documentation.



## Parameters and Dialog Box



### Enable

Specify whether the block is enabled. If selected (the default), Simulink Design Verifier uses the block when proving properties of a model. Clearing this option disables the block, that is, causes Simulink Design Verifier 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

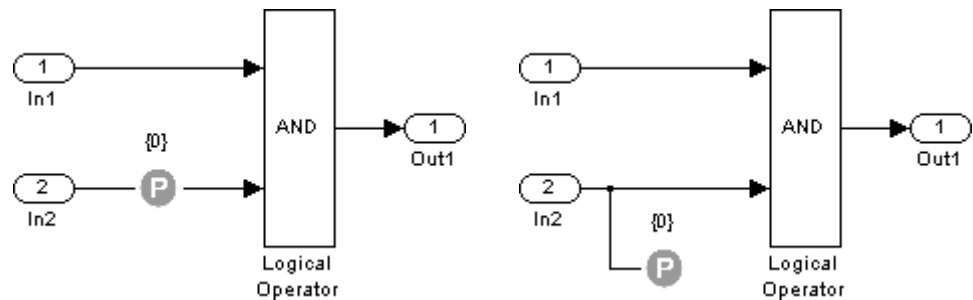
Specify the proof objective (see “Specifying Proof Objectives” on page 10-8).

## 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, Simulink Design Verifier 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 Simulink Design Verifier 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** Simulink and Real-Time Workshop ignore the Test Condition block during model simulation and code generation, respectively. Simulink Design Verifier 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

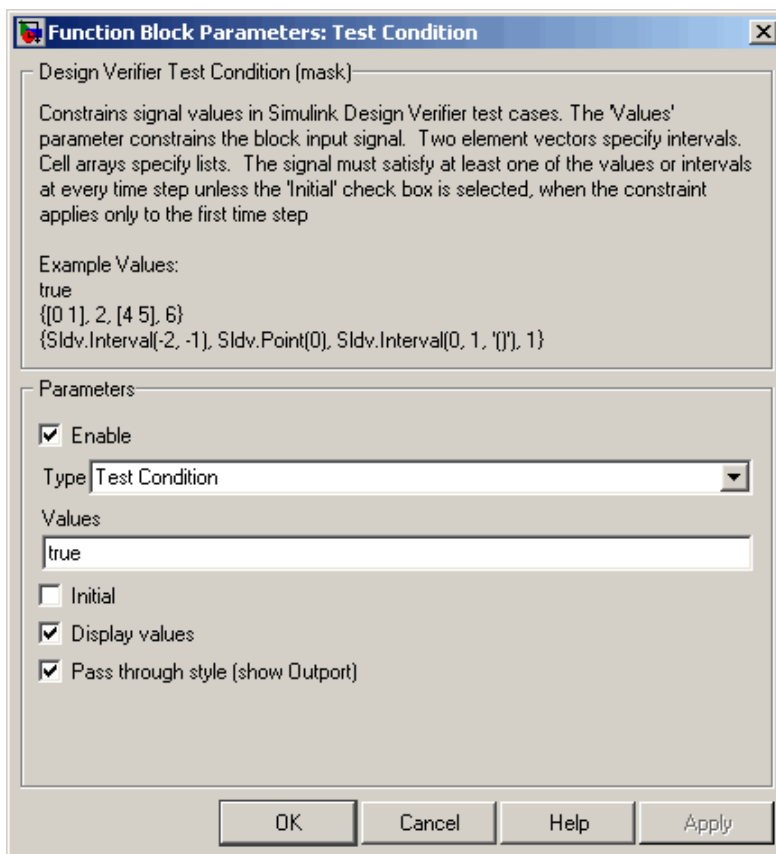
If you specify multiple scalars and intervals for a Test Condition block, Simulink Design Verifier combines them using a logical OR operation when generating test cases. Consequently, Simulink Design Verifier considers the entire test condition to be satisfied if any single scalar or interval is satisfied.

## Data Type Support

The Test Condition block accepts signals of all built-in data types supported by Simulink. For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in the Simulink documentation.

# Test Condition

## Parameters and Dialog Box



### Enable

Specify whether the block is enabled. If selected (the default), Simulink Design Verifier uses the block when generating tests for a model. Clearing this option disables the block, that is, causes Simulink Design Verifier 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**

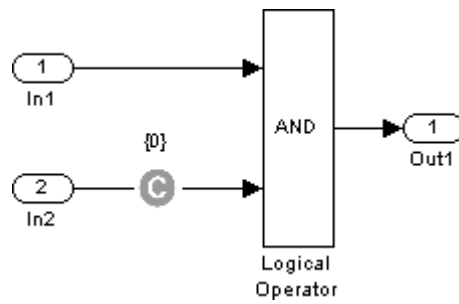
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

**Pass through style**

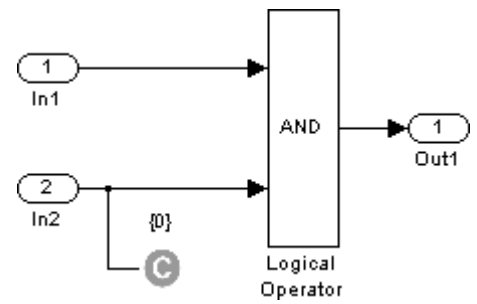
Specify whether the block displays an output port in the model editor. If selected (the default), the block displays its output port, allowing its input signal to pass through as the block output. If not selected, the block hides its output port and terminates the input signal. The following figure illustrates 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, Simulink Design Verifier 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 Simulink Design Verifier 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** Simulink and Real-Time Workshop ignore the Test Objective block during model simulation and code generation, respectively. Simulink Design Verifier 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:

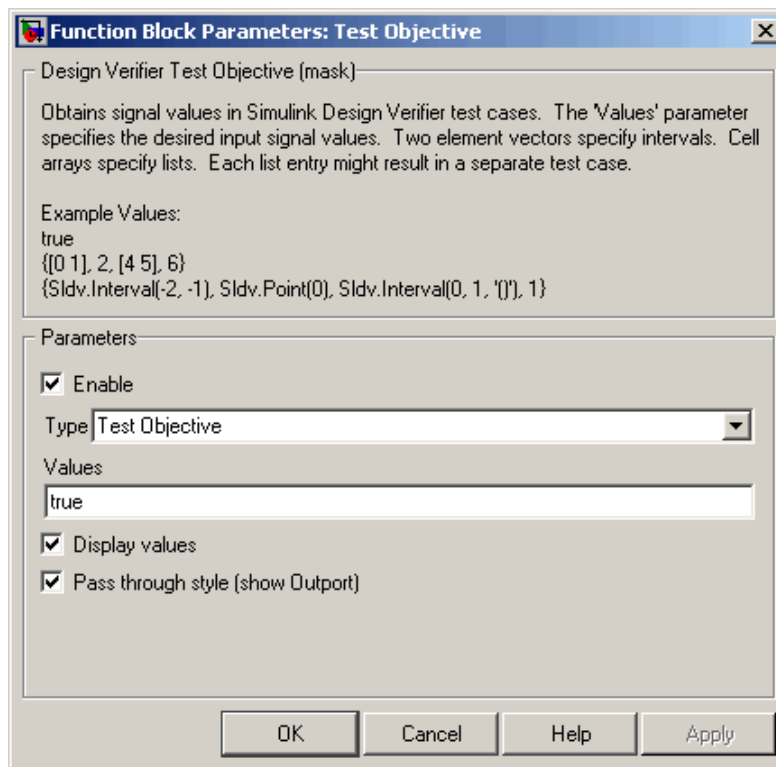
- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

## Data Type Support

The Test Objective block accepts signals of all built-in data types supported by Simulink. For a discussion on the data types supported by Simulink, see “Data Types Supported by Simulink” in the Simulink documentation.

# Test Objective

## Parameters and Dialog Box



### Enable

Specify whether the block is enabled. If selected (the default), Simulink Design Verifier uses the block when generating tests for a model. Clearing this option disables the block, that is, causes Simulink Design Verifier 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

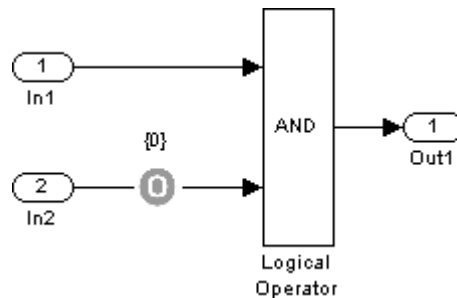
Specify the test objective (see “Specifying Test Objectives” on page 10-19).

## Display values

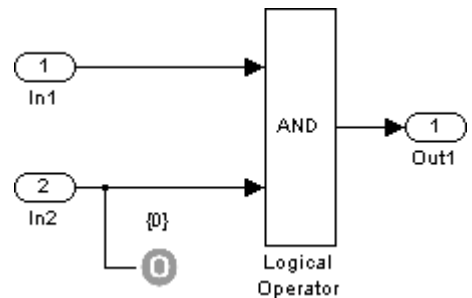
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

## 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 Constraint

# 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 Simulink Design Verifier. Real-Time Workshop 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, Simulink Design Verifier displays a warning.

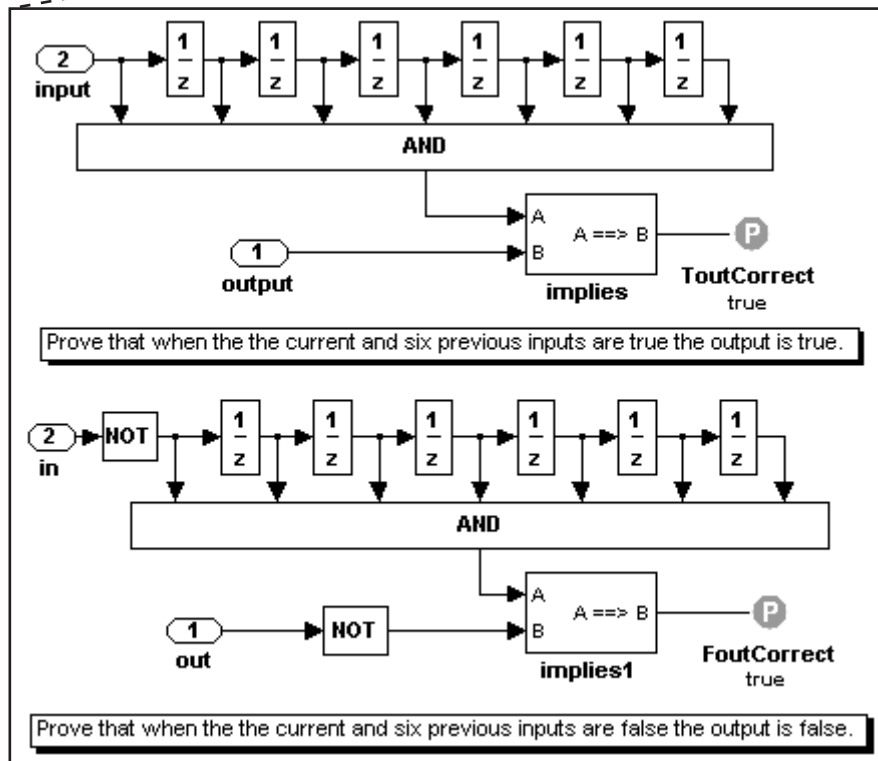
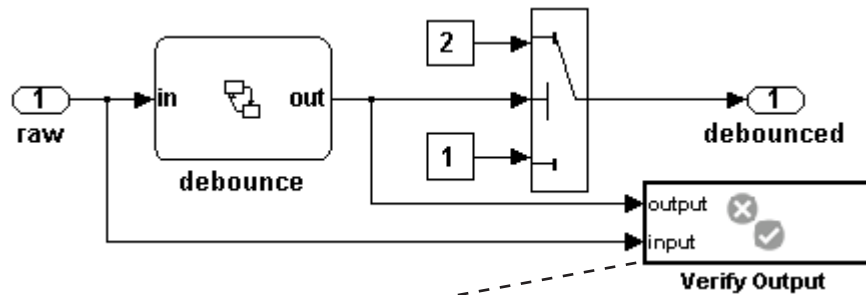
---

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 Constraint, Test Objective



# Configuration Parameters

---

The following table lists parameters that you can use to configure the behavior of Simulink Design Verifier. 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 <b>Assertion blocks</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}
DVBlockReplacement	Set by the <b>Apply block replacements</b> option on the <b>Design Verifier &gt; Block Replacements</b> pane of the Configuration Parameters dialog box.	'on'   {'off'}
DVBlockReplacementModel-FileName	Set by the <b>File path of the output model</b> option on the <b>Design Verifier &gt; Block Replacements</b> pane of the Configuration Parameters dialog box.	string {'\$modelName\$_replacement'}
DVBlockReplacementRules-List	Set by the <b>List of block replacement rules</b> option on the <b>Design Verifier &gt; Block Replacements</b> pane of the Configuration Parameters dialog box.	string {'<FactoryDefaultRules>'}

Parameter	Description	Values
DVDataFileName	Set by the <b>Data file name</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration Parameters dialog box.	string { '\$ModelName\$_sldvdata' }
DVDisplayReport	Set by the <b>Display report</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	{ 'on' }   'off'
DVDisplayUnsatisfiable-Objectives	Set by the <b>Display unsatisfiable test objectives</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	{ 'on' }   'off'
DVHarnessModelFileName	Set by the <b>Harness model file name</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration Parameters dialog box.	string { '\$ModelName\$_harness' }
DVMakeOutputFilesUnique	Set by the <b>Make output file names unique by adding a suffix</b> check box on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	{ 'on' }   'off'
DVMaxProcessTime	Set by the <b>Maximum analysis time</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	double { '600' }

Parameter	Description	Values
DVMaxTestCaseSteps	Set by the <b>Maximum test case steps</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	int32 {'500'}
DVMaxViolationSteps	Set by the <b>Maximum violation steps</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	int32 {'20'}
DVMode	Set by the <b>Mode</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	{'TestGeneration'}   'PropertyProving'
DVModelCoverageObjectives	Set by the <b>Model coverage objectives</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	'None'   'Decision'   'ConditionDecision'   {'MCDC'}
DVOutputDir	Set by the <b>Output directory</b> option on the <b>Design Verifier</b> pane of the Configuration Parameters dialog box.	string { 'sldv_output/\$ModelName\$' }
DVParameters	Set by the <b>Apply parameters</b> option on the <b>Design Verifier &gt; Parameters</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'

Parameter	Description	Values
DVParametersConfigFileName	Set by the <b>Parameter configuration file</b> option on the <b>Design Verifier &gt; Parameters</b> pane of the Configuration Parameters dialog box.	string {'sldv_params_template.m'}
DVProofAssumptions	Set by the <b>Proof assumptions</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}
DVProvingStrategy	Set by the <b>Strategy</b> option on the <b>Design Verifier &gt; Property Proving</b> pane of the Configuration Parameters dialog box.	'FindViolation'   {'Prove'}   'ProveWithViolationDetection'
DVReportFileName	Set by the <b>Report file name</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	string {'\$ModelName\$_report'}
DVReportIncludeGraphics	Set by the <b>Include screen shots and plots</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	'on'   {'off'}
DVSaveDataFile	Set by the <b>Save test data to file</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'

Parameter	Description	Values
DVSaveHarnessModel	Set by the <b>Save test harness as model</b> option on the <b>Design Verifier &gt; Results</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'
DVSaveReport	Set by the <b>Generate report of the results</b> option on the <b>Design Verifier &gt; Report</b> pane of the Configuration Parameters dialog box.	{'on'}   'off'
DVTestConditions	Set by the <b>Test conditions</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}
DVTestObjectives	Set by the <b>Test objectives</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	'EnableAll'   'DisableAll'   {'UseLocalSettings'}
DVTestSuiteOptimization	Set by the <b>Test suite optimization</b> option on the <b>Design Verifier &gt; Test Generation</b> pane of the Configuration Parameters dialog box.	{'CombinedObjectives'}   'IndividualObjectives'



# Simulink Block Support

---

The following table summarizes Simulink Design Verifier 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 Simulink Design Verifier. All support notes appear at the end of the table.

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
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	—

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
Continuous	Derivative	Not supported
	Integrator	Not supported
	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	Not supported
	Dead Zone	Not supported
	Dead Zone Dynamic	—
	Hit Crossing	—
	Quantizer	—
	Rate Limiter	—
	Rate Limiter Dynamic	—
	Relay	Not supported
	Saturation	—
	Saturation Dynamic	—
	Wrap To Zero	—

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
Discrete	Difference	—
	Discrete Derivative	—
	Discrete Filter	Not supported
	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	—
Weighted Moving Average	Not supported	
Zero-Order Hold	—	

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
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	Not supported	

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
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

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
Math Operations	Abs	—
	Add	—
	Algebraic Constraint	Not supported
	Assignment	—
	Bias	—
	Complex to Magnitude-Angle	—
	Complex to Real-Imag	—
	Divide	—
	Dot Product	Not supported
	Gain	—
	Magnitude-Angle to Complex	Not supported
	Math Function	SN3
	Matrix Concatenate	—
	MinMax	—
	MinMax Running Resettable	—
	Permute Dimensions	—
	Polynomial	—
	Product	—
	Product of Elements	—
	Real-Imag to Complex	Not supported
	Reshape	—
Rounding Function	—	
Sign	—	

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
Math Operations (continued)	Sine Wave Function	Not supported
	Slider Gain	—
	Squeeze	—
	Subtract	—
	Sum	—
	Sum of Elements	—
	Trigometric Function	Not supported
	Unary Minus	Not supported
	Vector Concatenate	—
	Weighted Sample Time Math	Not supported
Model Verification	Assertion	—
	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	—



<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
Ports & Subsystems	Atomic Subsystem	—
	Code Reuse Subsystem	—
	Configurable Subsystem	—
	Enabled Subsystem	—
	Enabled and Triggered Subsystem	—
	For Iterator Subsystem	—
	Function-Call Generator	—
	Function-Call Subsystem	SN7
	If	—
	If Action Subsystem	—
	Model	Not supported
	Subsystem	—
	Switch Case	—
	Switch Case Action Subsystem	—
	Triggered Subsystem	—
While Iterator Subsystem	—	

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
Signal Attributes	Bus to Vector	—
	Data Type Conversion	—
	Data Type Conversion Inherited	—
	Data Type Duplicate	—
	Data Type Propagation	—
	Data Type Scaling Strip	—
	IC	—
	Probe	Not supported
	Rate Transition	—
	Signal Conversion	—
	Signal Specification	—
	Weighted Sample Time	Not supported
	Width	Not supported

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
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	—	

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
Sinks	Display	—
	Floating Scope	—
	Outport (Out1)	—
	Scope	—
	Stop Simulation	Not supported
	Terminator	—
	To File	—
	To Workspace	—
	XY Graph	—

<b>Sublibrary</b>	<b>Block</b>	<b>Support Notes</b>
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)	—
	Pulse Generator	Not supported
	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	Not supported
	Fcn	SN4
	Level-2 M-file S-Function	Not supported
	MATLAB Fcn	Not supported
	S-Function	Not supported
	S-Function Builder	Not supported

Symbol	Note
—	Simulink Design Verifier supports the block and requires no special notes.
SN1	Input and output must have the same data type, either single or double.
SN2	Cannot specify either the <b>Interpolation method</b> or the <b>Extrapolation method</b> parameter as Cubic Spline.
SN3	Supports only mod and reciprocal options for <b>Function</b> parameter.
SN4	Supports all operators except ^, and supports only the mathematical functions abs, ceil, fabs, and floor.
SN5	Supports only <b>Number of table dimensions</b> that specify either 1 or 2.
SN6	Supports only virtual signal buses.
SN7	Supports only scalar signals that trigger execution of Function-Call Subsystems.

**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 Simulink Design Verifier and defines how instances of specific blocks will be replaced by an alternate implementation. Simulink Design Verifier 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). Simulink Design Verifier 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 3-14

## **Specifying Parameter Configurations**

- “Constructing the Example Model” on page 4-7
- “Parameterizing the Constant Block” on page 4-9
- “Specifying a Parameter Configuration” on page 4-11
- “Analyzing the Example Model” on page 4-12
- “Simulating the Test Cases” on page 4-14

## **Generating Test Cases**

- “Constructing the Example Model” on page 6-4
- “Checking Compatibility of the Example Model” on page 6-6
- “Configuring Test Generation Options” on page 6-9
- “Analyzing the Example Model” on page 6-12
- “Customizing Test Generation” on page 6-20
- “Reanalyzing the Example Model” on page 6-24

## **Proving Properties of a Model**

- “Constructing the Example Model” on page 7-5
- “Instrumenting the Example Model” on page 7-9
- “Configuring Property Proving Options” on page 7-12
- “Analyzing the Example Model” on page 7-14
- “Customizing the Example Proof” on page 7-22
- “Reanalyzing the Example Model” on page 7-24

## B

- block replacements
  - configuration 3-14
  - example 3-7
  - execution 3-15
  - factory defaults 3-3
  - introduction 3-2
  - template 3-6
- block support
  - limitations 2-2
  - summary 12-2

## C

- configuration parameters
  - block replacements 5-6
  - Design Verifier 5-5
  - parameters 5-8
  - property proving 5-10
  - report 5-14
  - results 5-12
  - summary 11-2
  - test generation 5-9

## M

- model compatibility
  - checking 2-5

## 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 11-2
  - running demo 1-6
  - workflow 1-17
- Simulink Design Verifier data files
  - anatomy 8-21
  - simulation 8-25
- Simulink Design Verifier options
  - saving 5-15
  - viewing 5-2
- Simulink Design Verifier report
  - table of contents 8-6
- Simulink Design Verifier reports
  - approximations 8-20
  - block replacements summary 8-12
  - summary 8-7
  - test cases/counterexamples 8-17
  - test/proof objectives 8-12
  - title 8-6
- 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 9-14
- sldvruntest function 9-16
- 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 8-5
- Test Objective block 10-19

## **U**

- unsupported features
  - Simulink 2-2
  - Stateflow 2-3

## **V**

- Verification Subsystem block 10-24