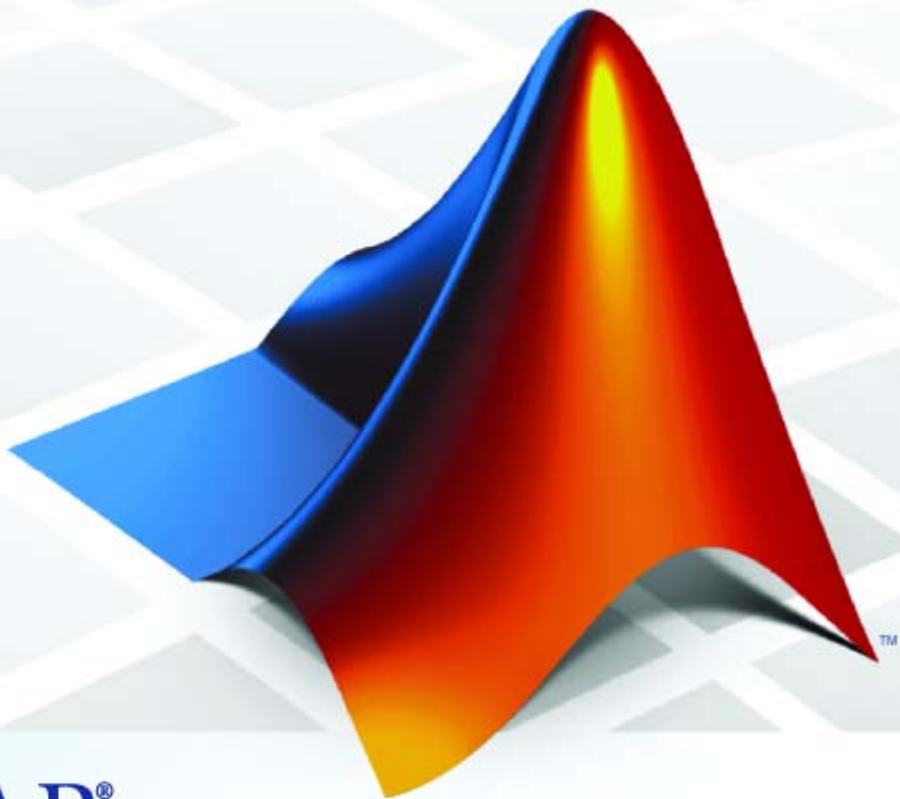


Simulink® Design Verifier™ 1

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 2007–2009 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

May 2007	Online only	New for Version 1.0 (Release 2007a+)
September 2007	Online only	Revised for Version 1.1 (Release 2007b)
March 2008	Online only	Revised for Version 1.2 (Release 2008a)
October 2008	Online only	Revised for Version 1.3 (Release 2008b)
March 2009	Online only	Revised for Version 1.4 (Release 2009a)
September 2009	Online only	Revised for Version 1.5 (Release 2009b)

Acknowledgment

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



Acknowledgment

Getting Started

1

Product Overview	1-2
Before You Begin	1-3
What You Need to Know	1-3
Required Products	1-3
Starting the Simulink® Design Verifier Software	1-4
Analyzing a Model	1-6
About This Demo	1-6
Opening the Model	1-6
Generating Test Cases	1-7
Combining Test Cases	1-23
Analyzing a Subsystem	1-26
Basic Workflow for Using the Simulink® Design Verifier Software	1-30
Learning More	1-31
Next Step	1-31
Product Help	1-31
The MathWorks Online	1-32

How the Simulink® Design Verifier Software Works

2

Model Analysis with Simulink® Design Verifier Software	2-2
Analyzing a Simple Model	2-3
Analyzing Large Models	2-5
Handling Incompatibilities with Automatic Stubbing	2-6
Stubbing	2-6
What Is Automatic Stubbing?	2-6
Analyzing a Model Using Automatic Stubbing	2-6
Approximations	2-14
Approximations During Model Analysis	2-14
Types of Approximations	2-14
Converting Floating-Point Arithmetic to Rational-Number Arithmetic	2-14
Linearizing 2-D Lookup Tables	2-15
Unrolling While Loops	2-15
Ensuring the Validity of the Analysis	2-15

Ensuring Compatibility with the Simulink® Design Verifier Software

3

Checking Model Compatibility	3-2
Model Is Compatible	3-4
Model Is Incompatible	3-4
Some Model Elements Are Incompatible	3-6
Unsupported Simulink Software Features	3-9
Simulink Software Features Not Supported	3-9
Simulink Block Support Limitations	3-11

Limitations of Support for Model Reference	3-11
Unsupported Stateflow Software Features	3-13
Support Limitations for the Embedded MATLAB	
Subset	3-15
Unsupported Embedded MATLAB Subset Features	3-15
Limitations of Embedded MATLAB Library Function	
Support	3-16
Fixed-Point Support Limitations	3-18

Working with Block Replacements

4

About Block Replacements	4-2
Built-In Block Replacements	4-3
Template for Block Replacement Rules	4-6
Defining Custom Block Replacements	4-7
Basic Workflow for Defining Custom Block	
Replacements	4-7
Specifying Replacement Blocks	4-7
Writing Block Replacement Rules	4-8
Example: Replacing Multiport Switch Blocks	4-8
Executing Block Replacements	4-16
Configuring Block Replacements	4-16
Replacing Blocks in a Model	4-17

Specifying Parameter Configurations

5

About Parameter Configurations	5-2
Template for Parameter Configurations	5-3
Defining Parameter Configurations	5-4
Parameter Configuration Example	5-7
About This Example	5-7
Constructing the Example Model	5-8
Parameterizing the Constant Block	5-10
Specifying a Parameter Configuration	5-11
Analyzing the Example Model	5-13
Simulating the Test Cases	5-15

Configuring Simulink® Design Verifier Options

6

Viewing Simulink® Design Verifier Options	6-2
Configuring Simulink® Design Verifier Options	6-5
Design Verifier Pane	6-5
Block Replacements Pane	6-7
Parameters Pane	6-9
Test Generation Pane	6-10
Property Proving Pane	6-12
Results Pane	6-14
Report Pane	6-17
Saving Simulink® Design Verifier Options	6-20

7

About Test Case Generation	7-2
Test Case Blocks	7-2
Test Case Functions	7-2
Workflow for Generating Test Cases	7-4
Generating Test Cases for a Model	7-5
About This Example	7-5
Constructing the Example Model	7-6
Checking Compatibility of the Example Model	7-7
Configuring Test Generation Options	7-11
Analyzing the Example Model	7-14
Customizing Test Generation	7-22
Reanalyzing the Example Model	7-26
Analyzing Contradictory Models	7-30
Generating Test Cases for a Subsystem	7-31

Proving Properties of a Model

8

About Property Proving	8-2
Proof Blocks	8-2
Proof Functions	8-2
Workflow for Proving Model Properties	8-4
Proving Properties in a Model	8-5
About This Example	8-5
Constructing the Example Model	8-6
Checking Compatibility of the Example Model	8-7
Instrumenting the Example Model	8-11
Configuring Property-Proving Options	8-14
Analyzing the Example Model	8-16

Customizing the Example Proof	8-22
Reanalyzing the Example Model	8-25
Analyzing Contradictory Models	8-26
Proving Properties in a Subsystem	8-28
Proving Complex Properties	8-29
Property-Proving Examples	8-29

Reviewing the Results

9

Examining Simulink® Design Verifier Data Files	9-2
About Simulink® Design Verifier Data Files	9-2
Overview of the sldvData Structure	9-2
Model Information Fields in sldvData	9-3
Simulating Models with Simulink® Design Verifier Data Files	9-7
Exploring Test Harness Models	9-8
About Test Harness Models	9-8
Anatomy of a Test Harness	9-8
Configuration of the Test Harness	9-13
Simulating the Test Harness	9-13
Creating a SystemTest TEST-File	9-15
Understanding Simulink® Design Verifier Reports	9-18
About Simulink® Design Verifier Reports	9-18
Front Matter	9-18
Summary Chapter	9-19
Analysis Information Chapter	9-20
Test / Proof Objectives Status Chapter	9-25
Model Items Chapter	9-29
Test Cases / Properties Chapter	9-30

Analyzing Large Models and Improving Performance

10

Sources of Model Complexity	10-2
Analyzing a Large Model	10-3
Types of Large Model Problems	10-3
Using the Default Parameter Values	10-4
Modifying the Analysis Parameters	10-5
Using the Large Model Optimization	10-6
Stopping the Analysis Before Completion	10-6
Generating Reports for Large Models	10-8
Managing Model Data to Simplify the Analysis	10-9
Simplifying Data Types	10-9
Constraining Data	10-9
Partitioning Model Inputs and Generating Tests	
Incrementally	10-13
Analyzing the Model Using a Bottom-Up Approach ...	10-15
Analyzing Logical Operations	10-16
Handling Models with Large State Spaces	10-17
Handling Problems with Counters and Timers	10-18
Techniques for Proving Properties of Large Models ..	10-20

Function Reference

11

Block Reference

12

Configuration Parameters

13

Design Verifier Pane	13-2
Design Verifier Pane Overview	13-3
Mode	13-3
Maximum analysis time	13-5
Display unsatisfiable test objectives	13-6
Automatic stubbing of unsupported blocks and functions ..	13-7
Output directory	13-8
Make output file names unique by adding a suffix	13-9
Design Verifier Pane: Block Replacements	13-10
Block Replacements Pane Overview	13-11
Apply block replacements	13-12
List of block replacement rules	13-13
File path of the output model	13-14
Design Verifier Pane: Parameters	13-15
Parameters Pane Overview	13-16
Apply parameters	13-16
Parameter configuration file	13-16
Design Verifier Pane: Test Generation	13-18
Test Generation Pane Overview	13-19
Model coverage objectives	13-20
Test conditions	13-21
Test objectives	13-22
Maximum test case steps	13-23
Test suite optimization	13-24

Design Verifier Pane: Property Proving	13-26
Property Proving Pane Overview	13-27
Assertion blocks	13-28
Proof assumptions	13-29
Strategy	13-30
Maximum violation steps	13-31
Design Verifier Pane: Results	13-32
Results Pane Overview	13-34
Save test data to file	13-35
Data file name	13-36
Include expected output values	13-37
Randomize data that does not affect outcome	13-39
Save test harness as model	13-41
Harness model file name	13-42
Reference input model in generated harness	13-43
Save test harness as SystemTest TEST-file (will reference saved data file)	13-44
SystemTest file name	13-45
Design Verifier Pane: Report	13-46
Report Pane Overview	13-47
Generate report of the results	13-48
Report file name	13-49
Include screen shots of properties and text objectives	13-50
Display report	13-51
Parameter Command-Line Information Summary	13-52

Simulink Block Support

14

Overview of Simulink Block Support	14-2
Additional Math and Discrete Library	14-3
Commonly Used Blocks Library	14-4

Continuous Library	14-5
Discontinuities Library	14-6
Discrete Library	14-7
Logic and Bit Operations Library	14-8
Lookup Tables Library	14-9
Math Operations Library	14-10
Model Verification Library	14-12
Model-Wide Utilities Library	14-13
Ports & Subsystems Library	14-14
Signal Attributes Library	14-15
Signal Routing Library	14-16
Sinks Library	14-17
Sources Library	14-18
User-Defined Functions Library	14-19

Glossary

Examples

A

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

Index

Getting Started

- “Product Overview” on page 1-2
- “Before You Begin” on page 1-3
- “Starting the Simulink® Design Verifier Software” on page 1-4
- “Analyzing a Model” on page 1-6
- “Analyzing a Subsystem” on page 1-26
- “Basic Workflow for Using the Simulink® Design Verifier Software” on page 1-30
- “Learning More” on page 1-31

Product Overview

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

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

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

Before You Begin

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

What You Need to Know

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

To learn more about these topics, see the following:

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

Required Products

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

- MATLAB®
- Simulink
- Simulink Verification and Validation

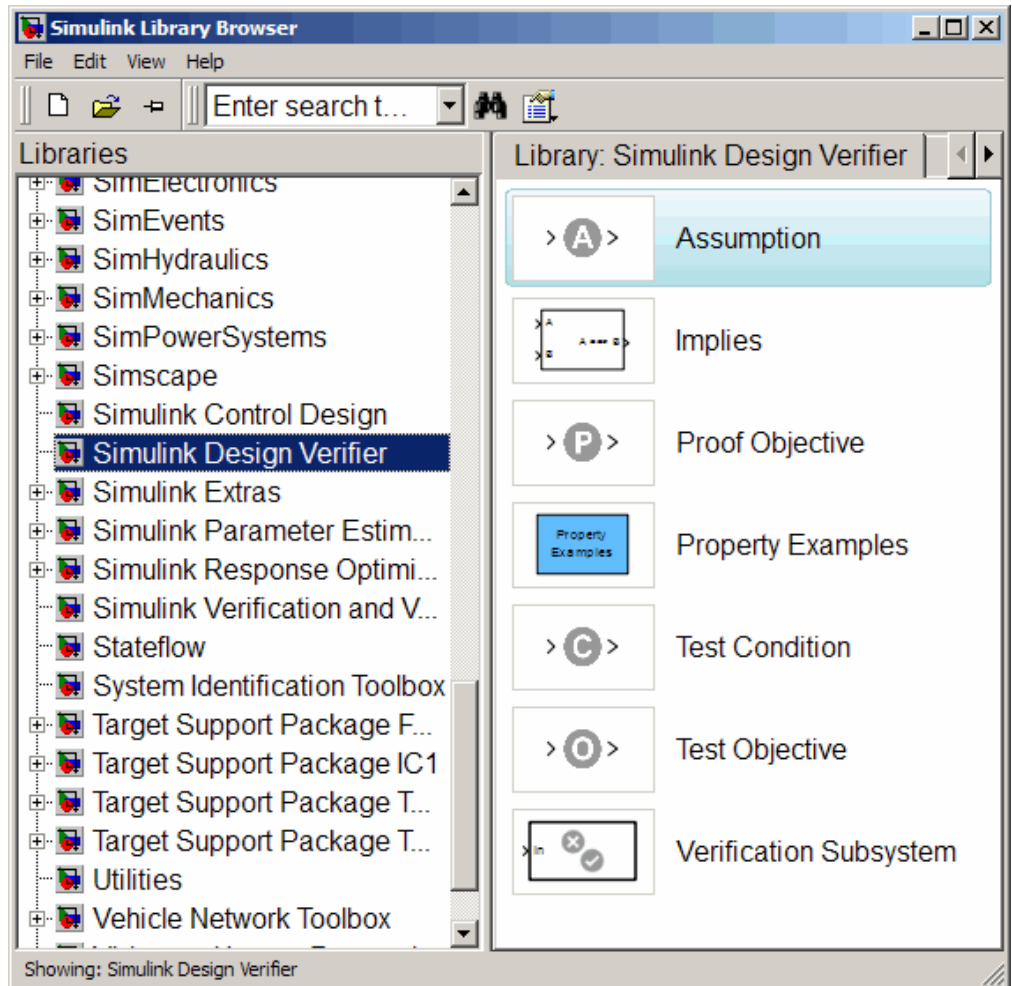
If you want to use the Simulink Design Verifier software with Stateflow® charts, you must have the following software product:

- Stateflow

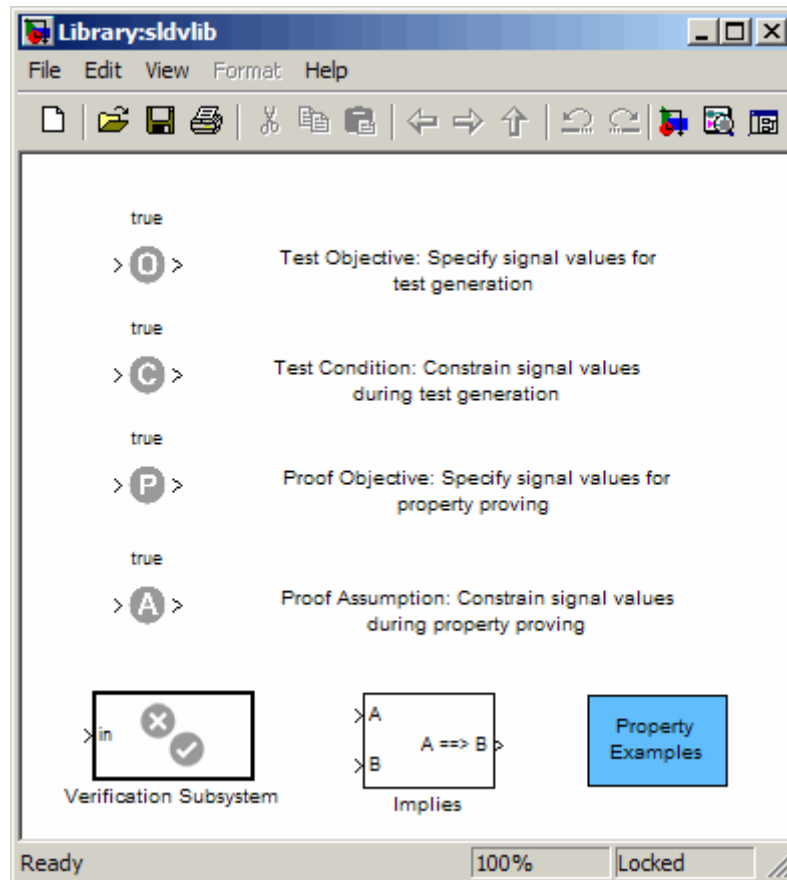
Starting the Simulink Design Verifier Software

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

To open the Simulink Design Verifier block library, type `simulink` at the MATLAB prompt to display the Simulink Library Browser, and then select the **Simulink Design Verifier** entry in the contents tree.



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



Analyzing a Model

In this section...
“About This Demo” on page 1-6
“Opening the Model” on page 1-6
“Generating Test Cases” on page 1-7
“Combining Test Cases” on page 1-23

About This Demo

The following sections describe a demo model, Cruise Control Test Generation. This demo illustrates how to use the Simulink Design Verifier software to generate test cases that achieve complete model coverage. Through this demo, you learn how to analyze models with the Simulink Design Verifier software and interpret the results.

Opening the Model

To open the Cruise Control Test Generation model, enter `sldvdemo_cruise_control` at the MATLAB prompt.

The Cruise Control Test Generation model opens.

slvdemo_cruise_control

File Edit View Simulation Format Tools Help

10.0 Normal

Simulink Design Verifier Cruise Control Test Generation

1 enable
2 brake
3 set [0 100]
6 speed
4 inc
5 dec

throt
target

Controller

This model is configured to generate test cases that achieve complete model coverage. By default Simulink Design Verifier generates test cases that satisfy objectives in the fewest steps. One of the test objectives forces the discrete integrator in the PI controller to exceed its upper limit. When you run Simulink Design Verifier without constraints the limit is exceeded in a single step by forcing speed to be 500. The constraint on speed limits the values in test cases between 0 and 100. This forces the test cases to take several samples to exceed the integrator limit.

Run (double-click)
Toggle Speed Constraint (double-click)
View Options (double-click)

Run Simulink Design Verifier Toggle Constraint View Simulink Design Verifier Options

Copyright 2006-2007 The MathWorks, Inc.

Ready 100% FixedStepDiscrete

Generating Test Cases

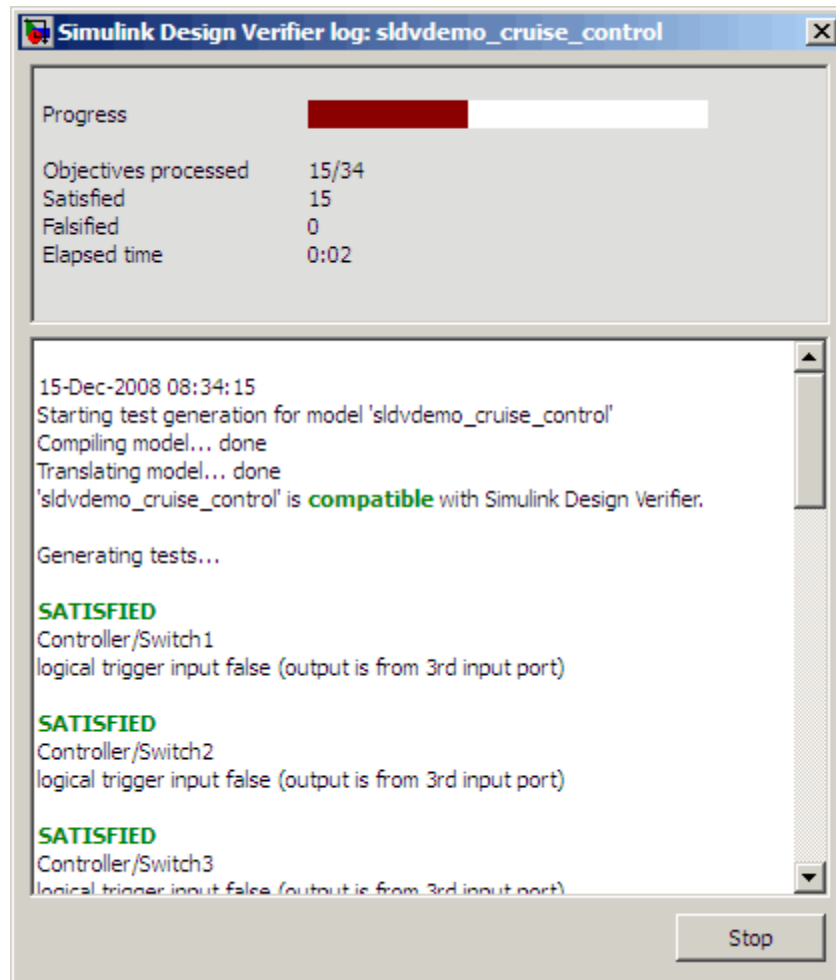
- “Running the Analysis” on page 1-8

- “Exploring the Test Harness” on page 1-10
- “Interpreting the Simulink® Design Verifier HTML Report” on page 1-15

Running the Analysis

To generate test cases for the Cruise Control Test Generation model, open the model window and double-click the block labeled **Run**.

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



The log window updates you on the progress of the Simulink Design Verifier software as it analyzes the model.

Note If you need to terminate an analysis while it is running, click **Stop**. The software asks you if you want to produce results. If you click **Yes**, the software creates the data file and report based on the results achieved so far. The names of those files appear in the log window.

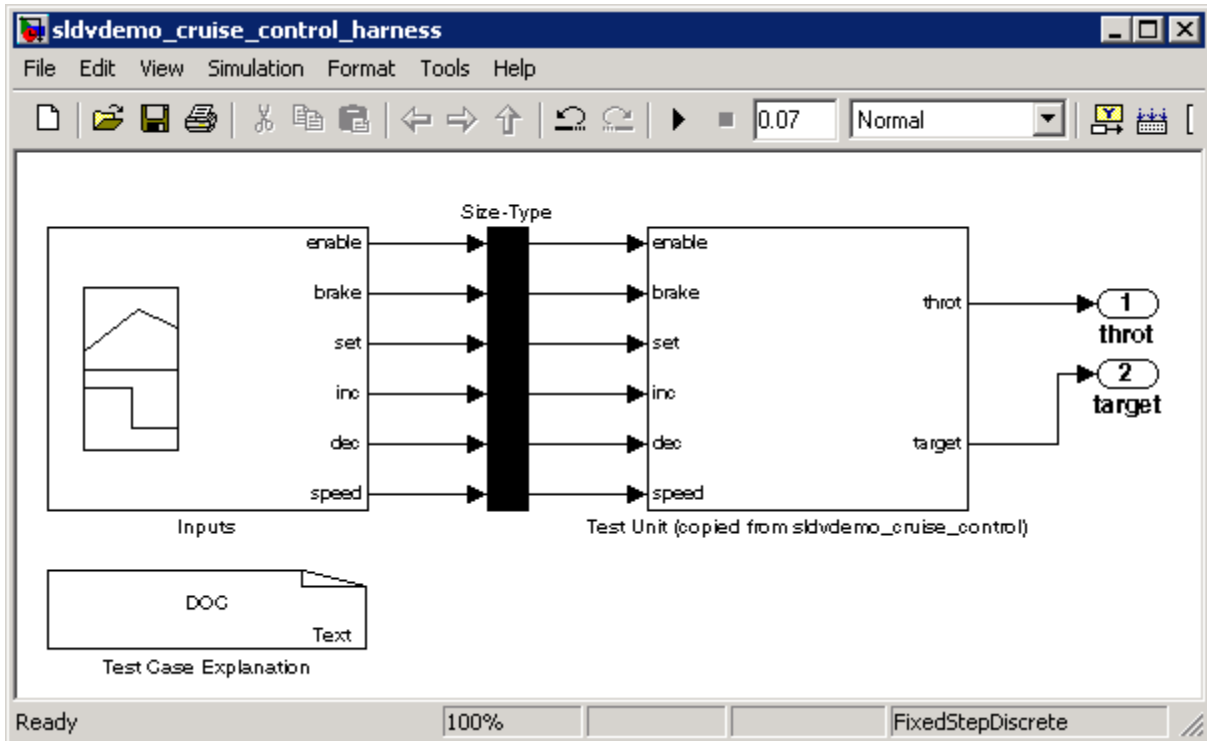
When the Simulink Design Verifier software completes its analysis, it opens:

- Test harness model: `sldvdemo_cruise_control_harness.mdl`
- Signal Builder dialog box containing the test-case signals
- HTML report containing the analysis results:
`sldvdemo_cruise_control_report.html`

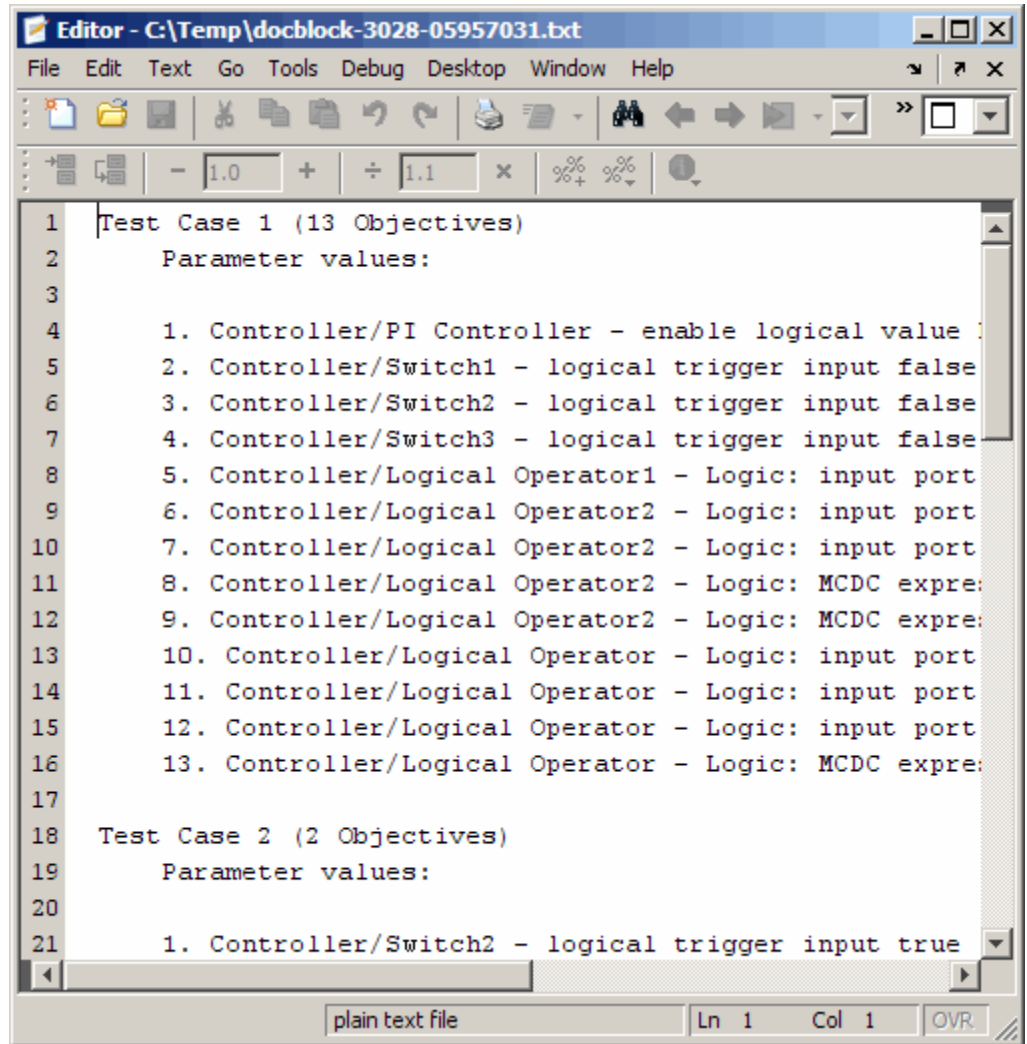
The sections that follow describe the test harness, the Signal Builder data, and the HTML report in detail.

Exploring the Test Harness


The Simulink Design Verifier software creates a test harness model when it completes its analysis. The test harness for the Cruise Control Test Generation model appears as shown in the following figure.

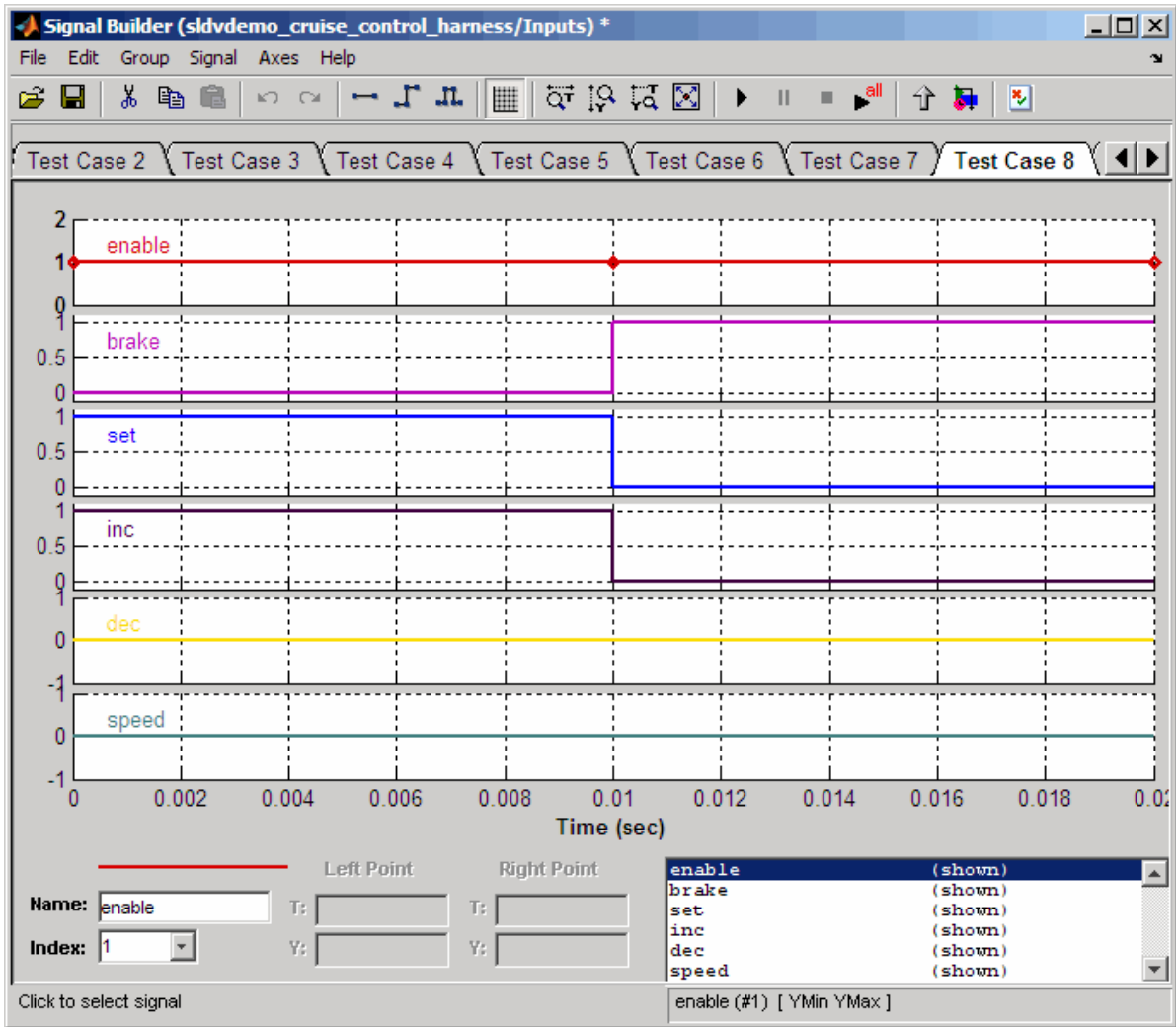


- 1 The block labeled Test Case Explanation is a DocBlock block that documents the generated test cases. Double-click the Test Case Explanation block to view a description of each test case in terms of the objectives that the test case satisfies.



- 2 The block labeled Test Unit is a Subsystem block that contains a copy of the original model the software analyzed. Double-click the Test Unit block to view its contents and confirm that it is a copy of the Cruise Control Test Generation model.

- 3** The block labeled Inputs is a Signal Builder block that contains the generated test case signals. Double-click the Inputs block to open the Signal Builder dialog box and view the 10 test case signals.
- 4** In the Signal Builder dialog box, click the right-facing arrow next to the test case tabs  to find the **Test Case 8** tab.
- 5** Click the **Test Case 8** tab to display the signal values for Test Case 8.




In Test Case 8 at 0.1 seconds:











- The enable signal remains 1.
- The brake signal transitions from 0 to 1.
- The inc and set signals transition from 1 to 0.

- The dec and speed signals remain 0.

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

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

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

Model Hierarchy/Complexity:		Test 1						
		D1	C1		MCDC			
1.	sldvdemo_cruise_control_harness	9	100%		100%		100%	
2. . . .	Test Unit (copied from sldvdemo_cruise_control)	8	100%		100%		100%	
3.	Controller	7	100%		100%		100%	
4.	PI Controller	4	100%			NA		NA

The coverage report indicates the Simulink Design Verifier software generated test cases that achieve complete coverage for the Cruise Control Test Generation model.

Interpreting the Simulink Design Verifier HTML Report

The Simulink Design Verifier software creates an HTML report that summarizes its analysis results.

If the report is not open in a Web Browser window, open it now. The path name is:

```
matlabroot/sldv_output/sldvdemo_cruise_control/sldvdemo_cruise_control_report.html
```

Note The log window contains the exact path name for the HTML report.

The HTML report includes the following chapters.

Table of Contents
1. Summary
2. Analysis Information
3. Test Objectives Status
4. Model Items
5. Test Cases

Each the following sections for a description of each report chapter:

- “Summary” on page 1-16
- “Analysis Information” on page 1-17
- “Test Objectives Status” on page 1-19
- “Model Items” on page 1-21
- “Test Cases” on page 1-22

Summary. In the **Table of Contents**, click **Summary** to display the Summary chapter, which includes the following information:

- Name of the model
- Mode of the analysis (test generation or property proving)
- Status of the analysis
- Number of objectives satisfied

Chapter 1. Summary

Analysis Information

Model: sldvdemo_cruise_control
Mode: TestGeneration
Status: Completed normally

Objectives Status

Number of Objectives: 34

Objectives Satisfied: 34

Analysis Information. In the **Table of Contents**, click **Analysis Information** to display information about the analyzed model and the analysis options.

Chapter 2. Analysis Information

Table of Contents

[Model Information](#)
[Analysis Options](#)
[Constraints](#)
[Approximations](#)

Model Information

File: \\mathworks\ah\devel\jobarchive\Aslrv\latest_pass\matlab\toolbox\sldv\sldvdemos\sldvdemo149\slvdemo149.m
 Version: 1.49
 Time Stamp: Thu Nov 20 22:52:48 2008
 Author:

Analysis Options

Mode: TestGeneration
 Test Suite Optimization: CombinedObjectives
 Maximum Testcase Steps: 500 time steps
 Test Conditions: UseLocalSettings
 Test Objectives: UseLocalSettings
 Model Coverage Objectives: MCDC
 Maximum Processing Time: 60s
 Block Replacement: off
 Parameters Analysis: on
 Parameters Configuration File: sldv_params_template.m
 Save Data: on
 Save Harness: on
 Save Report: on

Constraints

Name	Constraint
constraint	[0, 100]

Approximations

Simulink Design Verifier performed the following approximations during analysis. These can impact the precision of the results generated by Simulink Design Verifier. Please see the product documentation for further details.

 	Type	Description
1	Rational approximation	The model includes floating-point arithmetic. Simulink Design Verifier approximates floating-point arithmetic with rational number arithmetic.

Test Objectives Status. In the **Table of Contents**, click **Test Objectives Status** to display a table of satisfied objectives. The following figure shows a partial list of the objectives satisfied in the Cruise Control Test Generation model.

Chapter 3. Test Objectives Status

Table of Contents

[Objectives Satisfied](#)

Objectives Satisfied

Simulink Design Verifier found test cases that exercise these test objectives.

#:	Type	Model Item	Description	Test Case
1	Decision	Controller/PI Controller	enable logical value F	1
2	Decision	Controller/PI Controller	enable logical value T	5
3	Decision	Controller/Switch1	logical trigger input false (output is from 3rd input port)	1
4	Decision	Controller/Switch1	logical trigger input true (output is from 1st input port)	5
5	Decision	Controller/PI Controller/Discrete-Time Integrator	integration result <= lower limit F	5
6	Decision	Controller/PI Controller/Discrete-Time Integrator	integration result <= lower limit T	10
7	Decision	Controller/PI Controller/Discrete-Time Integrator	integration result >= upper limit F	5

The Objectives Satisfied table lists the following information for the model:

- **#** — Objective number.
- **Type** — Objective type.
- **Model Item** — Element in the model for which the objective was tested. Click this link to display the model with this element highlighted.
- **Description** — Description of the objective.
- **Test case** — Test case that achieves the objective. Click this link to get more information about that test case.

In the row for objective 17, click the test case number (8) to display more information about test case 8 in the report's Test Cases chapter.

Test Case 8

Summary

Length: 0.01 Seconds (2 sample periods)

Objective Count: 2

Objectives

Step	Time	Model Item	Objectives
2	0.01	Controller/Logical Operator2 Controller/Logical Operator2	Logic: input port 2 T Logic: MCDC expression for output with input port 2 T

Generated Input Data

Time	00.01
Step	12
enable	11
brake	01
set	10
inc	1-
dec	0-
speed	00

In this example, Test Case 8 satisfies 2 model coverage objectives. The following signal values achieve the objectives listed in the **Objectives** column of the table:

- The enable signal remains 1.
- The brake signal transitions from 0 to 1 at 0.1 seconds.

- The inc and set signals transition from 1 to 0 at 0.1 seconds.
- The dec and speed signals remain 0.

This information matches what you see in the test harness model. Specifically, the Inputs block in the test harness depicts identical signal values for Test Case 8, and the Test Case Explanation block lists 2 objectives that Test Case 8 achieves (see “Exploring the Test Harness” on page 1-10).

Model Items. In the **Table of Contents**, click **Model Items** to see detailed information about each item in the model that defines coverage objectives. This table includes the status of the objective at the end of the analysis. Click the links in the table to get detailed information about the satisfied objectives.

Chapter 4. Model Items

Table of Contents

[Controller/PI Controller](#)
[Controller/Switch1](#)
[Controller/PI Controller/Discrete-Time Integrator](#)
[Controller/Switch2](#)
[Controller/Switch3](#)
[Controller/Logical Operator1](#)
[Controller/Logical Operator2](#)
[Controller/Logical Operator](#)

This section presents, for each object in the model defining coverage objectives, the list of objectives and their individual status at the end of the analysis. It should match the coverage report obtained from running the generated test suite on the model, either from the harness model or by using the `sldvrntests` command.

Controller/PI Controller

[View](#)

#:	Type	Description	Status	Test Case
1	Decision	enable logical value F	Satisfied	1
2	Decision	enable logical value T	Satisfied	5

Controller/Switch1

[View](#)

#:	Type	Description	Status	Test Case
3	Decision	logical trigger input false (output is from 3rd input port)	Satisfied	1
4	Decision	logical trigger input true (output is from 1st input port)	Satisfied	5

Test Cases. In the **Table of Contents**, click **Test Cases** to display detailed information about each generated test case, including:

- Length of time to execute the test case
- Number of objectives satisfied

- Detailed information about the satisfied objectives
- Input data

See the section for Test Case 8 in “Test Objectives Status” on page 1-19

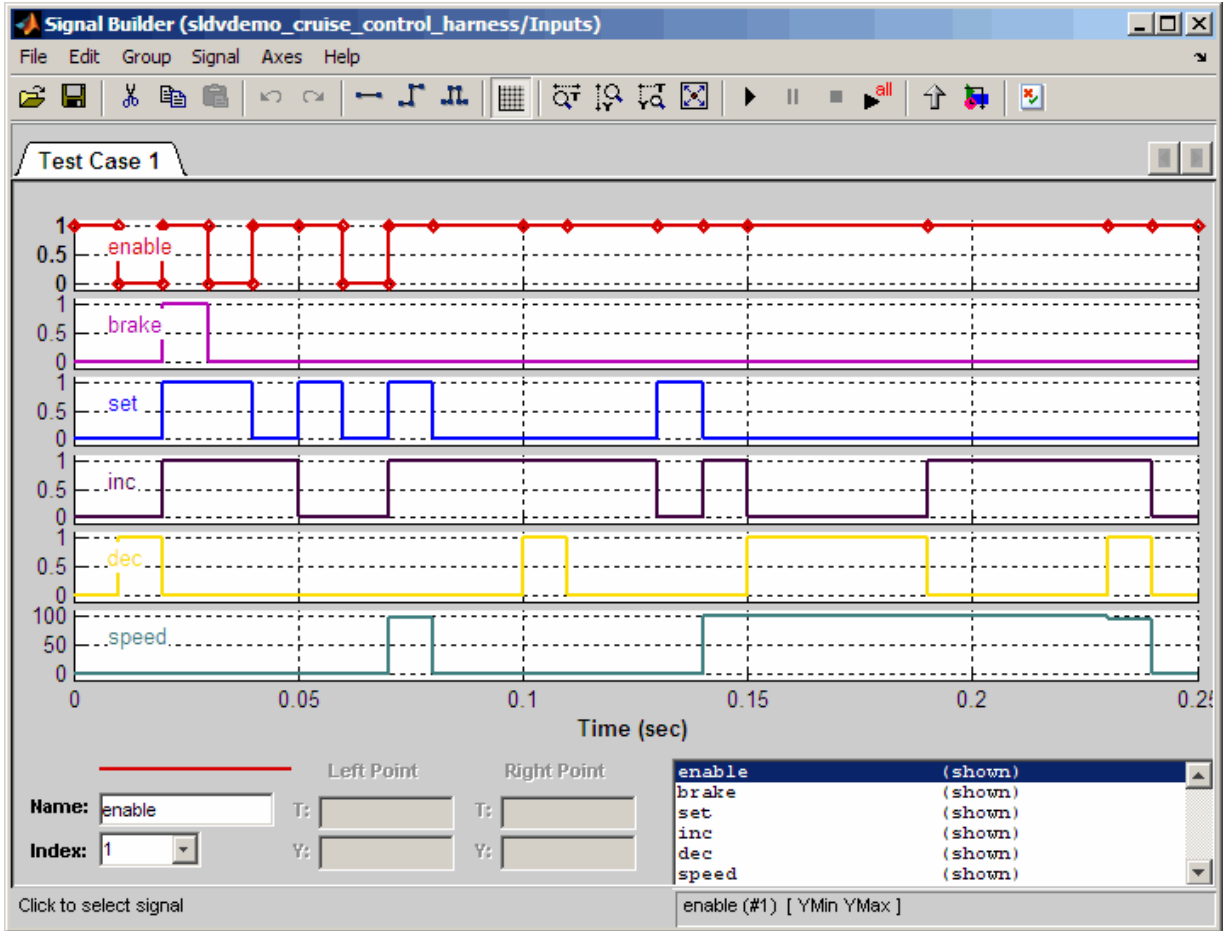
Combining Test Cases

If you prefer to review results that are combined into a smaller number of test cases, set the **Test suite optimization** parameter to **Long test cases**. When you use the **Long test cases** optimization, the analysis generates fewer, but longer, test cases that each satisfy multiple test objectives. This optimization creates a more efficient analysis and easier-to-review results.

Open the `sldvdemo_cruise_control` model and rerun the analysis with the **Long test cases** optimization:

- 1** Select **Tools > Design Verifier > Options**.
- 2** In the **Select** tree on the left side of the Configuration Parameters dialog box, in the **Design Verifier** category, select **Test Generation**.
- 3** Set the **Test suite optimization** parameter to **Long test cases**.
- 4** Click **Apply** and **OK** to close the Configuration Parameters dialog box.
- 5** In the `sldvdemo_cruise_control` model, double-click the block labeled **Run**.

The Signal Builder dialog box now contains one test case instead of ten test cases.



This HTML report contains one section describing Test Case 1.

Test Case 1

Summary

Length: 0.24 Seconds (25 sample periods)

Objective Count: 34

Objectives

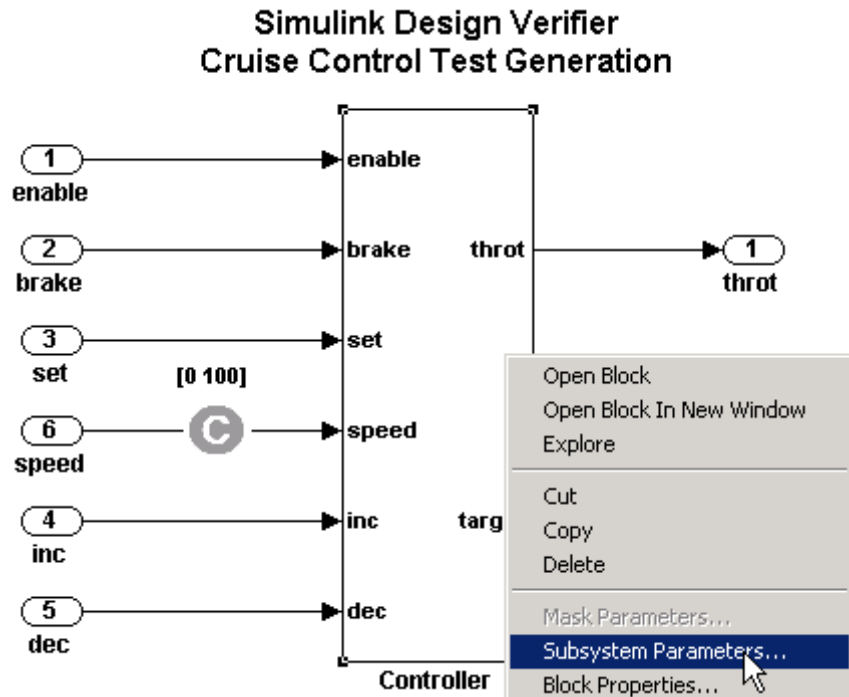
Step	Time	Model Item	Objectives
1	0	Controller/Switch2 Controller/Switch1 Controller/Switch3 Controller/Logical Operator Controller/Logical Operator1 Controller/Logical Operator2 Controller/Logical Operator2 Controller/PI Controller Controller/Logical Operator Controller/Logical Operator Controller/Logical Operator Controller/Logical Operator2 Controller/Logical Operator2	logical trigger input false (output is from 3rd input port) logical trigger input false (output is from 3rd input port) logical trigger input false (output is from 3rd input port) Logic: MCDC expression for output with input port 3 F Logic: input port 1 F Logic: MCDC expression for output with input port 2 F Logic: MCDC expression for output with input port 1 F enable logical value F Logic: input port 3 F Logic: input port 2 T Logic: input port 1 T Logic: input port 2 F Logic: input port 1 F
2	0.01	Controller/Switch3 Controller/Logical Operator	logical trigger input true (output is from 1st input port) Logic: input port 1 F
3	0.02	Controller/Switch1 Controller/Logical Operator Controller/Logical Operator2 Controller/Logical Operator Controller/Logical Operator2 Controller/Logical Operator1	logical trigger input true (output is from 1st input port) Logic: MCDC expression for output with input port 2 F Logic: MCDC expression for output with input port 1 T Logic: input port 2 F Logic: input port 1 T Logic: input port 1 T
4	0.03	Controller/Logical Operator	Logic: MCDC expression for output with input port 1 F

Analyzing a Subsystem

In addition to analyzing a model, you can analyze a subsystem within a model. This technique is good for large models, where you want to review the analysis in smaller, manageable reports.

This example analyzes the Controller subsystem in the `sldvdemo_cruise_control` model from “Analyzing a Model” on page 1-6.

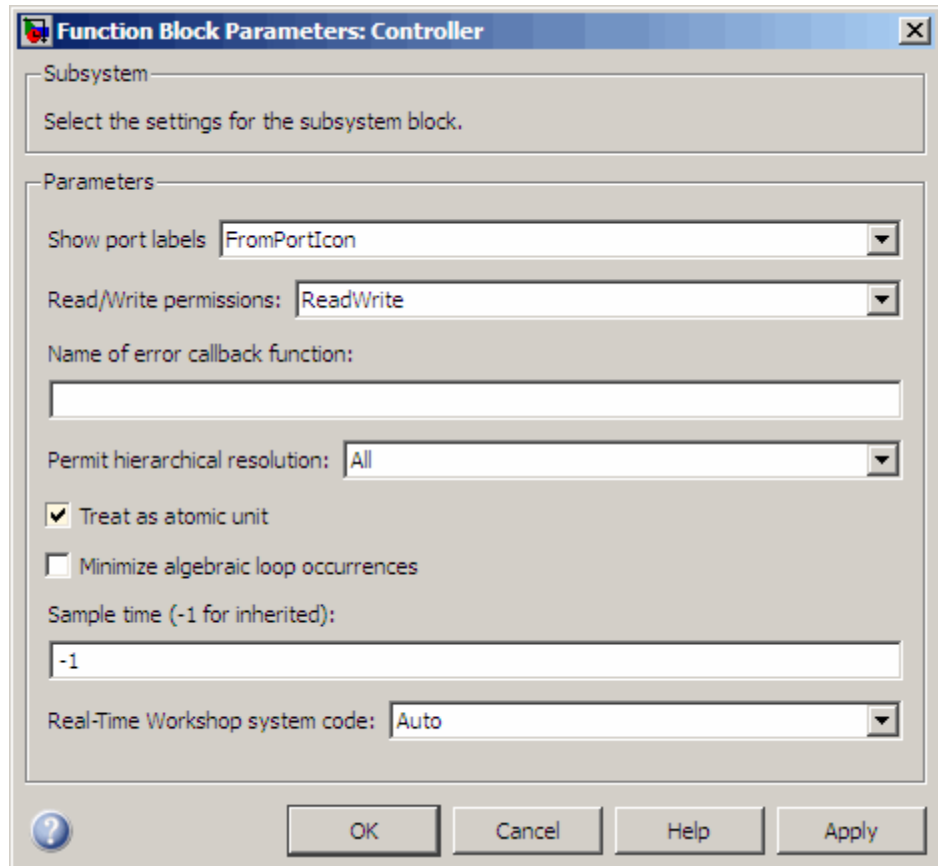
- 1 Enter `sldvdemo_cruise_control` at the MATLAB command line to open the Cruise Control Test Generation model.
- 2 Right-click the Controller subsystem, and select **Subsystem Parameters**.



- 3 In the Function Block Parameters dialog box, select **Treat as atomic unit**.

An *atomic subsystem* executes as a unit relative to the parent model; subsystem block execution does not interleave with parent block execution. This makes it possible to extract subsystems for use as standalone models.

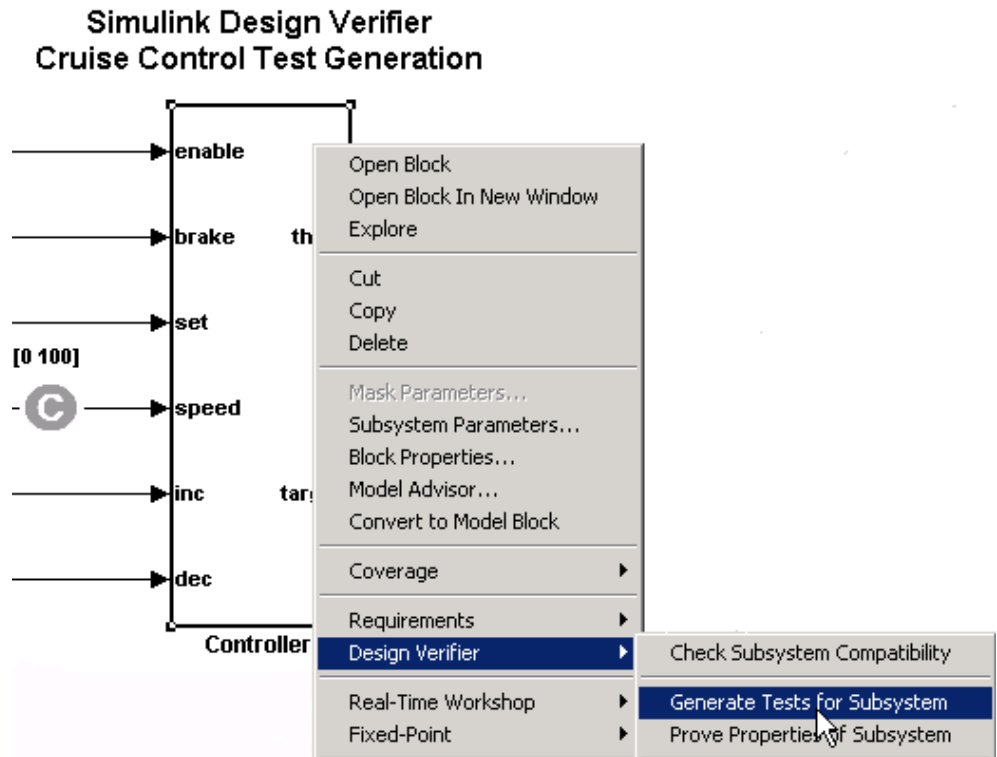
You must set the **Treat as atomic unit** parameter to analyze a subsystem with the Simulink Design Verifier software.



After you set the parameter, other options become available, but you can ignore them.

- 4 Click **Apply** and **OK** to close the dialog box.

- 5 Select **File > Save As** and save the Cruise Control Test Generation model under a new name.
- 6 To start the subsystem analysis and generate test cases, right-click the Controller subsystem, and select **Design Verifier > Generate Tests for Subsystem**.



- 7 The Simulink Design Verifier software creates and opens the following output. Except for the new model, all of these correspond to the model analysis output:
 - A new model containing just the Controller subsystem: `Controller.mdl`
 - Test harness model: `Controller_harness.mdl`
 - Signal Builder dialog box containing the test-case signals

- HTML report containing the analysis results: `Controller_report.htm`
- 8** Review the results of the subsystem analysis (harness model and HTML report) and compare them to the results of the full-model analysis described in “Analyzing a Model” on page 1-6.
- The subsystem analysis analyzes the Controller as a standalone model.
 - The Controller subsystem contains all the test objectives in the Cruise Control Test Generation model, so both analyses generate the same test cases.

Basic Workflow for Using the Simulink Design Verifier Software

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 3, "Ensuring Compatibility with the Simulink® Design Verifier Software"
2	Optionally, prepare your model for analysis.	Chapter 4, "Working with Block Replacements" Chapter 5, "Specifying Parameter Configurations"
3	Set Simulink Design Verifier options.	Chapter 6, "Configuring Simulink® Design Verifier Options"
4	Generate test cases for your model or prove its properties.	Chapter 7, "Generating Test Cases" Chapter 8, "Proving Properties of a Model"
5	Interpret the results.	Chapter 9, "Reviewing the Results"

Learning More


In this section...
“Next Step” on page 1-31
“Product Help” on page 1-31
“The MathWorks Online” on page 1-32

Next Step

To begin learning how to use the Simulink Design Verifier software, see Chapter 3, “Ensuring Compatibility with the Simulink® Design Verifier Software”. Also see the following topics to continue your exploration of the software:

For...	See...
Exercise that walks you through the process of generating test cases for a model	“Generating Test Cases for a Model” on page 7-5
Exercise that walks you through the process of proving a model property	“Proving Properties in a Model” on page 8-5

Product Help

More information is available with your product installation. In the MATLAB desktop, click  for help, and then click the product name in the **Contents** pane.

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

The MathWorks Online

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

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

How the Simulink Design Verifier Software Works

- “Model Analysis with Simulink® Design Verifier Software” on page 2-2
- “Analyzing a Simple Model” on page 2-3
- “Analyzing Large Models” on page 2-5
- “Handling Incompatibilities with Automatic Stubbing” on page 2-6
- “Approximations” on page 2-14

Model Analysis with Simulink Design Verifier Software

Simulink Design Verifier software is an efficient analysis tool that explores the simulation behavior of a Simulink model. It searches the possible values of model inputs and block parameters to find a simulation that satisfies test objectives. The software also proves model properties and generates examples of violations.

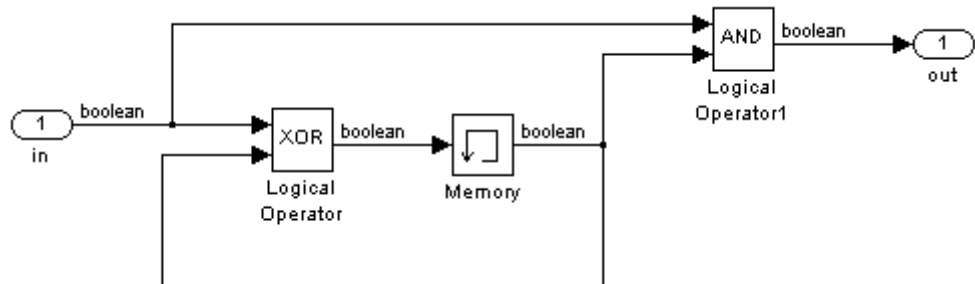
Such analysis always begins with the initial configuration of the model and can span an arbitrary number of time steps. Generally, there is an infinite number of paths through the model because the values of inputs are independent from one time step to the next, and there is no fixed limit to the number of time steps.

If the software finds no way to reduce the search space, it would continue its analysis indefinitely. Thus, the software limits the analysis by tracking the persistent information in the model such as discrete states, data-store memories, and persistent variables.

After an analysis explores all possible inputs and parameters from all possible configurations, the results equal those of a complete search of every possible infinite sequence of inputs parameters.

Analyzing a Simple Model

This simple Simulink model includes 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. The following table describes the complete behavior of the model, including the behavior that would result from an arbitrarily long sequence of inputs.

#	Input	Memory Value	Output of XOR Block = Next Memory Value	Output of AND Block
1	false	false	false	false
2	true	false	true	false
3	false	true	true	false
4	true	true	false	true

Suppose you want to generate test cases that result in a true output; this goal is your *test objective*. If you run the Simulink Design Verifier software to generate test cases that result in a true output, the software searches this table to see if such a scenario is possible.

After the Simulink Design Verifier software discovers a configuration that satisfies the test objective (in this case, when both the input and the Memory block output are true), it needs to find a path to reach this configuration from

the initial conditions. If the initial memory value is true, the test case only needs to be a single time step (row 4) where the input was true.

If the initial memory value is false (the default), the test case must force the memory value to be true. In this example, the path requires two steps:

- 1** The input value is true and the memory value is false (row 2). Thus, the output of the XOR block is true, making the memory value true.
- 2** Now that the input value and memory value are both true (row 4), the output is true, so the analysis achieves the specified test objective.

An infinite number of test cases can cause the output to be true, and regardless of the state value, the output can be held false for an arbitrary time before making it true. When the Simulink Design Verifier software searches, it returns the first test case it encounters that satisfies the objective. This case is invariably the simulation with the fewest time steps. Sometimes you may find this result 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 case generation. During test case generation, option parameters explicitly specify the search criteria. For example, you can specify that Simulink Design Verifier software find paths for all outputs or find only those paths that make where the output is true.

During property proving, you specify a functional requirement, or property, that you want the Simulink Design Verifier software to prove, for example, that the output is always true. If the search completes without finding a path that violates the property, the proof of that property completes successfully. If the software finds a path where the output is false, it creates a counterexample that causes the output to be false.

Analyzing Large Models

In larger, more complicated models, the Simulink Design Verifier software uses mathematical techniques to simplify the analysis:

- It identifies portions of the model that do not affect the desired objectives.
- It discovers relationships within the model that reduce the complexity of the search.
- It reuses intermediate results from one objective to another.

In this way, the problem is reduced to a search through the logical values that describe your model.

For detailed information about analyzing large models, see Chapter 10, “Analyzing Large Models and Improving Performance”.

Handling Incompatibilities with Automatic Stubbing

In this section...
“What Is Automatic Stubbing?” on page 2-6
“Analyzing a Model Using Automatic Stubbing” on page 2-6

What Is Automatic Stubbing?

Automatic stubbing allows you to run a test case generation or property-proving analysis on a model that contains elements that the Simulink Design Verifier software does not support.

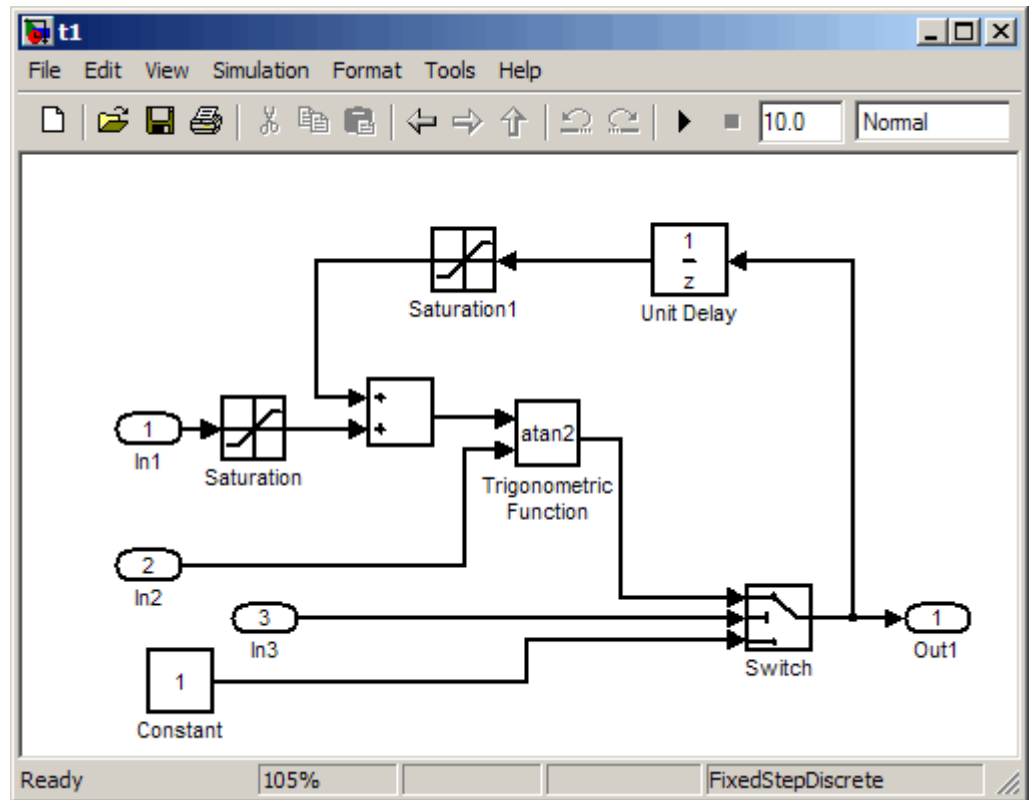
When you enable automatic stubbing option, the software considers only the interface of the unsupported elements, not their actual behavior. This technique allows the software to complete the analysis. However, the analysis may achieve only partial results if any of the unsupported model elements affect the simulation outcome.

Analyzing a Model Using Automatic Stubbing

This section describes a workflow for using automatic stubbing, using a simple Simulink model (t1) as an example.

- “Checking Model Compatibility” on page 2-7
- “Turning On Automatic Stubbing” on page 2-10
- “Reviewing the Results” on page 2-12
- “Achieving Complete Results” on page 2-13

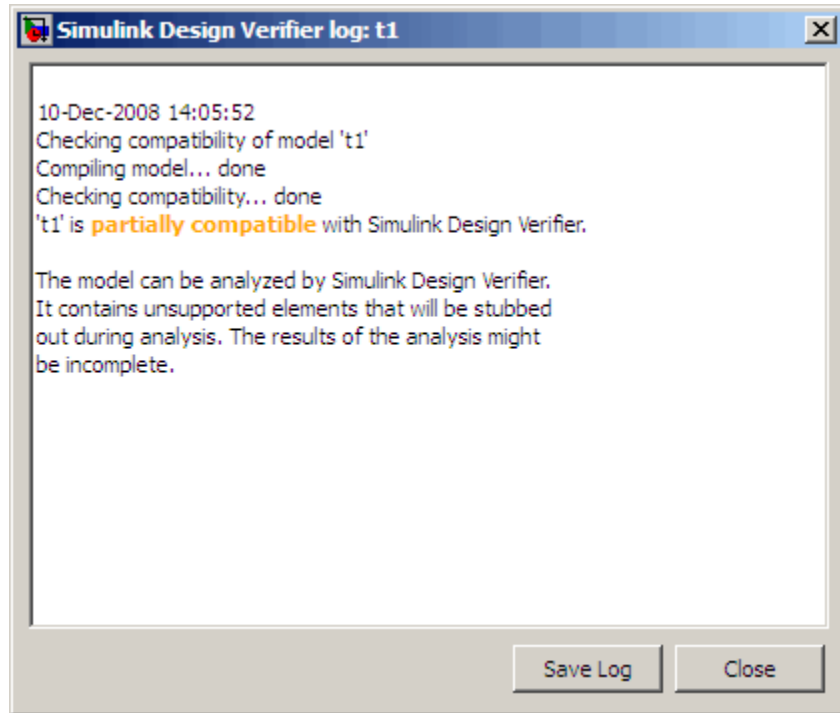
The t1 model contains a Trigonometric Function block, which is not compatible with the Simulink Design Verifier software.



Checking Model Compatibility

From the Model Editor, there are two ways to check whether a model is compatible with the Simulink Design Verifier software:

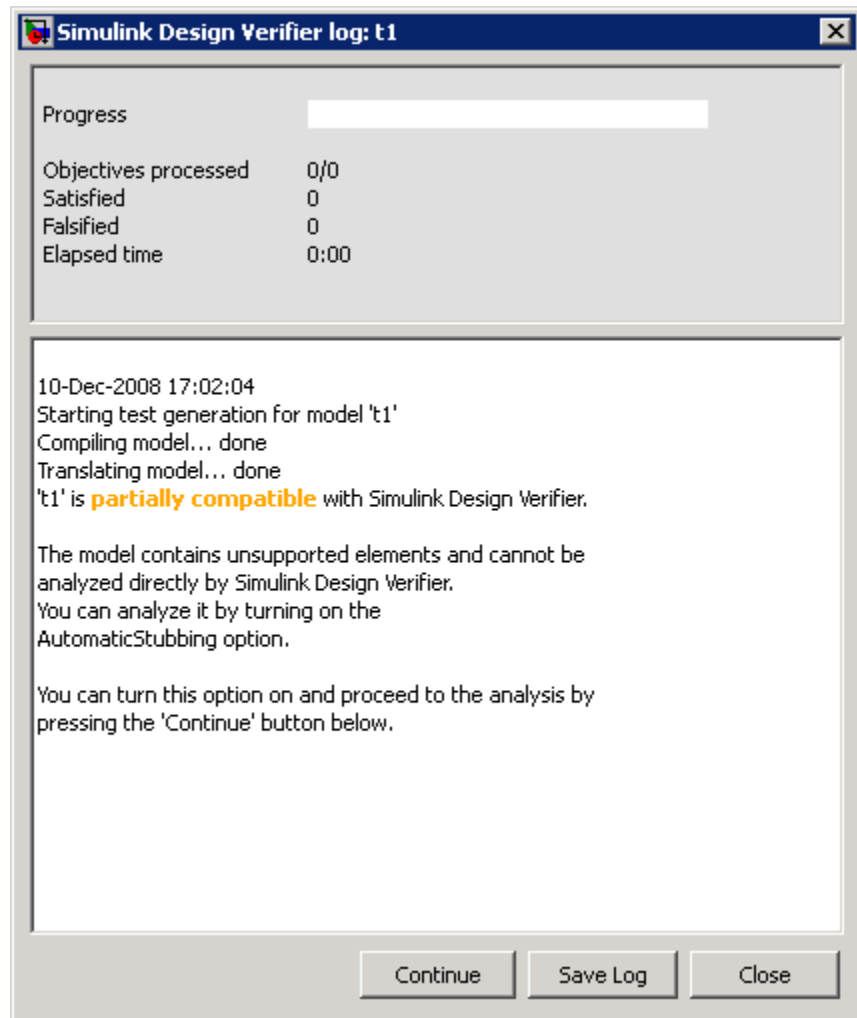
- Run the Simulink Design Verifier compatibility check by selecting **Tools > Design Verifier > Check Model Compatibility**.



- Select the analysis that you want:
 - **Tools > Design Verifier > Generate Tests**
 - **Tools > Design Verifier > Prove Properties**

The software first checks the compatibility of the model. If the model itself is incompatible, for example, if it uses a variable-step solver, the analysis cannot continue.

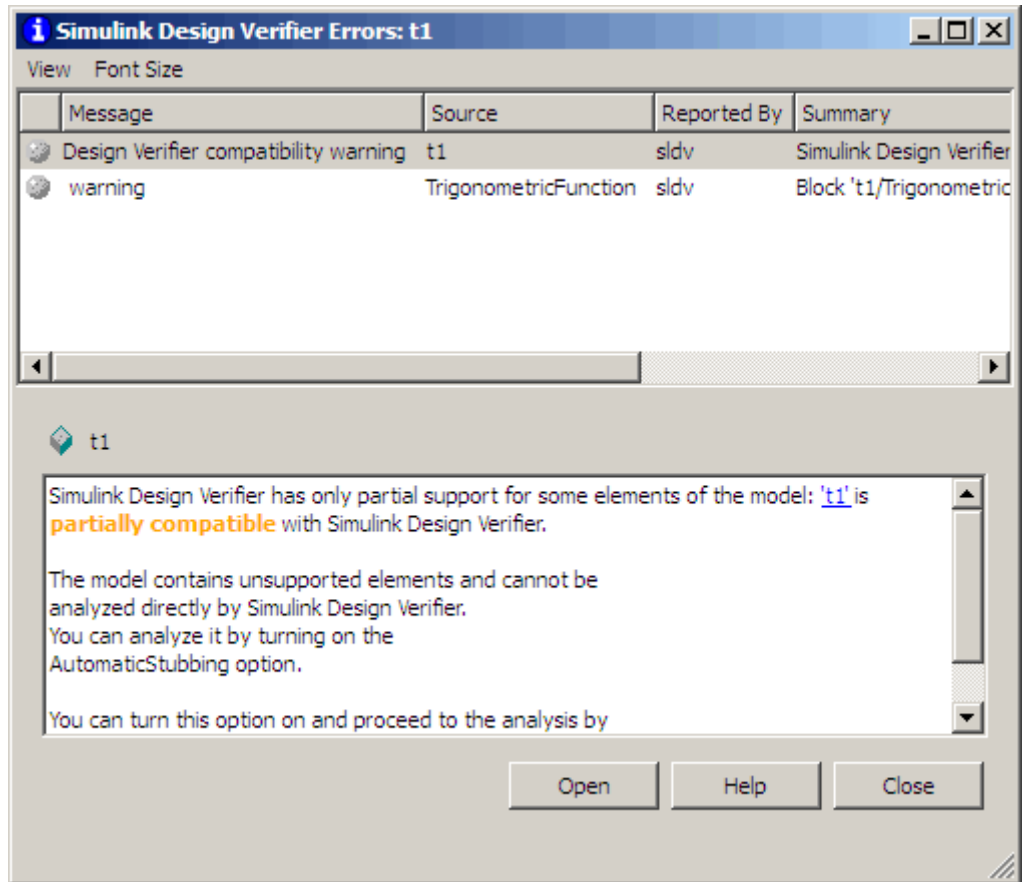
If it finds incompatible elements in the model, the software stops and asks if you want to turn on automatic stubbing.



You can:

- Save the log file.
- Continue the analysis.
- Terminate the analysis.

The Simulation Diagnostics Viewer is also displayed, listing the incompatibilities. (For more information about this dialog box, see “Simulation Diagnostics Viewer” in the *Simulink User’s Guide*.)

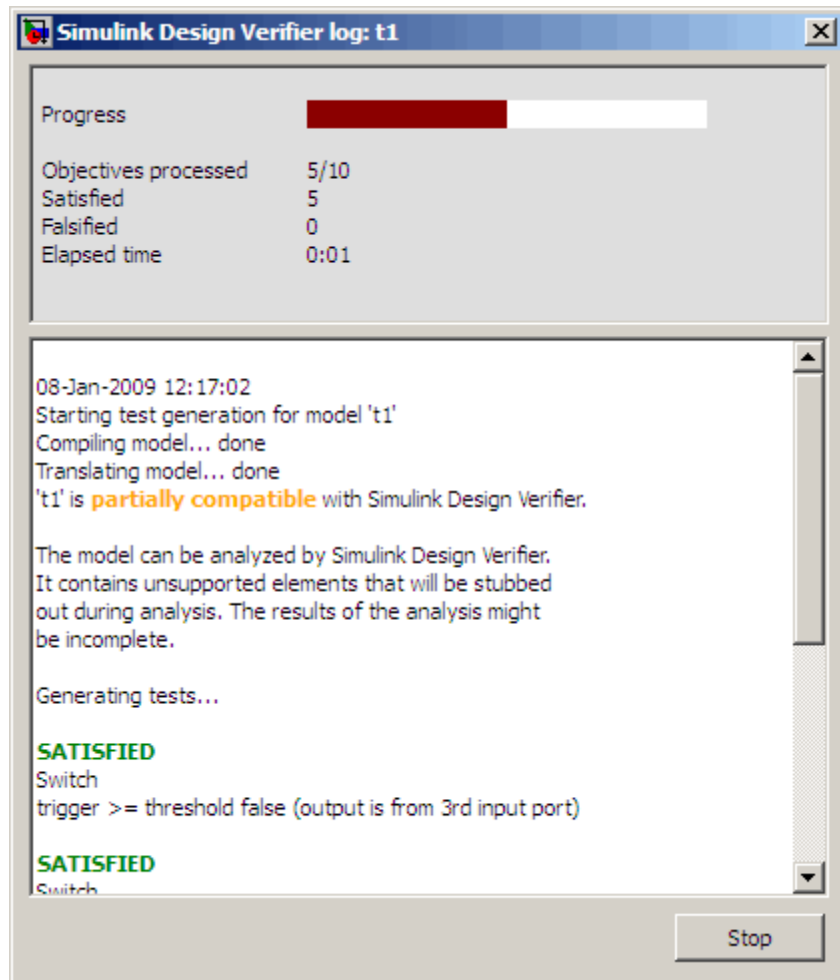


Turning On Automatic Stubbing

There are two ways to turn on automatic stubbing:

- If you have not turned on automatic stubbing and the analysis finds at least one incompatibility, the analysis stops and asks if you want to turn on automatic stubbing. Click **Continue** to proceed with the analysis.

- Before starting the analysis, in the Configuration Parameters dialog box, on the main Design Verifier pane, select **Automatic stubbing of unsupported block and functions**. When you run the analysis, you are notified that stubbing is turned on and the analysis continues.



Reviewing the Results

If you ran the analysis with automatic stubbing enabled, make sure to review the results. In this report, you see a table of unsupported blocks that the software encountered.

Unsupported Blocks

The following blocks are not supported by Simulink Design Verifier. They were abstracted during the analysis. This can lead Simulink Design Verifier to produce only partial results for parts of the model that depends on the output values of these blocks.

Block	Type
Trigonometric Function	Trigonometry

The Summary report for the t1 example model shows that one objective was satisfied without generating a test case. The software cannot generate the test case because it does not understand the operation of the Trigonometric Function block.

Chapter 1. Summary

Analysis Information

Model:	t1
Mode:	TestGeneration
Status:	Completed normally

Objectives Status

Number of Objectives:	10
Objectives Satisfied:	9
Objectives Satisfied - No Test Case:	1

Achieving Complete Results

If your analysis does not achieve complete results because of the stubbing, you can define custom block replacements to give a more precise definition of the unsupported blocks. For more information:

- “Defining Custom Block Replacements” on page 4-7.
- Enter

```
echodemo s1dvdemo_blockreplacement_unsupportedblocks
```

to step through the “Block Replacements for Unsupported Models” demo.

Approximations

In this section...
“Approximations During Model Analysis” on page 2-14
“Types of Approximations” on page 2-14
“Converting Floating-Point Arithmetic to Rational-Number Arithmetic ” on page 2-14
“Linearizing 2-D Lookup Tables” on page 2-15
“Unrolling While Loops” on page 2-15
“Ensuring the Validity of the Analysis” on page 2-15

Approximations During Model Analysis

The Simulink Design Verifier software attempts to generate inputs and parameters to achieve test and proof objectives. However, there could be an infinite number of values for the software to search. To create reasonable limits on the analysis, the software performs approximations to simplify the analysis. The software records any approximations it performed in the Analysis Information chapter of the Simulink Design Verifier HTML report.

Types of Approximations

Simulink Design Verifier software performs three types of approximations when it analyzes a model:

- “Converting Floating-Point Arithmetic to Rational-Number Arithmetic ” on page 2-14
- “Linearizing 2-D Lookup Tables” on page 2-15
- “Unrolling While Loops” on page 2-15

Converting Floating-Point Arithmetic to Rational-Number Arithmetic

The Simulink Design Verifier software simplifies the linear arithmetic of floating-point numbers by approximating them with infinite-precision rational numbers. The software discovers how the logical relationships

between these values affects the proof and test objectives. This analysis enables the software to support supervisory logic that is commonly found in embedded controls designs.

If your model contains floating-point values in the signals, input values, or block parameters, the Simulink Design Verifier software converts those values to rational numbers before performing its analysis.

Note As a result of these approximations, Simulink Design Verifier software does not consider the effect of round-off error, or the upper and lower bounds of floating-point numbers.

Linearizing 2-D Lookup Tables

The Simulink Design Verifier software does not support nonlinear arithmetic. If your model contains any Lookup Table (2-D) blocks, the software approximates nonlinear 2-D interpolation with linear interpolation by fitting planes to each interpolation interval, if necessary.

Unrolling While Loops

If your model or any Stateflow chart in your model contains a while loop, the Simulink Design Verifier software tries to find a bound that allows the while loop to exit. To find a bound, it unrolls the while loop and executes it three times. If the software does not find a bound for a test case generation analysis, it sets the number of loop iterations to three for the purpose of the analysis. If you are performing a property-proving analysis, the analysis terminates.

Ensuring the Validity of the Analysis

The Simulink Design Verifier software records all approximations it performed in the Analysis Information chapter of the HTML report. (For a description of the contents of this chapter, see “Analysis Information Chapter” on page 9-20.)

Review the analysis results carefully when the software uses approximations. Evaluate your model to identify which blocks or subsystems caused the software to perform the approximations.

In rare cases, an approximation can result in test cases that fail to achieve test objectives, or counterexamples that fail to falsify proof objectives. For example, suppose the software generates a test case signal that should achieve an objective by exceeding a threshold; a floating-point round-off error might prevent that signal from attaining the threshold value.

Ensuring Compatibility with the Simulink Design Verifier Software

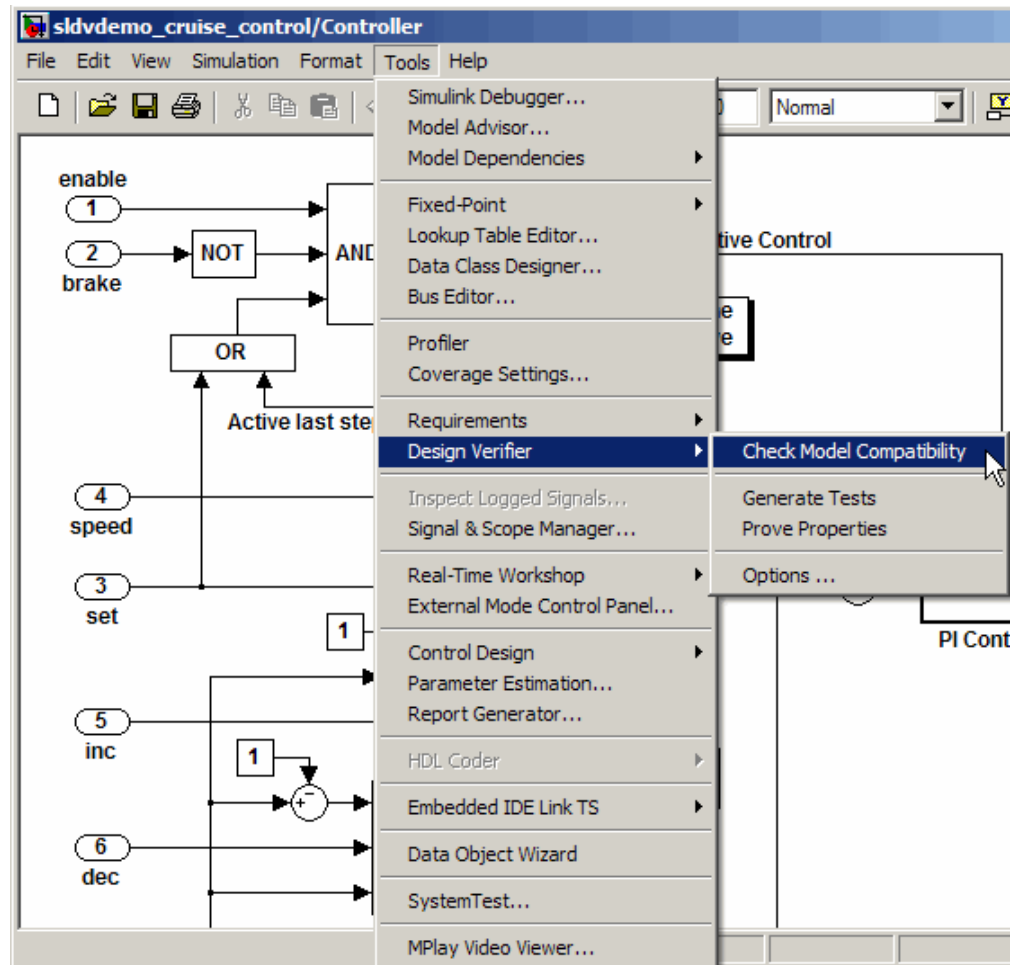
- “Checking Model Compatibility” on page 3-2
- “Unsupported Simulink Software Features” on page 3-9
- “Unsupported Stateflow Software Features” on page 3-13
- “Support Limitations for the Embedded MATLAB Subset” on page 3-15
- “Fixed-Point Support Limitations” on page 3-18

Checking Model Compatibility

The Simulink Design Verifier software supports a broad range of Simulink and Stateflow software features. However, there are features that the product does not support. Therefore, you must avoid using particular features in models that you plan to analyze with the Simulink Design Verifier software.

The Simulink Design Verifier software automatically checks the compatibility of your model before it begins a test-generation or property-proving analysis.

In addition, you can run the same check before you start the analysis. To run this check, in the model window, select **Tools > Design Verifier > Check Model Compatibility**.



Alternatively, you can use the `sldvcompat` function to run the compatibility checker programmatically at the command line or in an M-file program. For more information, see the `sldvcompat` reference page.

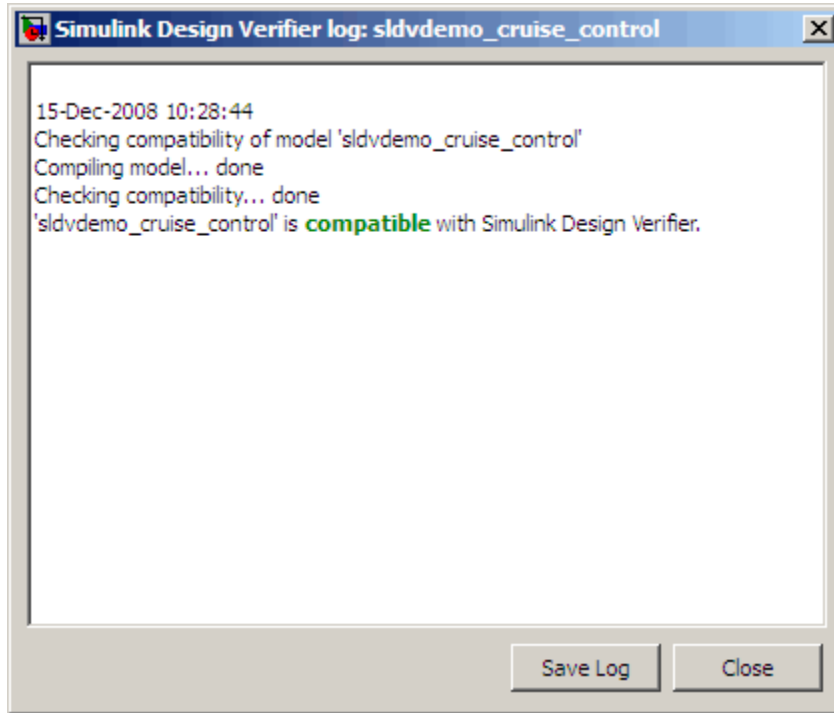
There are three outcomes of a compatibility check:

- “Model Is Compatible” on page 3-4
- “Model Is Incompatible” on page 3-4

- “Some Model Elements Are Incompatible” on page 3-6

Model Is Compatible

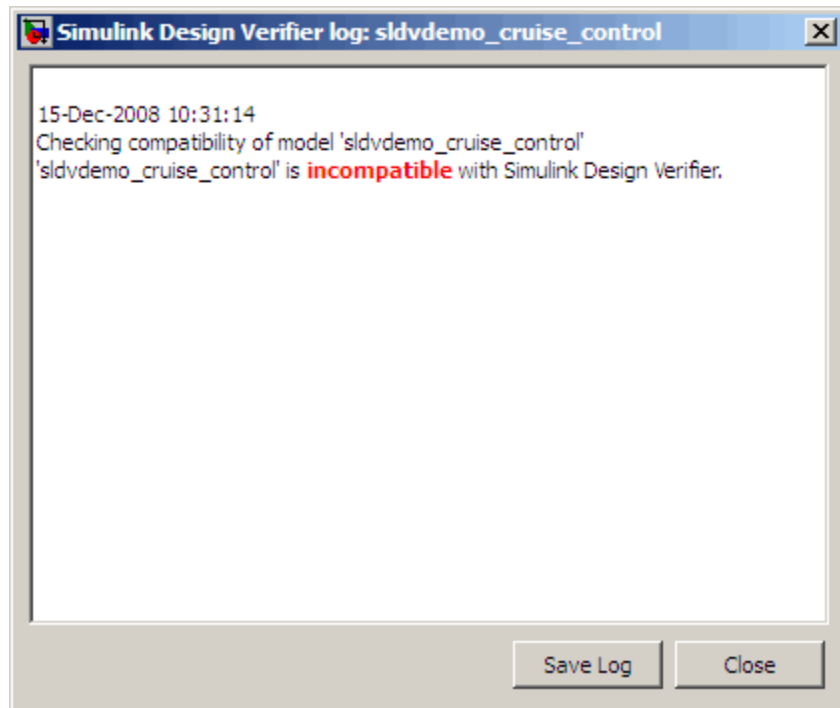
In the log window, you see if your model is compatible with the Simulink Design Verifier software.



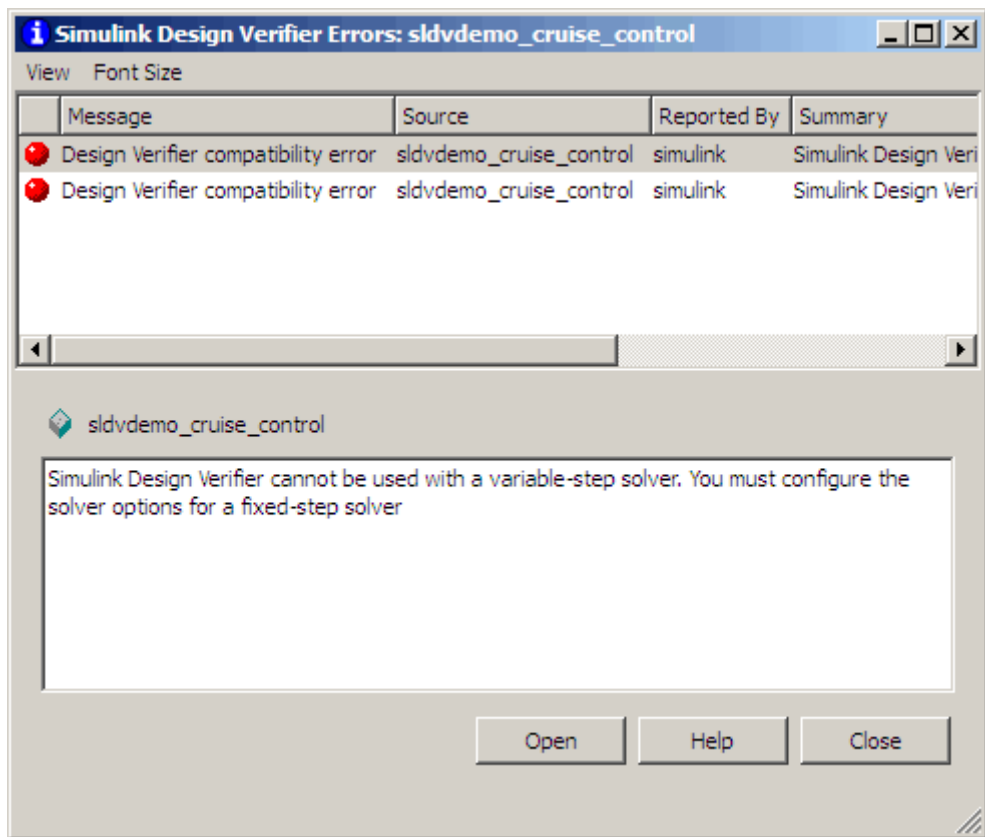
Model Is Incompatible

If the model itself is incompatible with the software, for example, if it uses a variable-step solver, you see two dialog boxes:

- Simulink Design Verifier log



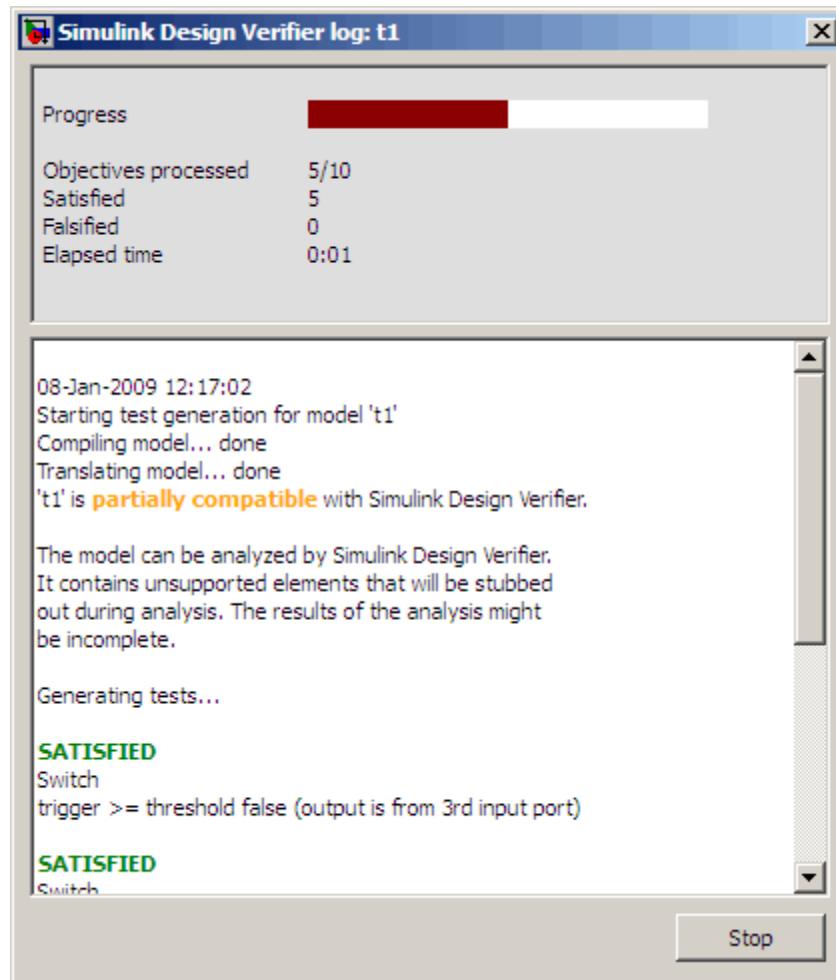
- Simulation Diagnostics Viewer. Use the information in this dialog box to identify and correct the incompatibility.



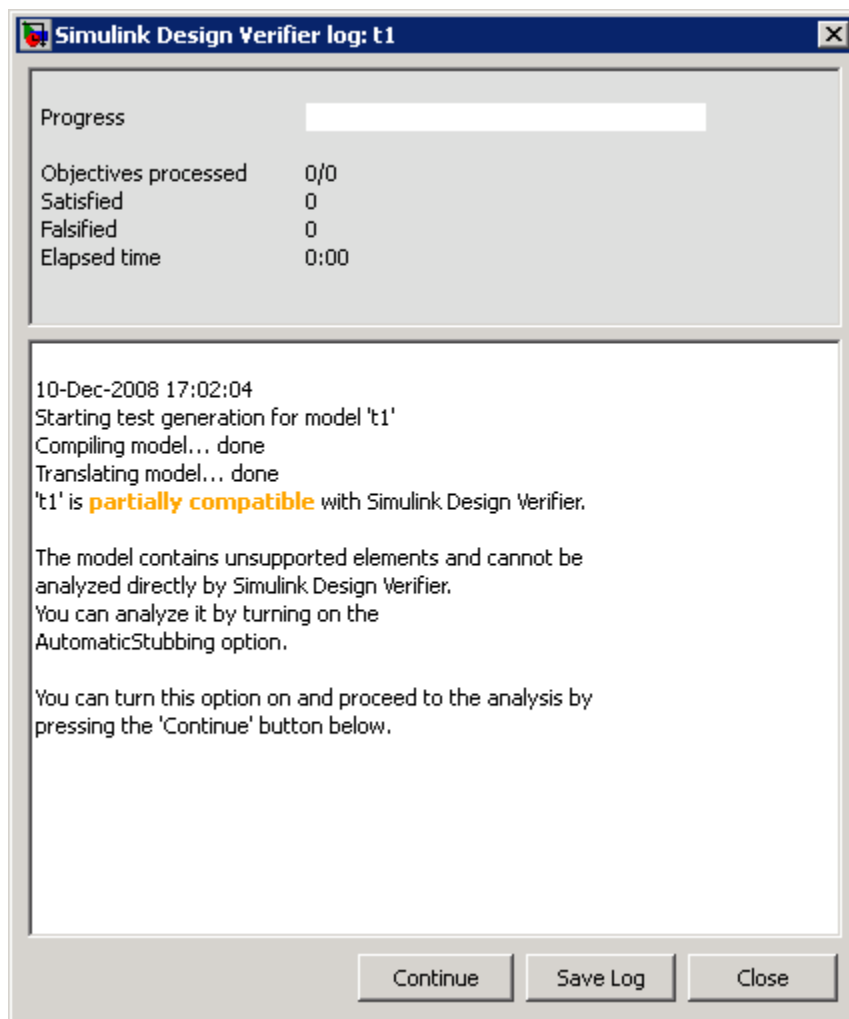
Note For more information about this dialog box, see “Simulation Diagnostics Viewer” in the *Simulink User’s Guide*.

Some Model Elements Are Incompatible

If at least one element in the model is incompatible, a message appears in the Simulink Design Verifier log window. If you have turned on automatic stubbing, the analysis proceeds.



If you have not turned on automatic stubbing, the analysis stops. You see a query asking if you want to turn it on so that the analysis can proceed.



Note For instructions on how to use automatic stubbing, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

Unsupported Simulink Software Features

In this section...
“Simulink Software Features Not Supported” on page 3-9
“Simulink Block Support Limitations” on page 3-11
“Limitations of Support for Model Reference” on page 3-11

Simulink Software Features Not Supported

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

Not Supported	Description
Variable-step solvers	The Simulink Design Verifier software supports only fixed-step solvers. (See “Choosing a Fixed-Step Solver” in the <i>Simulink User’s Guide</i> .)
Callback functions	The Simulink Design Verifier software does not execute model callback functions during the analysis. As a result, if a callback function changes any model parameters or workspace variables, the analysis does not reflect those changes. Callbacks called prior to analysis, such as the <code>PreLoadFcn</code> or <code>PostLoadFcn</code> model callbacks, are fully supported.
Complex signals	The Simulink Design Verifier software supports only real signals. (For contrast, see “Complex Signals” in <i>Simulink User’s Guide</i> .)

Not Supported	Description
Variable-size signals	<p>The Simulink Design Verifier software does not support variable-size signals. A variable-size signal is a signal whose size (number of elements in a dimension), in addition to its values, can change during model execution.</p> <p>For more information, see “Working with Variable-Size Signals” in <i>Simulink User’s Guide</i>.</p>
Multiword fixed-point data types	<p>The Simulink Design Verifier software does not support multiword fixed-point data types.</p>
Signals with nonzero sample time offset	<p>The Simulink Design Verifier software does not support models with signals that have nonzero sample time offsets.</p>
Nonzero start times	<p>Although Simulink allows you to specify a nonzero simulation start time, the Simulink Design Verifier software generates signal data that begins only at zero. If your model specifies a nonzero start time:</p> <ul style="list-style-type: none"> • If you do not select the Reference input model in generated harness parameter (the default), the harness model is a subsystem. The software sets the start time of the harness model to 1 and continues the analysis. • If you select the Reference input model in generated harness parameter, a Model block references the harness model. The Simulink Design Verifier software cannot change the start time of the harness model, so the analysis stops and you see a recommendation to set the Start time parameter to 0.

Simulink Block Support Limitations

The Simulink Design Verifier software provides various levels of support for Simulink blocks. The software either fully or partially supports particular blocks. It does not support other blocks.

If your model contains unsupported blocks, you can turn on automatic stubbing, which considers the interface of the unsupported blocks, but not their behavior. However, if any of the unsupported blocks affect the simulation outcome, the analysis may achieve only partial results. For details about automatic stubbing, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

To guarantee 100% coverage, avoid using unsupported blocks in models that you analyze with the Simulink Design Verifier software.

Similarly, specify only the block parameters that the Simulink Design Verifier software recognizes for blocks that it partially supports. See Chapter 14, “Simulink Block Support”.

Limitations of Support for Model Reference

The Simulink Design Verifier software supports the Model block, but with the following limitations. The software cannot analyze a model that contains one or more Model blocks if:

- Simulink Design Verifier software does not support protected referenced models. Protected referenced models are encoded to obscure their contents. This feature allows third parties to use the referenced model without being able to view the intellectual property that makes up the model. For more information, see “Protecting Referenced Models” in the *Simulink User’s Guide*.
- The parent model or any of the referenced models gives an error when one of the following model parameters is set to Error:
 - **Diagnostics > Connectivity > Element name mismatch**
 - **Diagnostics > Connectivity > Mux blocks used to create bus signals**

You can use the **Element name mismatch** diagnostic along with bus objects to ensure that your model meets bus element naming requirements imposed by some blocks.

If your model contains Mux blocks that create bus signals, refer to “Tips” in “Mux blocks used to create bus signals” to resolve this problem.

- A referenced model references a variable in its workspace that is either defined in its own workspace or in the base MATLAB workspace, and it is also defined with the same name in the parent model’s workspace. Rename the variable used by the referenced model to a unique name so that you can analyze the model.

Exception: If the parent model and a referenced model both define an instance of a `Simulink.Signal` object used as local data storage with the same name, the software can analyze the model.

- Any of the models in the model reference hierarchy have algebraic loops that cannot be eliminated with algebraic loop minimization. If you encounter this limitation, set the **Minimize algebraic loop** parameter on the **Diagnostics** pane of the Configuration Parameters dialog box to Error. Then, update the model to identify the location of algebraic loop in the model.

To eliminate this problem so that the software can analyze the model, break any algebraic loops with Unit Delay blocks to ensure that the execution order is predictable.

For more information, see “Algebraic Loops” in the *Simulink User’s Guide*.

Unsupported Stateflow Software Features

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

Not Supported	Description
m1 namespace operator, m1 function, m1 expressions	<p>The Simulink Design Verifier software does not support calls to MATLAB functions or access to MATLAB workspace variables, which the Stateflow software allows (see “Using MATLAB Functions and Data in Actions” in the <i>Stateflow and Stateflow® Coder™ User’s Guide</i>).</p>
C math functions	<p>The Simulink Design Verifier software supports calls to the following C math functions: <code>abs</code>, <code>ceil</code>, <code>fabs</code>, <code>floor</code>, <code>fmod</code>, <code>labs</code>, <code>ldexp</code>, and <code>pow</code> (only for an integer exponent).</p> <p>The software does not support calls to other C math functions that the Stateflow software allows. Turning on automatic stubbing allows these functions to be eliminated during the analysis. For details about automatic stubbing, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.</p> <p>For information about C math functions in Stateflow, see “Calling C Functions in Actions” in the <i>Stateflow and Stateflow Coder User’s Guide</i></p>

Not Supported	Description
Recursion	The Simulink Design Verifier software does not support recursive functions, which the Stateflow software allows you to implement using graphical functions (see “Using Graphical Functions to Extend Actions” in the <i>Stateflow and Stateflow Coder User’s Guide</i>). Also, the Simulink Design Verifier software does not support recursion that the Stateflow software allows you to implement using a combination of event broadcasts and function calls.
Custom C or C++ code	The Simulink Design Verifier software does not support custom C or C++ code, which the Stateflow software allows (see “Building Targets” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).
Textual functions with literal string arguments	The Simulink Design Verifier software does not support literal string arguments to textual functions in a Stateflow chart.
Machine-parented data and events	The Simulink Design Verifier software does not support machine-parented data and events (i.e., defined at the level of the Stateflow machine in the Stateflow hierarchy), which the Stateflow software allows (see “Defining Data” and “Defining Events” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).
Absolute-time temporal logic	The Simulink Design Verifier software does not support absolute-time temporal logic, which the Stateflow software allows (see “Operators for Absolute-Time Temporal Logic” in the <i>Stateflow and Stateflow Coder User’s Guide</i>).

Support Limitations for the Embedded MATLAB Subset

In this section...

“Unsupported Embedded MATLAB Subset Features” on page 3-15

“Limitations of Embedded MATLAB Library Function Support” on page 3-16

Unsupported Embedded MATLAB Subset Features

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

Not Supported	Description
Complex numbers	<p>The Simulink Design Verifier software supports only real numbers. The Embedded MATLAB subset also supports complex numbers.</p> <p>For more information, see “Working with Complex Numbers” in the <i>Embedded MATLAB™ User’s Guide</i>.</p>
Characters	<p>The Simulink Design Verifier software does not support characters, which the Embedded MATLAB subset allows.</p> <p>For more information, see “Working with Characters” in the <i>Embedded MATLAB™ User’s Guide</i>.</p>

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

Limitations of Embedded MATLAB Library Function Support

The Simulink Design Verifier software provides various levels of support for Embedded MATLAB library functions. The software either fully or partially supports particular functions. It does not support other functions.

If your model contains unsupported functions, you can turn on automatic stubbing, which considers the interface of the unsupported functions, but not their behavior. However, if any of the unsupported functions affect the simulation outcome, the analysis may achieve only partial results. For details about automatic stubbing, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

To guarantee 100% coverage, avoid using unsupported Embedded MATLAB library functions in models that you analyze with the Simulink Design Verifier software.

Avoid using unsupported Embedded MATLAB library functions in models that you analyze with the Simulink Design Verifier software. See Chapter 15, “Embedded MATLAB Subset Support” for a list of the Embedded MATLAB

library functions for which the Simulink Design Verifier software provides limited or no support.

Fixed-Point Support Limitations

The Simulink Design Verifier software supports fixed-point data types in models that it analyzes, with one exception. Parameter configurations do not support fixed-point data types. For more information about configuring Simulink Design Verifier parameters, see Chapter 5, “Specifying Parameter Configurations”.

For detailed information about these limitations, see “Tunable Expression Limitations” in the *Real-Time Workshop® User’s Guide*.

Working with Block Replacements

- “About Block Replacements” on page 4-2
- “Built-In Block Replacements” on page 4-3
- “Template for Block Replacement Rules” on page 4-6
- “Defining Custom Block Replacements” on page 4-7
- “Executing Block Replacements” on page 4-16

About Block Replacements

Using the Simulink Design Verifier software, you can define rules to replace blocks automatically in your model. For example, you can work around a block that is incompatible with the software 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.

When performing block replacements, the software copies your model and replaces blocks in the copy, without altering your original model. In this way, you can easily customize a model for analysis.

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

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

You replace any block with any built-in block, library block, or subsystem.

Block replacements are extensible, allowing you to define your own libraries of replacement blocks and custom block replacement rules. Use this capability 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, objectives to its output signals, or eliminating the contents of a subsystem or Model block to simplify your analysis.

Note You can use automatic stubbing as an alternative to block replacements. Automatic stubbing replaces unsupported blocks with elements that have the same interface. For more information, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

Built-In Block Replacements

The Simulink Design Verifier software provides a set of block replacement rules and a corresponding library of replacement blocks. Use these built-in block replacements when analyzing models. 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` folder. There are 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.

File Name	Description
blkrep_rule_lookup_normal.m blkrep_rule_lookup_configss.m	A rule that replaces Lookup Table blocks with an implementation that includes test objectives for each breakpoint and interval specified by the Vector of input values parameter.
blkrep_rule_lookup2D_normal.m blkrep_rule_lookup2D_configss.m	A rule that adds Test Condition/Proof Assumption blocks to the input ports of Lookup Table (2-D) blocks. Each Test Condition/Proof Assumption block constrains signal values to the interval specified by the corresponding breakpoint vector.
blkrep_rule_mpswitch2_normal.m blkrep_rule_mpswitch2_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter is 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).

File Name	Description
blkrep_rule_mpswitch3_normal.m blkrep_rule_mpswitch3_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter is 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).
blkrep_rule_mpswitch4_normal.m blkrep_rule_mpswitch4_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter is 4. The Test Condition/Proof Assumption block constrains signal values to the interval [1, 4] (or [0, 3] if the block uses zero-based indexing).
blkrep_rule_mpswitch5_normal.m blkrep_rule_mpswitch5_configss.m	A rule that adds a Test Condition/Proof Assumption block to the control input port of Multiport Switch blocks whose Number of inputs parameter is 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 that each switch position be exercised when the values of the first and third input ports are different.

File Name	Description
blkrep_rule_selector IndexVecPort_normal.m blkrep_rule_selector IndexVecPort_configss.m	A rule that adds a Test Condition/Proof Assumption block to the index port of Selector blocks whose Index Option parameter is Index vector (port) . The Test Condition/Proof Assumption block constrains signal values to an interval whose endpoints are derived from the values of the Selector block's Input port size and Index mode parameters.
blkrep_rule_selector StartingIdxPort_normal.m blkrep_rule_selector StartingIdxPort_configss.m	A rule that adds a Test Condition/Proof Assumption block to the index port of Selector blocks whose Index Option parameter is Starting index (port) . The Test Condition/Proof Assumption block constrains signal values to an interval whose endpoints are derived from the values of the Selector block's Input port size , Output size , and Index mode parameters.

You can find the library of replacement blocks that corresponds to the factory default rules at:

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

Template for Block Replacement Rules

To help you create block replacement rules, the Simulink Design Verifier software provides an annotated M-file template that contains a skeleton implementation of the requisite callbacks. You can find the template at:

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

To create a block replacement rule, make a copy of the template and edit the copy for the behavior that you want for the rule you are creating. For information on how to implement your rule, see the comments in the template. For information on using the template to write custom block replacements rules, see “Writing Block Replacement Rules” on page 4-8.

Defining Custom Block Replacements

In this section...

“Basic Workflow for Defining Custom Block Replacements” on page 4-7

“Specifying Replacement Blocks” on page 4-7

“Writing Block Replacement Rules” on page 4-8

“Example: Replacing Multiport Switch Blocks” on page 4-8

Basic Workflow for Defining Custom Block Replacements

To replace certain blocks in your model in a way that the factory-default block replacement rules do not handle, create custom block replacement rules by completing the following tasks:

- “Specifying Replacement Blocks” on page 4-7
- “Writing Block Replacement Rules” on page 4-8

Specifying Replacement Blocks

A replacement block can be one of the built-in blocks in the Simulink model library or a block in a user-created library.

In the Simulink Design Verifier software, replacement blocks have the following restrictions:

- They must be built-in blocks or subsystems.
- They cannot be Model blocks, nor can they include any Model blocks.

Note A Model block cannot be a replacement block, but you can replace Model blocks with built-in blocks or subsystems.

- They must reside in a block library that is available on your MATLAB search path.

- If the replacement block is a subsystem, any Inport and Outport blocks *must* have the default names (In1 and Out1).

After constructing your replacement block, write a custom block replacement rule.

Writing Block Replacement Rules

In the Simulink Design Verifier software, block replacement rules have the following restrictions:

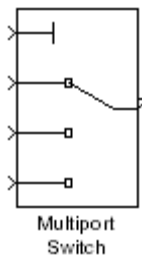
- The M-file that represents a block replacement rule must include particular callbacks. The MathWorks recommends that you use the block replacement rule template as a starting point for writing a custom rule. (See “Template for Block Replacement Rules” on page 4-6.)
- The M-file that represents a block replacement rule must be on the MATLAB search path.

Example: Replacing Multiport Switch Blocks

- “Why Replace Multiport Switch Blocks?” on page 4-8
- “Creating the Library and Replacement Block” on page 4-9
- “Writing the Rule for the Replacement Block” on page 4-12

Why Replace Multiport Switch Blocks?

A Multiport Switch block has one control input port and one or more data input ports; the default number of data inputs is 3.



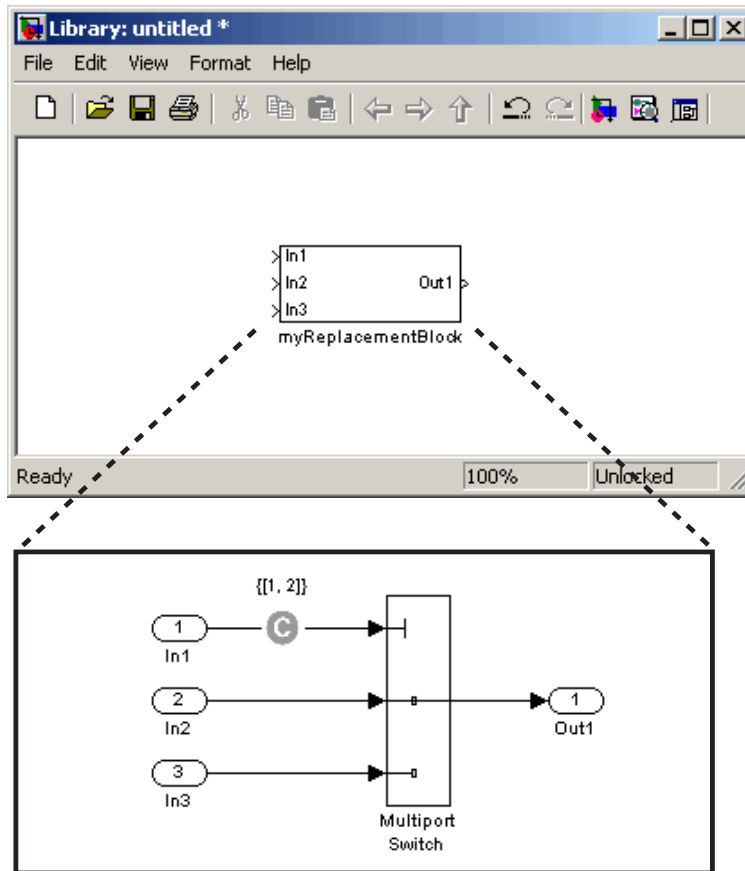
A model may have test objectives on some blocks whose output is directly or indirectly connected to the Multiport Switch block. For example, a Saturation block may send data to the control input port. If so, the analysis may create test cases that satisfy those objectives. However, those test cases may create values that are out of range for the control input port, regardless of whether the Multiport Switch block uses zero-based indexing or one-based indexing. This causes the simulation to fail.

In this example, you create a rule to replace all Multiport Switch blocks that have two data inputs and do not use zero-based indexing. The replacement block is a subsystem that has a Test Condition block that constrains the value of the control input to 1 or 2, so that the analysis does not create out-of-range data input values. This allows the analysis to satisfy the objectives on blocks that are connected to the control input port of the Multiport Switch block.

Creating the Library and Replacement Block

To create a user library and specify a replacement block as a masked subsystem:

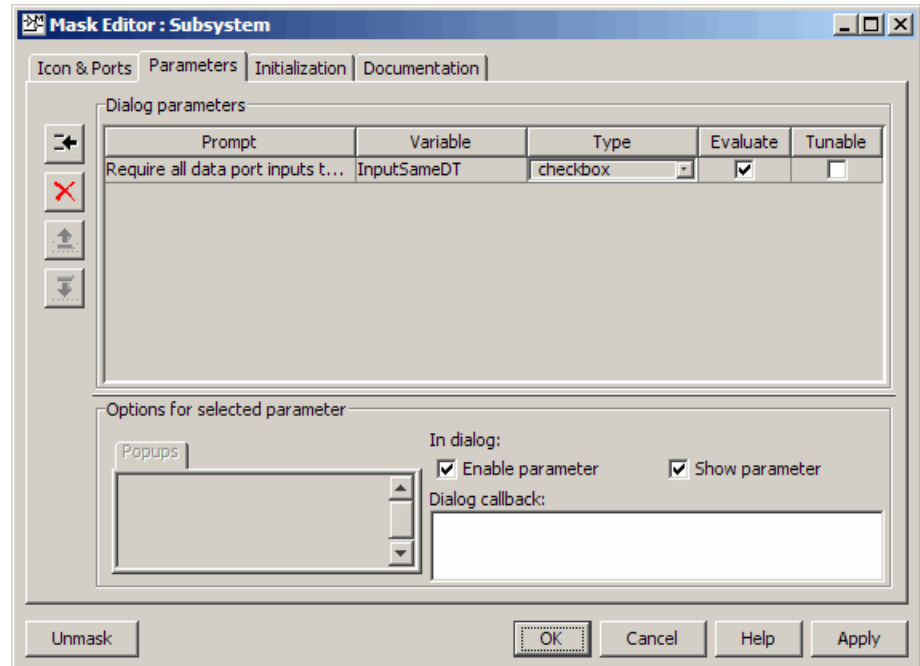
- 1 In the Simulink Library Browser, select **File > New > Library**.
- 2 In your library, create a subsystem named `myReplacementBlock` to represent your replacement block. It should look like the following graphic, with several parameters set:
 - In the Multiport Switch block, set the **Number of inputs** parameter to 2.
 - In the Test Condition block, set the **Values** parameter to `{[1, 2]}`.



- 3 To create a mask for your subsystem, select the subsystem, right-click, and select **Mask subsystem** from the context menu. For information and a tutorial for creating block masks, see “Working with Block Masks” in the *Simulink User’s Guide*.

For this example, 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 you create mask parameters that control the behavior of parameters associated with their underlying blocks, specify actual parameter names as dialog box 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; therefore, it specifies the name of the dialog box variable in this example.

- In the **Initialization** pane, in the **Initialization commands** field, the following commands specify that the subsystem inherit the `InputSameDT` parameter value of the top-level model:

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

- 4 Save your block library as `custom_rule.mdl` in a folder on your MATLAB search path.

Writing the Rule for the Replacement Block

To write a rule for the replacement block:

- 1 Copy the block replacement rule template

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

- 2 Save it as `custom_rule_switch.m`.

Note For steps 2 through 10, edit the copy of the template that you saved.

- 3 Rename the function, as defined on the first line of the M-file. The function name must match 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:

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

- 4 Identify the type of block that you want 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 that you want to replace. Alternatively, you can

determine this value by referring to “Block-Specific Parameters” in the *Simulink Reference*.

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

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

- 5 Identify the replacement block by specifying its full block path name as the `rule.ReplacementPath` object. Consider using the `gcb` function as a way to get the full block path name.

This example replaces Multiport Switch blocks with the replacement block developed in “Specifying Replacement Blocks” on page 4-7. The `rule.ReplacementPath` object specifies the full block path name:

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

- 6 Identify the type of subsystem that the Simulink Design Verifier software uses when replacing blocks by specifying a value for the `rule.ReplacementMode` object. Valid values are:
 - `Normal` — When using this rule, the software replaces blocks with a copy of the subsystem specified by the `rule.ReplacementPath` object.
 - `ConfigurableSubSystem` — When using this rule, the software replaces blocks with a Configurable Subsystem block. With the Configurable Subsystem block, you can 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';
```

- 7 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 `$original.parameter$`. `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;
```

- 8 To define the callback functions, keep the following lines in the file:

```
rule.IsReplaceableCallback = @replacementTestFunction;
.
.
.
rule.PostReplacementCallback = @postReplacementFunction;
```

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

The following example instructs the Simulink Design Verifier software to replace only the Multiport Switch blocks whose `NumInputPorts` parameter is 2 and whose `zeroidx` parameter is off:

```
function out = replacementTestFunction(blockH)
% Specify the logic that determines when the Simulink Design
% Verifier software replaces a block in your model. For example,
% restrict replacements to only the blocks whose parameters
% specify particular values.
```



```
out = false;
numInputPorts = eval(get_param(blockH,'NumInputPorts'));
zeroIdx = eval(get_param(blockH,'zeroIdx'));
if numInputPorts==2 && zeroIdx=='off',
    out = true;
end
```

- 10** Optionally, you can customize the `postReplacementFunction` subfunction to specify the actions the software performs after a block has been replaced. For an example of a `postReplacementFunction` subfunction, open the following file:

```
matlabroot/toolbox/sldv/sldv/blkrep_rule_selectorIndexVecPort_normal.m
```

- 11** Save the edited M-file.

Executing Block Replacements

In this section...
“Configuring Block Replacements” on page 4-16
“Replacing Blocks in a Model” on page 4-17

Configuring Block Replacements

You must configure block replacement options before executing block replacements in your model. To specify block replacement options from the model window:

- 1 Open the `sldvdemo_cruise_control` model.
- 2 Save a copy of this model and name it `my_sldvdemo_cruise_control`.
- 3 From the **Tools** menu of your Simulink model, select **Design Verifier > Options**.

The Configuration Parameters dialog box displays the main pane of the **Design Verifier** category.

- 4 In the **Select** tree of the Configuration Parameters dialog box, click the **Block Replacements** category.
- 5 On the Block Replacements pane, select **Apply block replacements** to enable block replacements.

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

- 6 In the **List of block replacement rules (in order of priority)** box, enter file names for the block replacement rules that you want to execute. The default value, `<FactoryDefaultRules>`, executes all the factory default rules.

You can specify multiple rules as a list delimited by spaces, commas, or carriage returns. The software executes the rules in the order that you list them. For example, to execute two of the factory default rules followed

by the custom block replacement example from “Defining Custom Block Replacements” on page 4-7, enter the following file names:

```
blkrep_rule_mpswitch4_normal  
blkrep_rule_lookup_normal  
custom_rule_switch
```

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

- 7** After applying the block replacement rules, in the **File path of the output model** box, enter a file name for the model that results. Optionally, include a path that is either absolute or relative to the path in the **Output directory** on the main **Design Verifier** pane.
- 8** Click **OK** to apply the changes and close the Configuration Parameters dialog box.
- 9** Save the `my_sldvdemo_cruise_control` model.

Alternatively, at the command line, you can use the `sldvoptions` function to specify the block replacement options associated with a Simulink Design Verifier options object, described in “Replacing Blocks in a Model” on page 4-17.

Replacing Blocks in a Model

After enabling the **Apply block replacements** option, you can execute block replacements in your model by starting a Simulink Design Verifier analysis. For example, to trigger block replacements and start the analysis from the Configuration Parameters dialog box, on the **Design Verifier** pane, click **Generate Test Cases**.

Note The Simulink Design Verifier software can execute block replacements only on models that do not have any saved changes.

When you are replacing blocks, the Simulink Design Verifier software copies your model and replaces blocks in the copy, without altering your original model. Upon completing its analysis, the software generates a report that displays information about the block replacements it executed.

When you enable block replacements and start the analysis, the software invokes the `sldvblockreplacement` command to copy the model and perform the block replacements, before continuing the analysis.

Alternatively, to perform only the block replacements, at the command line or from an M-file program, you can use the `sldvblockreplacement` function. You pass a handle to the model and the `sldvoptions` structure as follows:

```
opts = sldvoptions;
opts.BlockReplacement = 'on'
opts.BlockReplacementRulesList = ...
'<FactoryDefaultRules>, custom_rule_switch';
[status, newmodelH] = sldvblockreplacement('modelH', opts);
```

`modelH` is a handle to the model whose blocks you want to replace. `newmodelH` is the handle to the new model with the block replacements.

Replacing the blocks in a model before running the analysis can help you debug the custom block replacement rules. Once the block replacement rules are working as you want, you can analyze the model that contains the block replacements.

See `sldvblockreplacement` for more information.

If you execute block replacements programmatically, in the MATLAB Command Window, the Simulink Design Verifier software displays 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 (in order of priority)** option (see “Configuring Block Replacements” on page 4-16). The columns of the preceding table identify the following information:

- Type — Type of rule, either built-in or custom
- Registration M-File name — Name of the M-file that expresses the rule
- Block types — `BlockType` parameter value of the block that the rule replaces
- Priority — 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)

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

Performed block replacements:

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


Specifying Parameter Configurations

- “About Parameter Configurations” on page 5-2
- “Template for Parameter Configurations” on page 5-3
- “Defining Parameter Configurations” on page 5-4
- “Parameter Configuration Example” on page 5-7

About Parameter Configurations

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

- Extend the results of a property 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 5-7).

Template for Parameter Configurations

To help you create a parameter configuration file, the Simulink Design Verifier software provides an annotated M-file template:

```
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 6-9).

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 5-4.

Defining Parameter Configurations

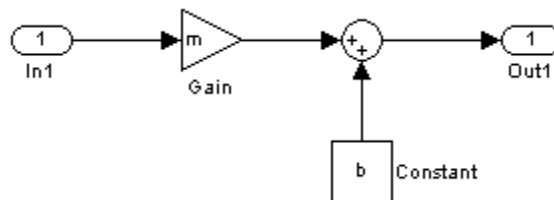
This section describes how to define parameter configurations and outlines the required syntax for their definition.

1 Define parameter configurations in an M-file function.

The Simulink Design Verifier software provides an annotated template for an M-file function that you can use as a starting point (see “Template for Parameter Configurations” on page 5-3).

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

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



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

```
params.m  
params.b
```

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

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

- '()' — Defines an open interval.

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

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

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

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

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

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

Note To indicate no constraint for an input parameter, specify `params.m = {}` or `params.m = []` in the M-file function, or omit the declaration. The Simulink Design Verifier software treats this parameter as free input and uses random parameter values.

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

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

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

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

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

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

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

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

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

Parameter Configuration Example

In this section...
“About This Example” on page 5-7
“Constructing the Example Model” on page 5-8
“Parameterizing the Constant Block” on page 5-10
“Specifying a Parameter Configuration” on page 5-11
“Analyzing the Example Model” on page 5-13
“Simulating the Test Cases” on page 5-15

About This Example

The next five tasks describe how to create and analyze a simple Simulink model, for which you generate test cases that achieve decision coverage. However, in this example, achieving complete decision coverage is possible only when the Simulink Design Verifier software treats a particular block parameter as a variable during its analysis. Toward that end, this example explains how to specify parameter configurations for use with the Simulink Design Verifier software.

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

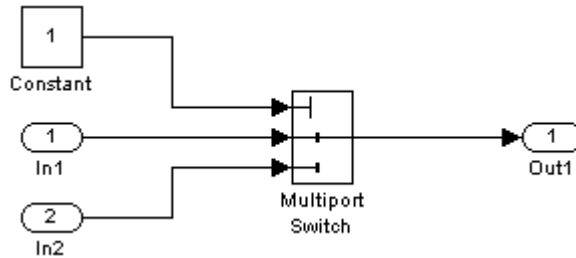
Task	Description	See...
1	Construct the example model.	“Constructing the Example Model” on page 5-8
2	Specify a variable as the value of a Constant block parameter.	“Parameterizing the Constant Block” on page 5-10
3	Constrain the value of the variable that the Constant block specifies.	“Specifying a Parameter Configuration” on page 5-11

Task	Description	See...
4	Generate test cases for your model and interpret the results.	“Analyzing the Example Model” on page 5-13
5	Simulate the test cases and measure the resulting decision coverage.	“Simulating the Test Cases” on page 5-15

Constructing the Example Model

In this task, you construct a simple Simulink model that you use throughout the remaining tasks.

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

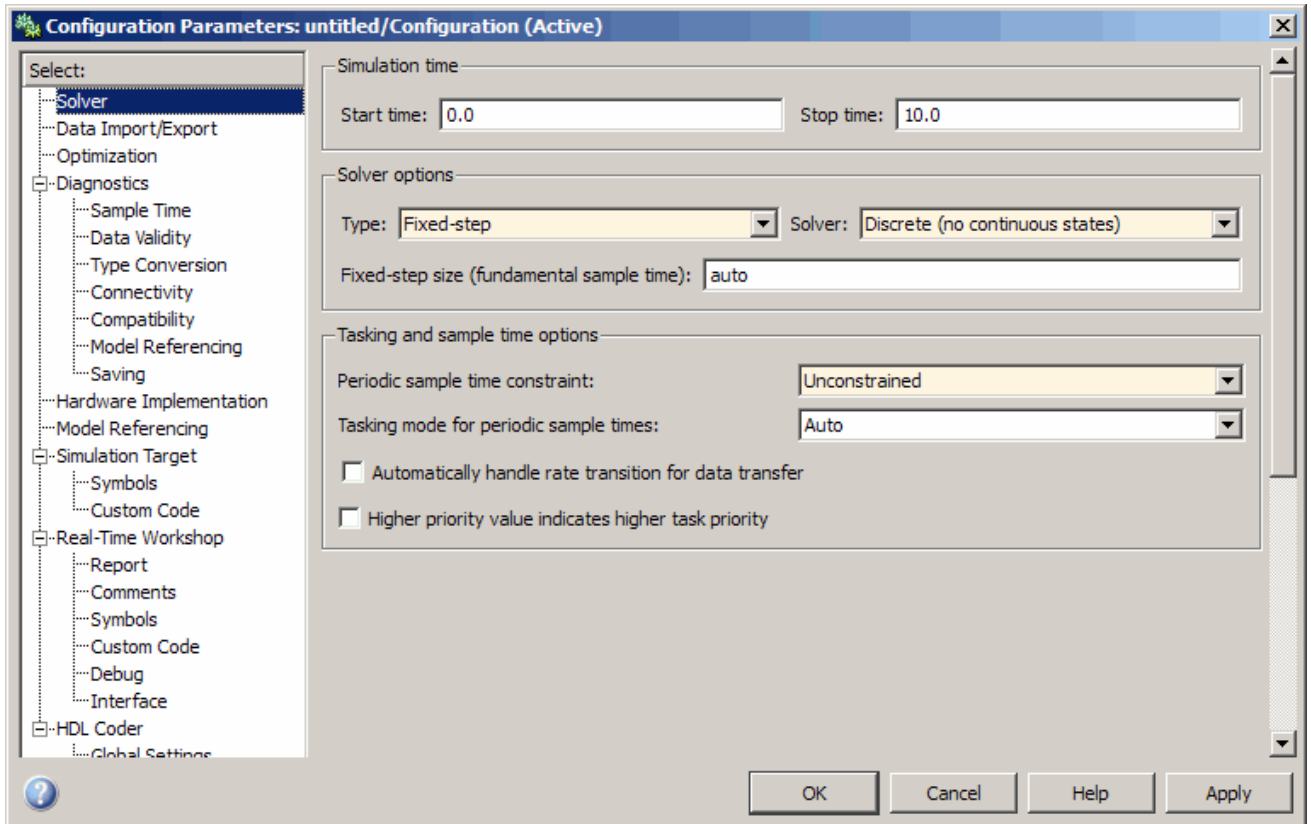


5 In your model window, select **Simulation > Configuration Parameters**.

The Configuration Parameters dialog box appears.

6 In the **Select** tree on the left side of the Configuration Parameters dialog box, click the **Solver** category. 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 looks as follows:



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

8 Save your model as `param_example.mdl` for use in the next step.

Parameterizing the Constant Block

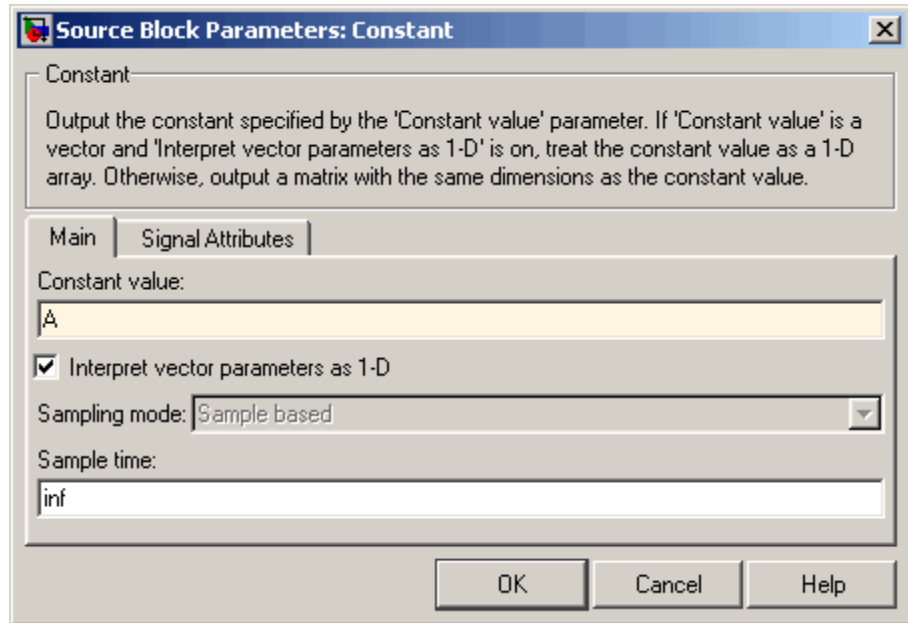
In this task, you parameterize the Constant block in your model. In particular, you specify a variable as the value of the Constant block's **Constant value** parameter.

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 should look as follows.



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

- 4 In the MATLAB Command Window, enter

```
A = 1;
```

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

- 5 Save your model for use in the next step.

Specifying a Parameter Configuration

In this task, you customize the parameter configuration file template so that it constrains the variable A.

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

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

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

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

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

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

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

```
function params = params_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 folder as the example model.
- 6** Close the MATLAB Editor.
- 7** In the Configuration Parameters dialog box, click **Browse** next to the **Parameter configuration file** option, and then select your parameter configuration file, `params_example_function.m`.
- 8** Click **Apply** and **OK** to apply your change and close the Configuration Parameters dialog box.

- 9 Save your model for use in the next step.

Analyzing the Example Model

In this task, you execute the Simulink Design Verifier analysis using the parameter configuration file you just created. The software generates test cases and produces results for you to interpret.

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

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

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

- 2 In the Simulink Design Verifier report **Table of Contents**, click **Test Cases**.
- 3 Click **Test Case 1** to display the subsection for that test case.

Test Case 1

Summary

Length: 0 Seconds (1 sample periods)
Objective Count: 1

Objectives

Step	Time	Model Item	Objectives
1	0	Multiport Switch	truncated input value = 1 (output is from input port 2)

Generated Parameter Values

Parameter	Value
A	1

Generated Input Data

Time	0
Step	1
In1	-
In2	-

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

- 4** Scroll down to the Test Case 2 section in the **Test Cases** chapter.

Test Case 2

Summary

Length: 0 Seconds (1 sample periods)

Objective Count: 1

Objectives

Step	Time	Model Item	Objectives
1	0	Multiport Switch	truncated input value = 2 (output is from input port 3)

Generated Parameter Values

Parameter	Value
A	2

Generated Input Data

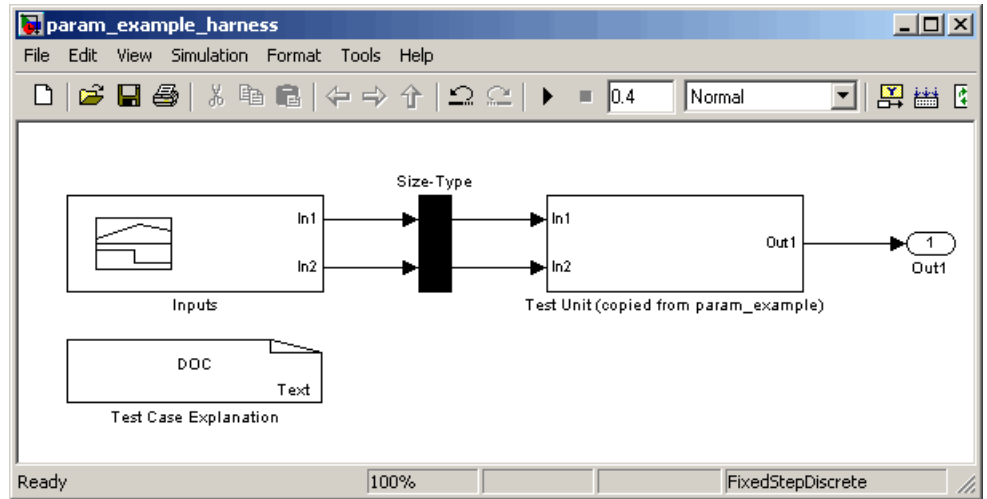
Time	0
Step	1
In1	-
In2	-

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.

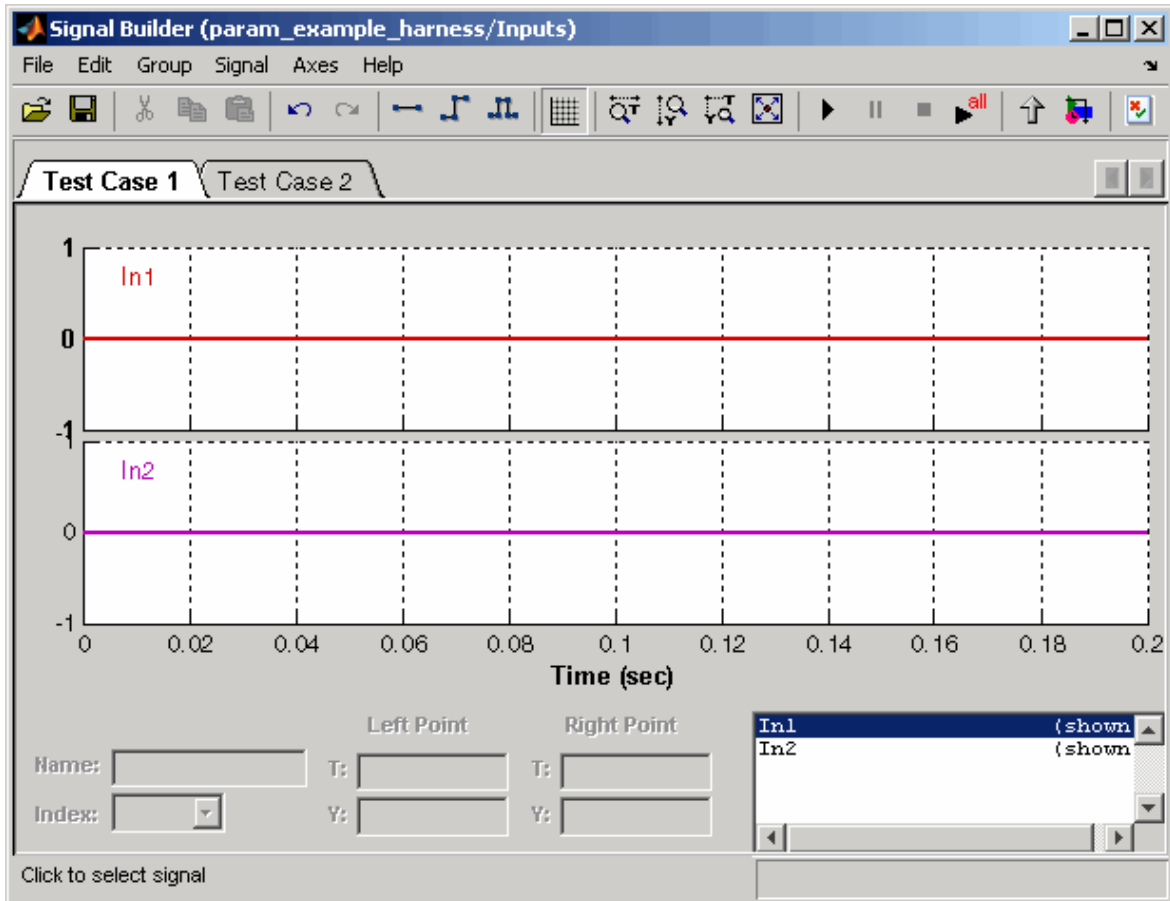
Simulating the Test Cases


In this final task, you simulate the test cases that the Simulink Design Verifier software generated in “Simulating the Test Cases” on page 5-15. In addition, you review the coverage report that results from the simulation.

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





- 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.



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

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

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

Summary	
Model Hierarchy/Complexity:	Test 1
	D1
1. param_example_harness1	2 100% 
2. . . . Test Unit (copied from param_example)	1 100% 

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

5 In the **Summary** section, 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_harness1](#)

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

MultiPortSwitch block "Multiport Switch"

Parent: [param_example_harness1/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)	2/4
= 2 (output is from input port 3)	2/4

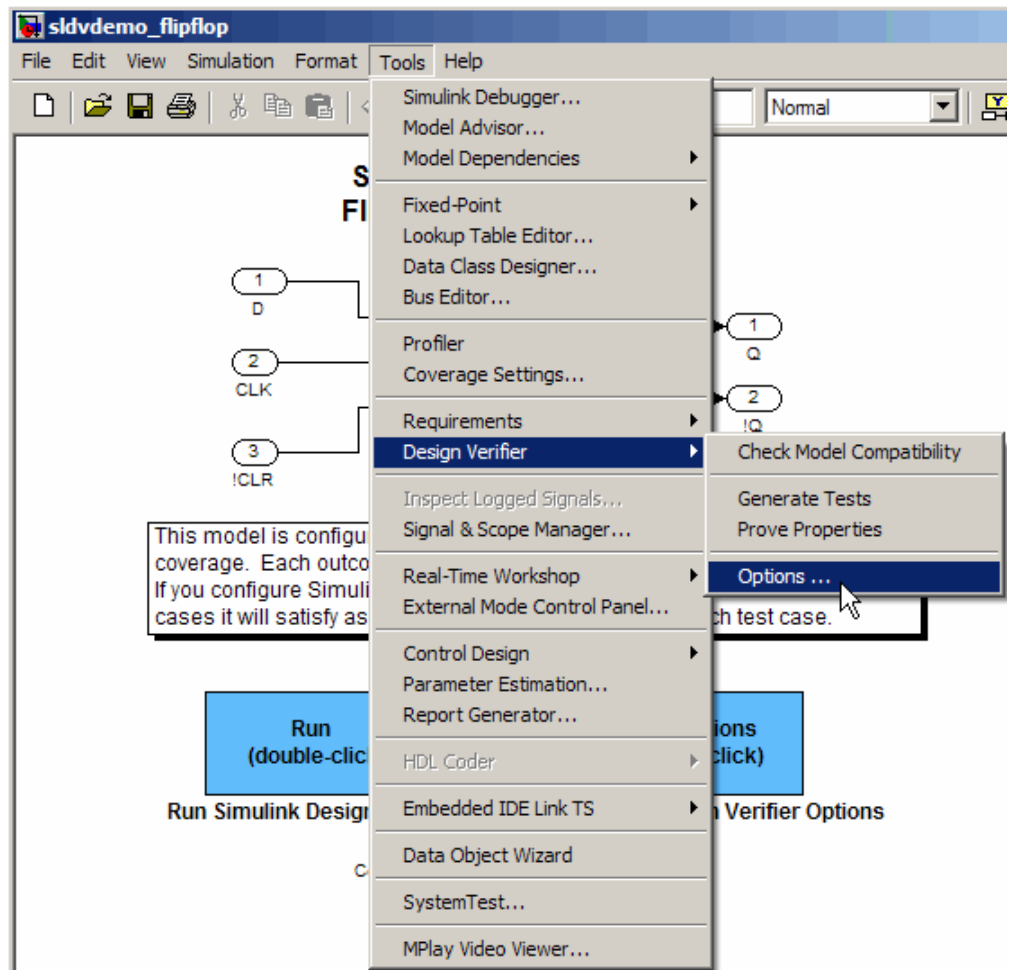
This section reveals that the Multiport Switch block achieves complete decision coverage because the test cases exercise each of its switch pathways.

Configuring Simulink Design Verifier Options

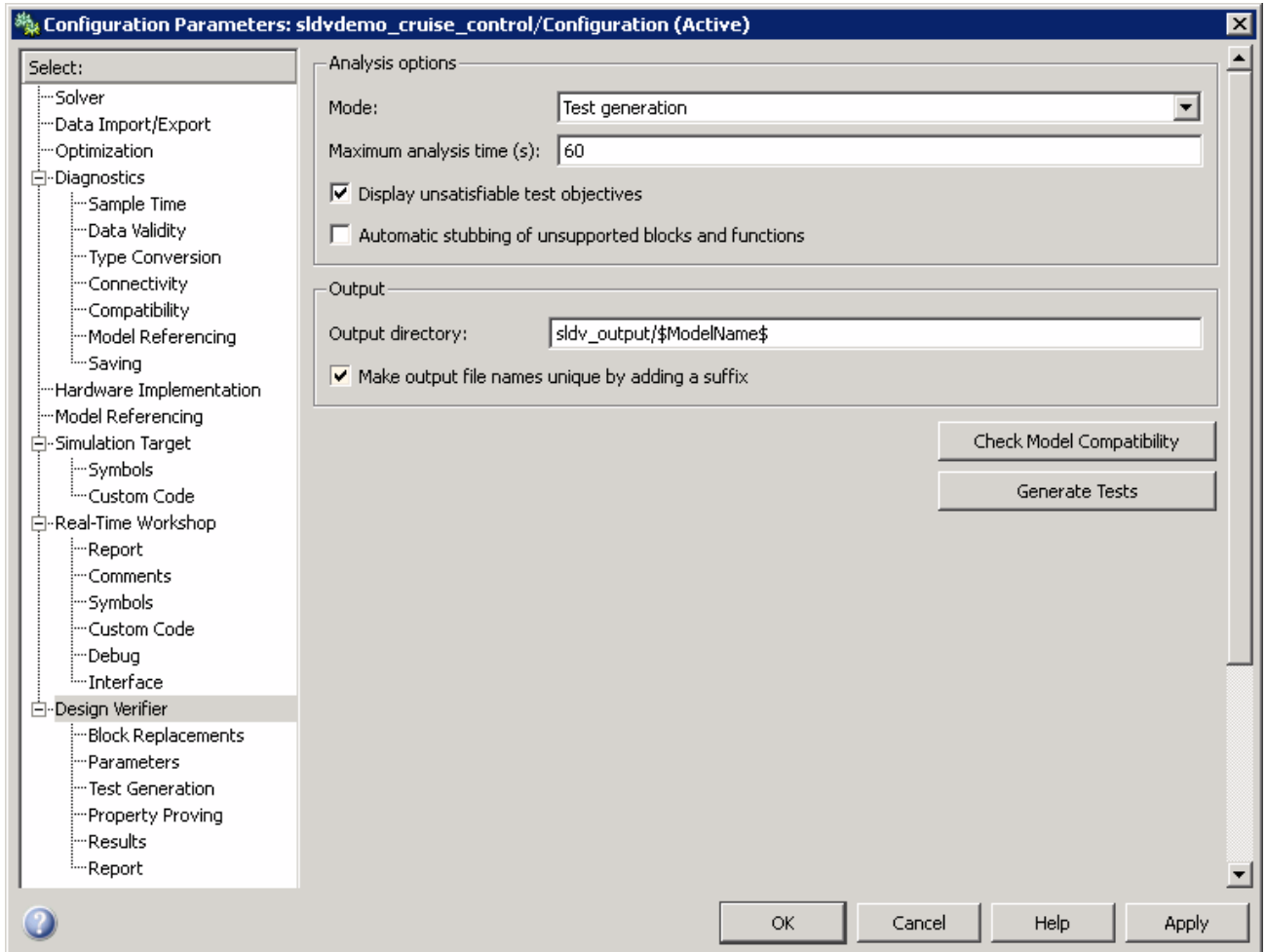
- “Viewing Simulink® Design Verifier Options” on page 6-2
- “Configuring Simulink® Design Verifier Options” on page 6-5
- “Saving Simulink® Design Verifier Options” on page 6-20

Viewing Simulink Design Verifier Options

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



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



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

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

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

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

See `sldvoptions` in Chapter 11, “Function Reference” for more information.

Configuring Simulink Design Verifier Options

In this section...

“Design Verifier Pane” on page 6-5

“Block Replacements Pane” on page 6-7

“Parameters Pane” on page 6-9

“Test Generation Pane” on page 6-10

“Property Proving Pane” on page 6-12

“Results Pane” on page 6-14

“Report Pane” on page 6-17

Design Verifier Pane

In the **Design Verifier** pane, you specify analysis options and configure Simulink Design Verifier output.

The screenshot shows the Design Verifier pane with two main sections: Analysis options and Output. In the Analysis options section, the Mode is set to Test generation, the Maximum analysis time (s) is 600, and the checkboxes for Display unsatisfiable test objectives and Automatic stubbing of unsupported blocks and functions are checked. In the Output section, the Output directory is set to sldv_output/\$ModelName\$, and the checkbox for Make output file names unique by adding a suffix is checked.

Analysis options	
Mode:	Test generation
Maximum analysis time (s):	600
<input checked="" type="checkbox"/> Display unsatisfiable test objectives	
<input type="checkbox"/> Automatic stubbing of unsupported blocks and functions	

Output	
Output directory:	sldv_output/\$ModelName\$
<input checked="" type="checkbox"/> Make output file names unique by adding a suffix	

The **Design Verifier** pane contains the following settings:

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

- “Check Model Compatibility” on page 6-7
- “Generate Tests or Prove Properties” on page 6-7

Analysis options

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

Mode. Specifies the mode in which the Simulink Design Verifier software operates— **Test generation** (the default) or **Property proving**. Depending on the value of this parameter, if you want to start an analysis, click the **Generate Tests** or **Prove Properties** button on this pane.

Maximum analysis time. Specifies the maximum time (in seconds) that the Simulink Design Verifier software spends analyzing the model. The default value is 600 seconds.

Display unsatisfiable test objectives. If you select this option, it causes the Simulink Design Verifier software to display a warning message in the Simulation Diagnostics Viewer when it cannot satisfy a test objective.

Tip If you first select **Display unsatisfiable test objectives**, set the **Test suite optimization** option to the **Combined objectives** strategy and analyze the model. If that test returns objectives without outcomes, select the **Individual objectives** strategy and reanalyze the model. The **Individual objectives** strategy analyzes each objective independently and more accurately identifies unsatisfiable objectives.

Automatic stubbing of unsupported blocks and functions. If you select this option, it specifies that the Simulink Design Verifier software ignores unsupported blocks and functions, and proceeds with the analysis.

Output

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

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

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

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

Check Model Compatibility

Click **Check Model Compatibility** to see if your model is compatible with the Simulink Design Verifier software. If you are setting options for a subsystem that you selected, click **Check Subsystem Compatibility**.

Generate Tests or Prove Properties

In the Configuration Parameters dialog box, click **Generate Tests** or **Prove Properties** to analyze a model.

If you set the **Mode** parameter to **Test generation**, click **Generate Tests** to begin a test-case generation analysis of the model. If you are setting options for a subsystem that you selected, click **Generate Tests for Subsystem**.

If you set the **Mode** parameter to **Property proving**, click **Prove Properties** to begin property-proving analysis of the model. If you are setting options for a subsystem that you selected, click **Prove Properties of Subsystem**.

Block Replacements Pane

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

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

Block replacements

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

Apply block replacements. If selected, this option causes the Simulink Design Verifier software to replace blocks in the model before its analysis (see Chapter 4, “Working with Block Replacements”). By default, this option is disabled. Enabling this option provides access to the **List of block replacement rules** and **File path of the output model** options.

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

Specify block replacement rules as a list delimited by spaces, commas, or carriage returns (see “Configuring Block Replacements” on page 4-16).

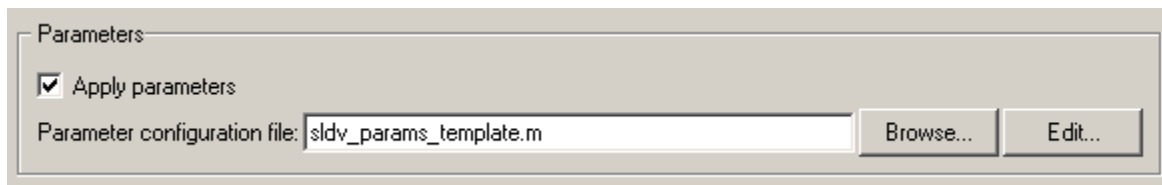
The default value is <FactoryDefaultRules>. If you specify the default value, the Simulink Design Verifier software uses its factory default block replacement rules (see “Built-In Block Replacements” on page 4-3).

File path of the output model. Specifies a folder for the model that results after applying the block replacement rules. Enter a path name that is either absolute or relative to the path name specified as the **Output directory**. This option is accessible only if **Apply block replacements** is selected.

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

Parameters Pane

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



Parameters

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

Apply parameters. If selected (the default), this option causes the Simulink Design Verifier software to use parameter configurations when analyzing a model (see Chapter 5, “Specifying Parameter Configurations”). Enabling this option provides access to the **Parameter configuration file** option.

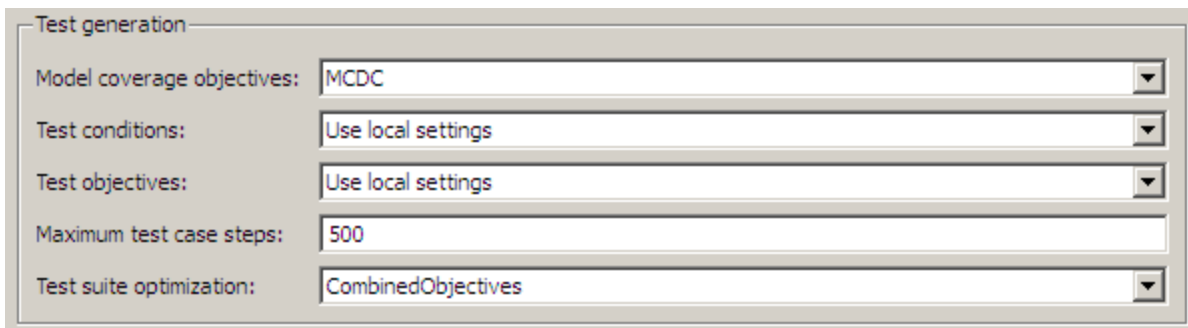
Parameter configuration file. Specifies an M-file function that defines parameter configurations for a model. Click the **Browse** button to select an existing M-file function using a file chooser dialog box. Click the **Edit** button to open the specified M-file function in an editor.

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

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

Test Generation Pane

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



The screenshot shows a 'Test generation' pane with the following settings:

Property	Value
Model coverage objectives:	MCDC
Test conditions:	Use local settings
Test objectives:	Use local settings
Maximum test case steps:	500
Test suite optimization:	CombinedObjectives

Test generation

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

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

When you set **Model coverage objectives** to MCDC, the Simulink Design Verifier software automatically enables every coverage objective for decision coverage and condition coverage as well. Similarly, enabling coverage for condition coverage causes every decision and condition coverage outcome to be

enabled. Each Simulink Design Verifier coverage objective includes all the objectives in a less-strict coverage metric.

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

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

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

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

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

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

- **Combined objectives** — Minimizes the number of test cases in a suite by generating test cases that address more than one test objective. Each test case tends to be long, i.e., it includes many time steps.

This option does not necessarily find unsatisfiable objectives, and often leaves them undecided. To identify unsatisfiable objectives, first, run the **Combined objectives** strategy to generate test cases. If the analysis returns objectives without outcomes, set the optimization strategy to **Individual objectives** and rerun the analysis to identify any unsatisfiable objectives.

- **Individual objectives** — Maximizes the number of test cases in a suite by generating test cases that each address only one test objective. Each test case tends to be short, i.e., it includes only a few time steps.

Since each test case is analyzed independently, use this strategy to find unsatisfiable objectives.

- **Large model** — Minimizes the number of test cases in a suite by generating cases that address more than one test objective. This strategy is tailored for large models that contain nonlinearities and numerous test objectives; consequently, it tends to use all the time that the **Maximum analysis time** option allots.
- **Long test cases** — Combines test cases to create a smaller number of test cases. This strategy generates fewer, but longer, test cases that each satisfy multiple test objectives and creates a more efficient analysis and easier-to-review results.

Property Proving Pane

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



The screenshot shows a 'Property proving' pane with the following settings:

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

Property proving

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

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

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

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

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

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

- **Find violation** — If this strategy is selected, the Simulink Design Verifier software searches for property violations within the number of simulation steps specified by the **Maximum violation steps** option. Enabling this option provides access to the **Maximum violation steps** option.

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

See “Techniques for Proving Properties of Large Models” on page 10-20.

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

Results Pane

In the **Results** pane, you specify options that control how the Simulink Design Verifier software handles the results that it generates.

The screenshot shows a configuration dialog box with three sections:

- Data file options:**
 - Save test data to file
 - Data file name:
 - Include expected output values
 - Randomize data that do not affect the outcome
- Harness model options:**
 - Save test harness as model
 - Harness model file name:
 - Reference input model in generated harness
- SystemTest options:**
 - Save test harness as SystemTest TEST-file (will reference saved data file)
 - SystemTest file name:

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

- “Data file options” on page 6-15
- “Harness model options” on page 6-16
- “SystemTest options” on page 6-17

Data file options

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

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

Data file name. Specifies a file name for the MAT-file containing the generated test data. Enter a path name that is either absolute or relative to the folder specified by **Output directory**. This option is accessible only if **Save test data to file** is selected.

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

Include expected output values. If selected, this option causes the Simulink Design Verifier software to simulate the model using the test case signals that it produces. For each test case, the software collects the simulation output values associated with Output blocks in the top-level system and includes those values in the MAT-file that it generates (see “TestCases Field / CounterExamples Field” on page 9-5).

Randomize data that does not affect outcome. If selected, this option causes the Simulink Design Verifier software to assign random values instead of zeros to test case or counterexample signals that have no impact on test or proof objectives in a model. In the Simulink Design Verifier report, the Generated Input Data table always displays a dash (–) for such signals (see “Test Cases / Properties Chapter” on page 9-30).

Harness model options

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

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

Harness model file name. Specifies a file name for the test harness model. Enter a path name that is either absolute or relative to the path name specified by **Output directory**. This option is accessible only if **Save test harness as model** is selected.

The default value is `$ModelName$_harness`. `$ModelName$` is a token that represents the model name.

Reference input model in generated harness. If selected, this option causes the Simulink Design Verifier software to use model reference to run the input model in the generated test harness instead of inserting a copy of the input model.

SystemTest options

Save test harness as SystemTest TEST-file (will reference saved data file). If selected, this option causes the Simulink Design Verifier software to produce the .test configuration file for running generated test cases inside the SystemTest™ environment. Enter a path name that is either absolute or relative to the path name specified by **Output directory**. Enabling this option provides access to the **SystemTest file name** option.

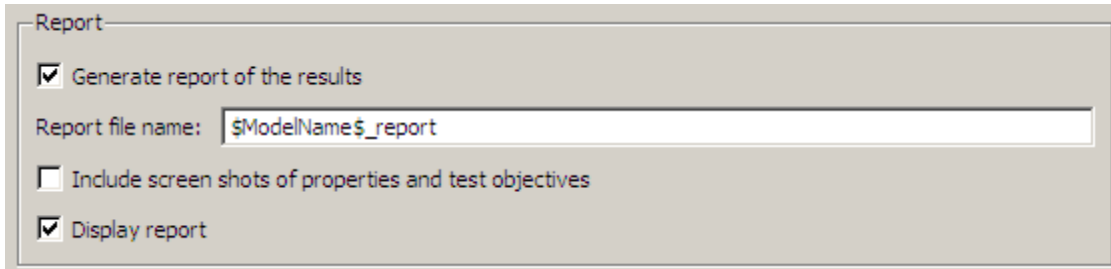
Note The option to create a SystemTest TEST-file is only available in test-generation mode; you cannot create this file when running a property-proving analysis.

SystemTest file name. Specifies a file name for the SystemTest TEST-file. Enter a path name that is either absolute or relative to the path name specified by **Output directory**. This option is accessible only if the **Save test harness as SystemTest TEST-file (will reference saved data file)** is selected.

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

Report Pane

In the **Report** pane, you specify options that control how the Simulink Design Verifier software reports its results.



The screenshot shows a dialog box titled "Report" with a light gray background. It contains four controls: a checked checkbox for "Generate report of the results", a text input field for "Report file name:" containing the text "\$ModelName\$_report", an unchecked checkbox for "Include screen shots of properties and test objectives", and a checked checkbox for "Display report".

Report

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

Generate report of the results. If selected, this option causes the Simulink Design Verifier software to save the HTML report it generates. If you select this option, you must also enable the **Save test harness as model** option (see “Harness model options” on page 6-16).

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

Report file name. Specifies a file name for the HTML report. Enter a path name that is either absolute or relative to the folder specified by **Output directory**. This option is accessible only if **Generate report of the results** is selected.

The default value is \$ModelName\$_report. \$ModelName\$ is a token that represents the model name.

Include screen shots of properties and test objectives. If selected, this option causes the Simulink Design Verifier software to insert a screen shot of each property to the corresponding section of the HTML report it generates. This option is only valid in property-proving mode. This option is disabled by default. It is accessible only if **Generate report of the results** is selected.

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

Saving Simulink Design Verifier Options

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

The Simulink Design Verifier options stay with the model, even if you open the model on a MATLAB installation that does not have a Simulink Design Verifier license. If you then open the model on a system with a Simulink Design Verifier license, the software can analyze the model with the blocks and options that you originally added to the model.

Generating Test Cases

- “About Test Case Generation” on page 7-2
- “Workflow for Generating Test Cases” on page 7-4
- “Generating Test Cases for a Model” on page 7-5
- “Generating Test Cases for a Subsystem” on page 7-31

About Test Case Generation

The Simulink Design Verifier software can generate test cases that satisfy coverage objectives for your model, including:

- Decision coverage
- Condition coverage
- Modified condition/decision coverage (MC/DC)

Test cases help you confirm that a model behaves correctly by demonstrating how its blocks execute in different modes. When generating test cases, the software performs a formal analysis of your model. After completing the analysis, the software produces a report that details the results and a test harness model that contains test cases. Review the report and simulate the test harness model to confirm that the test cases achieve coverage objectives for your model.

Also, you can generate test cases based on the use of test case blocks or functions.

Test Case Blocks

The software provides two blocks that allow you to customize test cases for your Simulink models:

- The Test Objective block defines the values of a signal that a test case must satisfy.
- The Test Condition block constrains the values of a signal during an analysis.

Test Case Functions

The software provides two Embedded MATLAB functions to customize test cases for a Simulink Model or Stateflow chart:

- `sldv.test` — Specifies a test objective
- `sldv.condition` — Specifies a test condition

These functions:

- Identify mathematical relationships for testing in a form that can be more natural than using block parameters
- Support specifying multiple objectives, assumptions, or conditions without complicating the model
- Provide access to the power of MATLAB
- Support separation of verification and model design

See the `sldv.test` reference page for an example.

Workflow for Generating Test Cases

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

- 1** Ensure that your model is compatible for use with the Simulink Design Verifier software (for an example, see “Checking Compatibility of the Example Model” on page 7-7).
- 2** Optionally, instrument your model with blocks that specify test objectives and test conditions (for an example, see “Customizing Test Generation” on page 7-22). Or, you can use Embedded MATLAB functions to customize test cases; see “Test Case Functions” on page 7-2.
- 3** Specify options that control how Simulink Design Verifier generates test cases for your model. (For an example, see “Configuring Test Generation Options” on page 7-11.)
- 4** Execute the Simulink Design Verifier analysis and review the results (for examples, see “Analyzing the Example Model” on page 7-14 and “Reanalyzing the Example Model” on page 7-26).

Generating Test Cases for a Model

In this section...
“About This Example” on page 7-5
“Constructing the Example Model” on page 7-6
“Checking Compatibility of the Example Model” on page 7-7
“Configuring Test Generation Options” on page 7-11
“Analyzing the Example Model” on page 7-14
“Customizing Test Generation” on page 7-22
“Reanalyzing the Example Model” on page 7-26
“Analyzing Contradictory Models” on page 7-30

About This Example

The sections that follow describe a Simulink model for which you generate test cases that achieve decision coverage. This example demonstrates the test-generation capabilities of the Simulink Design Verifier software.

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

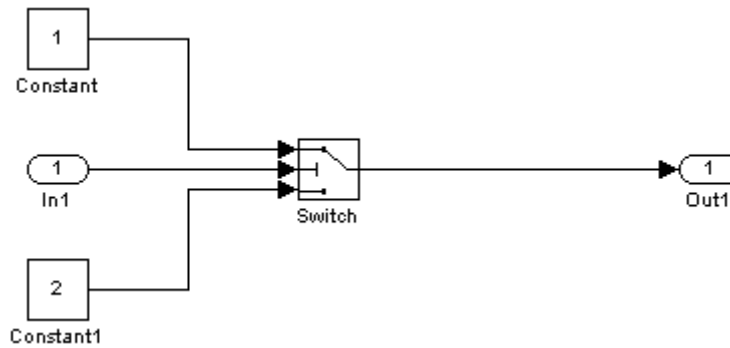
Task	Description	See...
1	Construct the example model.	“Constructing the Example Model” on page 7-6
2	Ensure your model’s compatibility with the Simulink Design Verifier software.	“Checking Compatibility of the Example Model” on page 7-7
3	Configure the Simulink Design Verifier software to generate tests.	“Configuring Test Generation Options” on page 7-11

Task	Description	See...
4	Generate test cases for your model and interpret the results.	“Analyzing the Example Model” on page 7-14
5	Add a Test Condition block to customize test generation.	“Customizing Test Generation” on page 7-22
6	Generate test cases for your modified model and interpret the results.	“Reanalyzing the Example Model” on page 7-26

Constructing the Example Model

In this task, you construct a Simulink model that you use throughout the remaining tasks:

- 1** Create a new Simulink model.
- 2** Copy the following blocks into your empty model window:
 - From the Sources library, an Inport block to initiate the input signal whose value the Simulink Design Verifier software controls
 - From the Signal Routing library, a Switch block to provide simple logic
 - From the Sources library, two Constant blocks to serve as Switch block data inputs
 - From the Sinks library, an Outport block to receive the output signal
- 3** In your model, double-click one of the Constant blocks and specify its **Constant value** parameter as 2.
- 4** Connect the blocks so that your model appears similar to the following diagram.



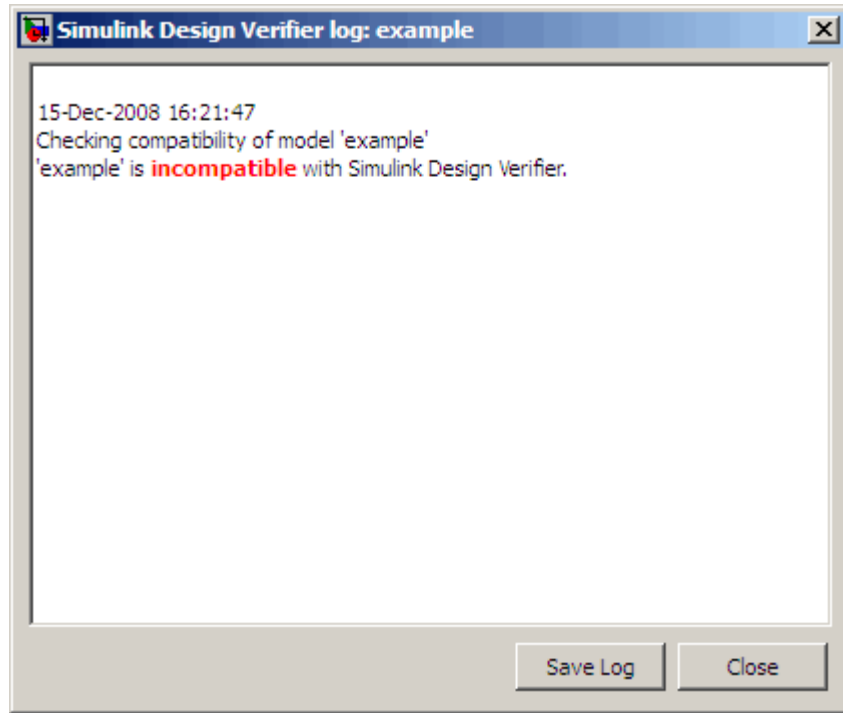
- 5 Save your model as `example.mdl` for use in the remaining tasks.

Checking Compatibility of the Example Model

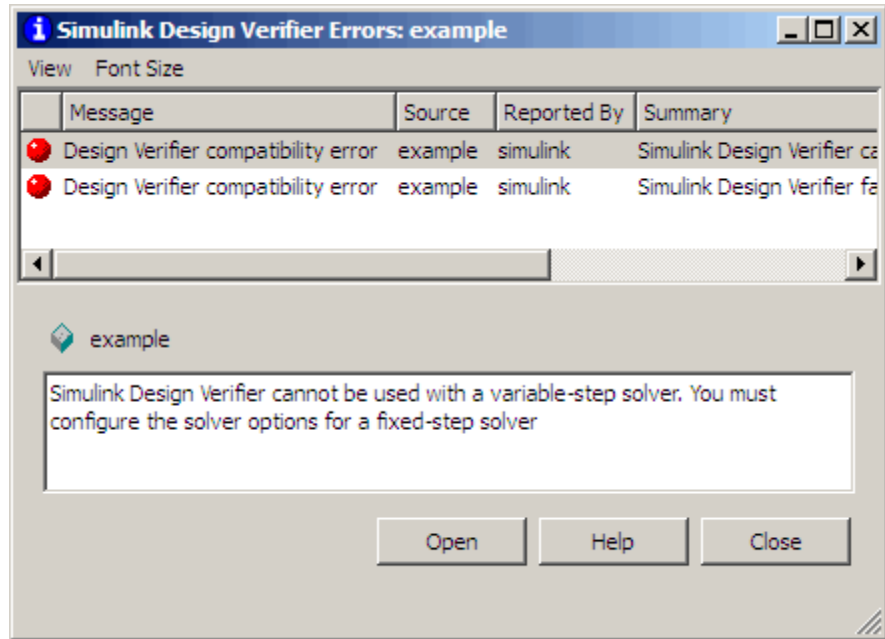
In this task, you ensure that your model is compatible for use with the Simulink Design Verifier software. Specifically, you check the compatibility of the `example` model:

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

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

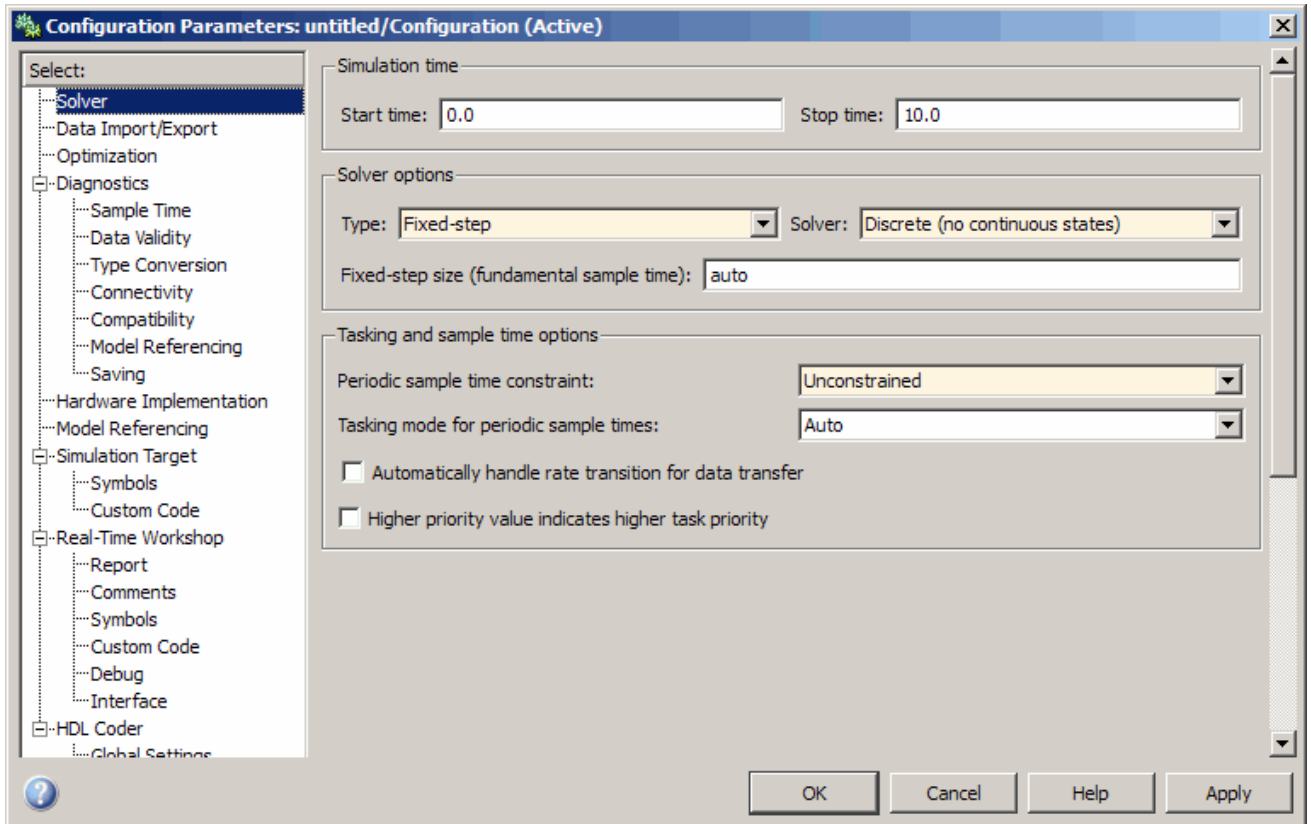


It also displays in the Simulation Diagnostics Viewer the following incompatibility error.



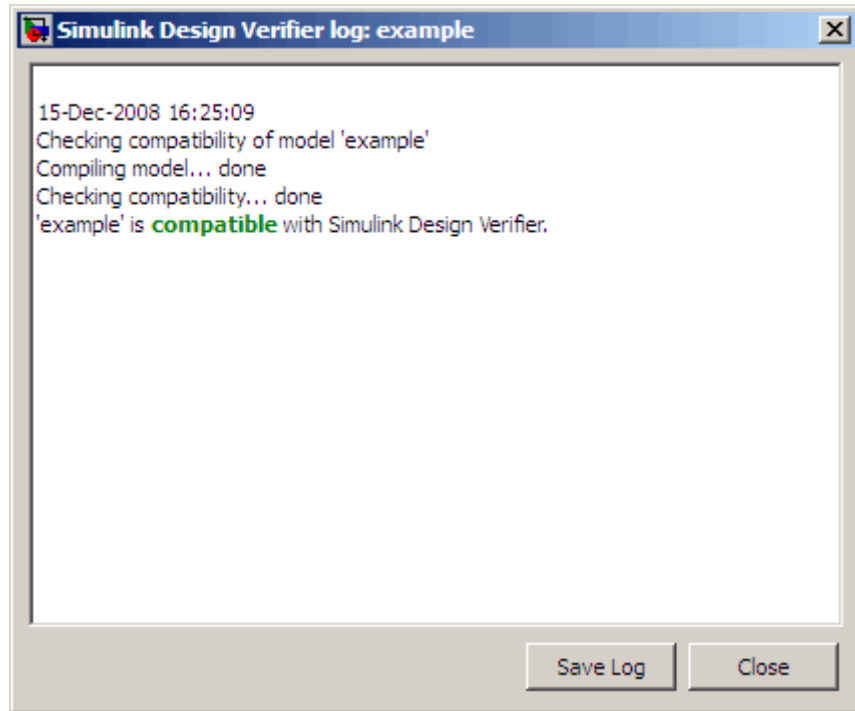
The error message informs you that the Simulink Design Verifier software does not support variable-step solvers. To work around this incompatibility, use a fixed-step solver.

- 2 In your Simulink model window, select **Simulation > Configuration Parameters** to open the Configuration Parameters dialog box.
- 3 On the left side of the Configuration Parameters dialog box, in the **Select** tree, click the **Solver** category (if not already selected). Under **Solver options** on the right side, set the **Type** option to **Fixed-step** and set the **Solver** option to **Discrete (no continuous states)**.



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

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



6 Save your model for use in the next task.

What If a Model Is Partially Compatible?

If the compatibility check indicates that your model is partially compatible, your model contains at least one element that is incompatible with the Simulink Design Verifier software. You can continue analyzing a partially compatible model if you turn on automatic stubbing. For details, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

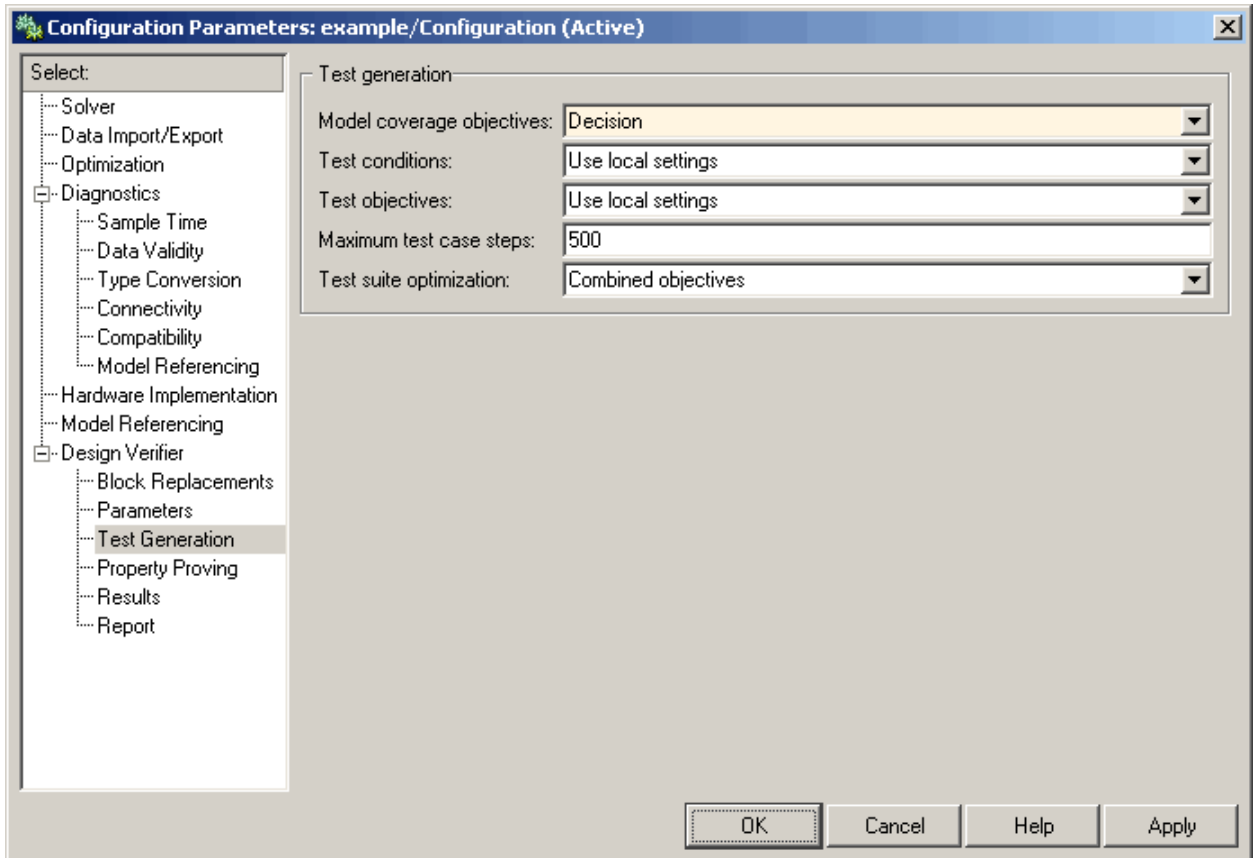
Configuring Test Generation Options

In this task, you configure the Simulink Design Verifier software to generate test cases that achieve complete decision coverage for your simple model:

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

In the Configuration Parameters dialog box, you see the Simulink Design Verifier options.

- 2** On the left side of the Configuration Parameters dialog box, in the **Select** tree, 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** On the left side of the Configuration Parameters dialog box, in the **Select** tree, click the **Test Generation** category.
- 4** On the **Test Generation** pane, set the value of the **Model coverage objectives** parameter to **Decision**.



Note The **Test suite optimization** parameter is set by default to Combined objectives. If you want to generate fewer but longer test cases, select Long test cases for the **Test suite optimization** parameter.

- 5 Click **Apply** and then **OK** to apply your change and close the Configuration Parameters dialog box.
- 6 Save your model for use in the next task.

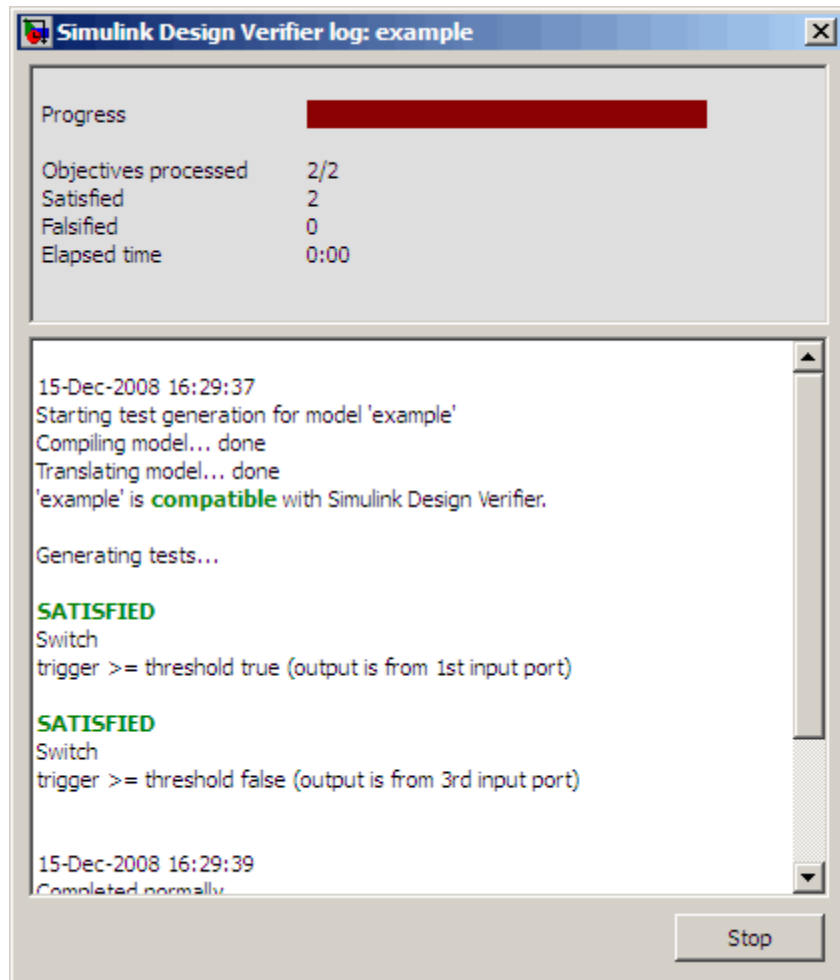
Note On the **Test Generation** pane, you can optionally specify values for other parameters that control how the Simulink Design Verifier software generates test cases for your model. For more information, see “Test Generation Pane” on page 6-10.

Analyzing the Example Model

In this task, you execute the Simulink Design Verifier analysis that you configured in the previous task. The software generates test cases for your example model and produces results for you to interpret:

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

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



In the Simulink Design Verifier log window, you can see how the proof progresses. You see information such as the number of test objectives processed and how many of those objectives were satisfied. Also in this dialog box, you can click **Stop** to terminate the proof at any time.

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

- An HTML report named `example_report.html`

- A test harness model named `example_harness.mdl`
- A Signal Builder window containing the test-case signals

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

- 2 Review the Simulink Design Verifier report, starting with the **Table of Contents**. Click an item to navigate in the report.

Table of Contents
1. Summary
2. Analysis Information
3. Test Objectives Status
4. Model Items
5. Test Cases

- 3 In the **Table of Contents**, click Summary to display the report's Summary chapter.

Chapter 1. Summary	
Analysis Information	
Model:	example
Mode:	TestGeneration
Status:	Completed normally
Objectives Status	
Number of Objectives:	2
Objectives Satisfied:	2

The Summary chapter lists information about the model and the status of the objectives—satisfied or not.

- 4 In the **Table of Contents**, click Analysis Information to display the Analysis Information chapter.

Chapter 2. Analysis Information

Table of Contents

[Model Information](#)

[Analysis Options](#)

[Approximations](#)

Model Information

File: C:\SLVNV\example.mdl
 Version: 1.2
 Time Stamp: Mon Dec 15 16:28:43 2008
 Author: slemaire

Analysis Options

Mode: TestGeneration
 Test Suite Optimization: CombinedObjectives
 Maximum Testcase Steps: 500 time steps
 Test Conditions: UseLocalSettings
 Test Objectives: UseLocalSettings
 Model Coverage Objectives: Decision
 Maximum Processing Time: 600s
 Block Replacement: off
 Parameters Analysis: off
 Save Data: on
 Save Harness: on
 Save Report: on

Approximations

Simulink Design Verifier performed the following approximations during analysis. These can impact the precision of the results generated by Simulink Design Verifier. Please see the product documentation for further details.

 	Type	Description
1	Rational approximation	The model includes floating-point arithmetic. Simulink Design Verifier approximates floating-point arithmetic with rational number arithmetic.

The Analysis Information chapter provides information about:

- The model you analyzed
- The options you specified for the analysis
- Approximations the software performed during the analysis

- In the **Table of Contents**, click **Test Objectives Status** to display the report's Test Objectives Status chapter.

Chapter 3. Test Objectives Status

Table of Contents

[Objectives Satisfied](#)

Objectives Satisfied

Simulink Design Verifier found test cases that exercise these test objectives.

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

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

- Under the table **Test Case** column, click **2** to display the Test Case 2 section.

Test Case 2

Summary

Length: 0 Seconds (1 sample periods)

Objective Count: 1

Objectives

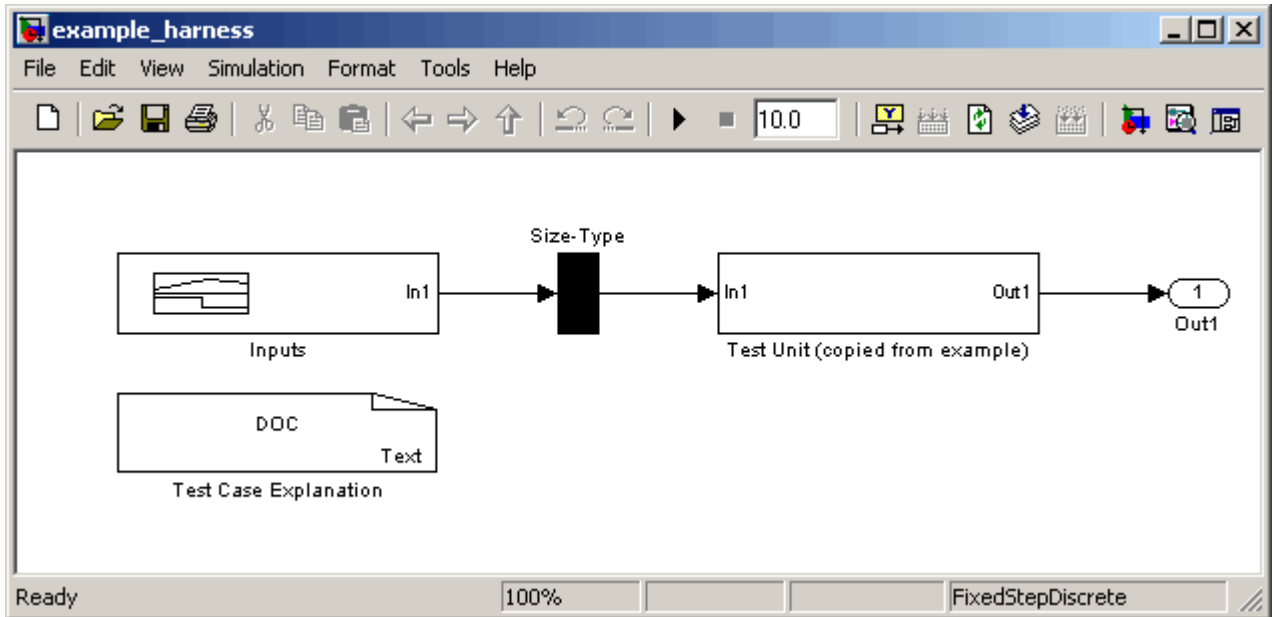
Step	Time	Model Item	Objectives
1	0	Switch	trigger >= threshold false (output is from 3rd input port)

Generated Input Data

Time	0
Step	1
In1	-1

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

- 7 Review the harness model named `example_harness.mdl`.

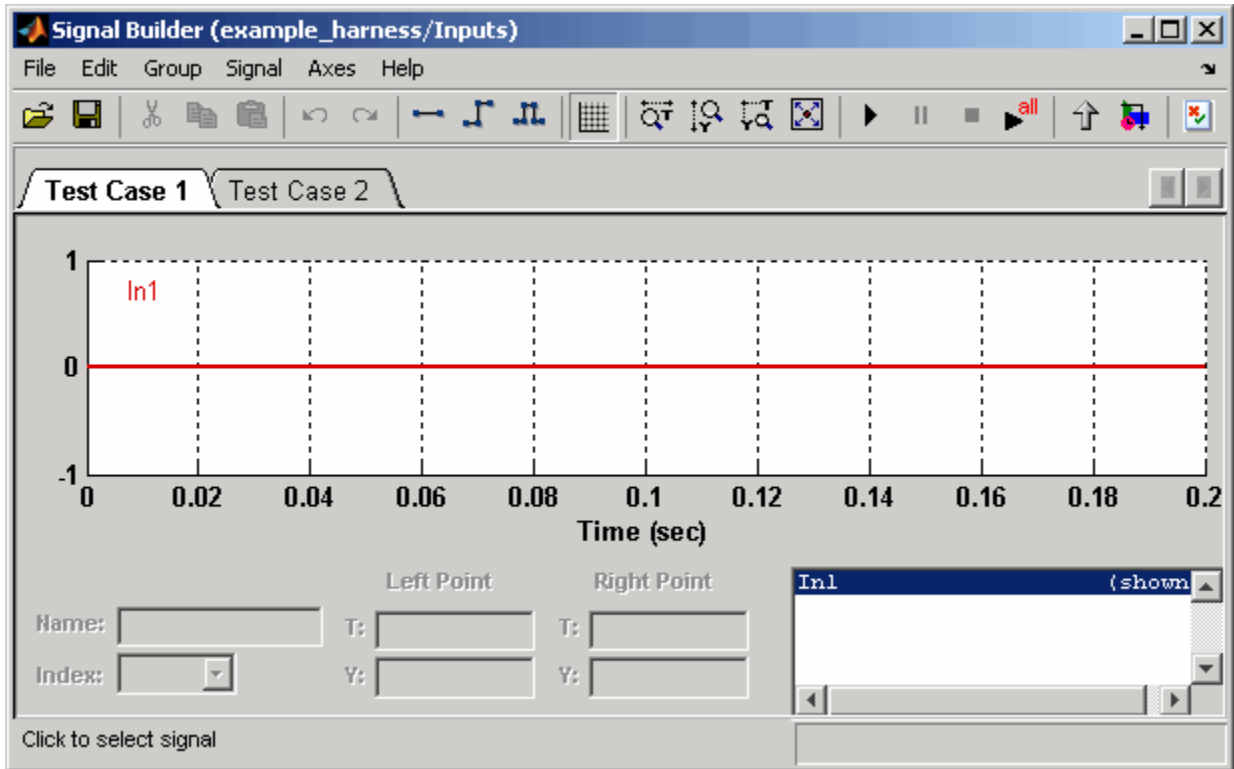


The harness model contains the following items:

- Signal Builder block named **Inputs** — Groups of signals that achieve test objectives in your model
- Subsystem block named **Test Unit** — A copy of your model
- DocBlock named **Test Case Explanation** — A text description of the test cases that the Simulink Design Verifier software generates



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

- 8 To simulate the test harness and confirm that the test cases achieve complete decision coverage, double-click the Inputs block to open the Signal Builder dialog box.



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

The Simulink Design Verifier software simulates the test harness using all the test cases, collects model coverage information, and displays a coverage report that includes the following Summary.

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

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

What If Analysis Generates Many Test Cases?

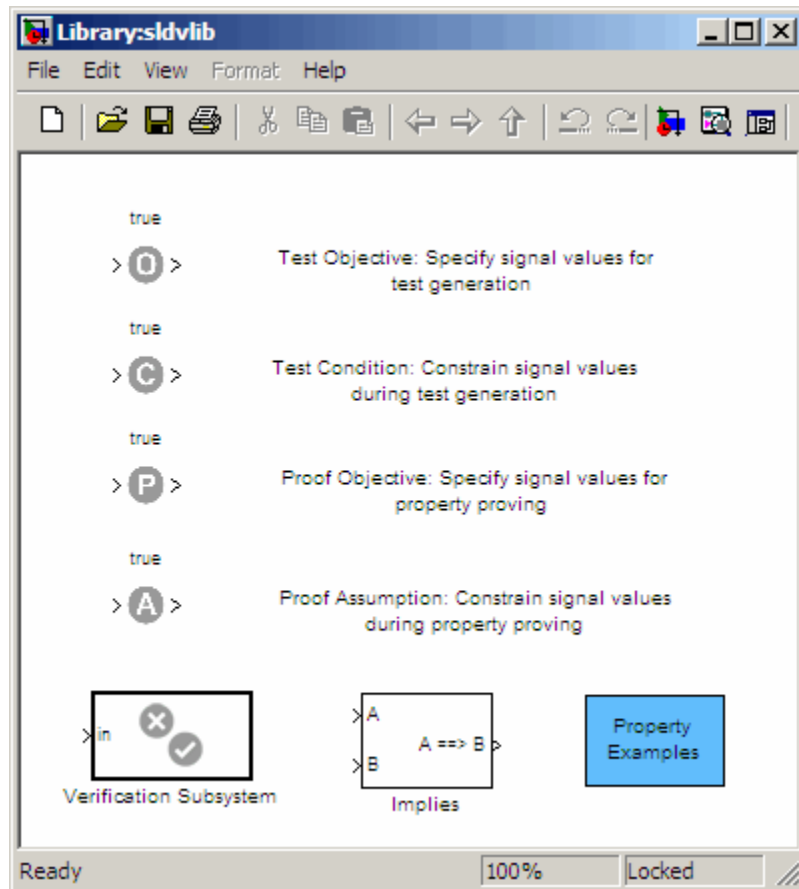
If you prefer to review results that are combined into a smaller number of longer test cases, set the **Test suite optimization** parameter to **Long test cases** and rerun the analysis. In the **Long test cases** optimization, the analysis generates fewer but longer test cases that each satisfy multiple test objectives. This optimization creates a more efficient analysis and easier-to-review results.

To compare the **Long test cases** results to the **Combined objectives** results (the default), see “Combining Test Cases” on page 1-23.

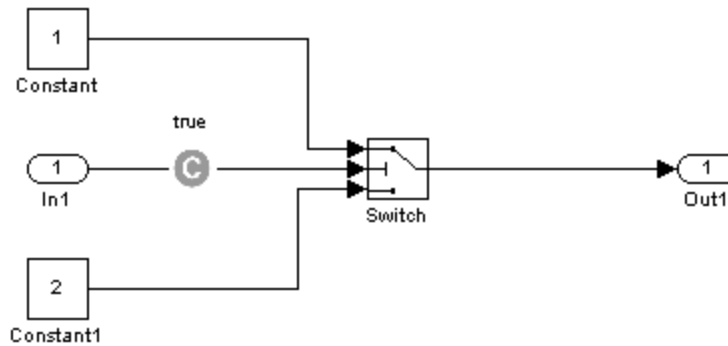
Customizing Test Generation

In this task, you modify the example model for which you attained complete decision coverage. Specifically, you customize test generation by adding and configuring a Test Condition block:

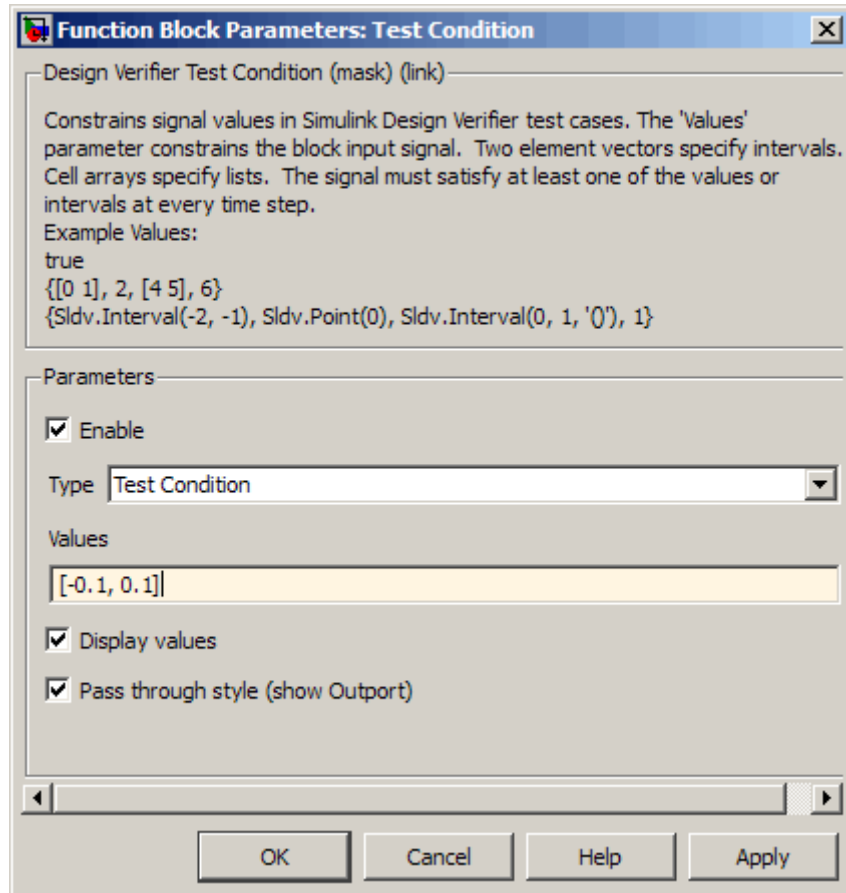
- 1 In the MATLAB Command Window, enter `sldvlib` to display the Simulink Design Verifier library.



- 2** Copy the Test Condition block to your model by dragging it from the Simulink Design Verifier library to your model window.
- 3** In the model window, insert the Test Condition block between the Switch and Outport blocks.



- 4** In your model, double-click the Test Condition block to access its attributes.
The Test Condition block parameter dialog box opens.
- 5** In the **Values** box, enter $[-0.1, 0.1]$. When generating test cases for this model, the Simulink Design Verifier software constrains the signal values, entering the Switch block control port to the specified interval.



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

7 Save your model for use in the next task.

Simulink Design Verifier blocks are preserved with a model, even if you open the model on a MATLAB installation that does not have a Simulink Design Verifier license. If you then open the model on a system with a Simulink Design Verifier license, the software can analyze the model with the blocks and options that you originally added to the model.

Reanalyzing the Example Model

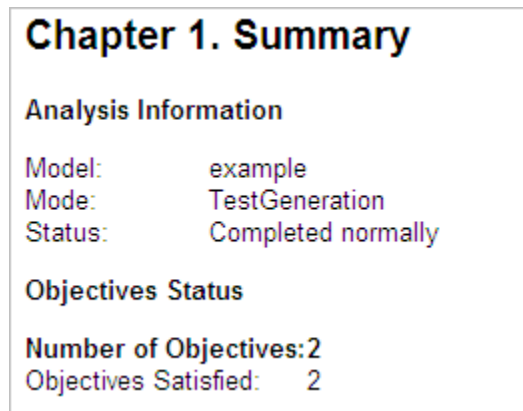
In this task, you analyze the example model with the Test Condition block. To observe how the Test Condition block affects test generation, compare the result of this analysis to the result that you obtained in “Analyzing the Example Model” on page 7-14.

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

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

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

- 2 To begin reviewing the report, in the **Table of Contents**, click **Summary**.



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

- 3 In the **Table of Contents**, click **Analysis Information**. Scroll to the bottom of this chapter, to the Constraints section.

Constraints

Name	Constraint
Test Condition	[-0.1, 0.1]

This section lists the Test Condition block that you added to constrain the value of the Switch block control signal to the interval [-0.1, 0.1].

- 4 In the **Table of Contents**, click **Test Objectives Status**.

Chapter 3. Test Objectives Status

Table of Contents

[Objectives Satisfied](#)

Objectives Satisfied

Simulink Design Verifier found test cases that exercise these test objectives.

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

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

- 5 Under the table **Test Cases** column, click 2.

Test Case 2

Summary

Length: 0 Seconds (1 sample periods)
Objective Count: 1

Objectives

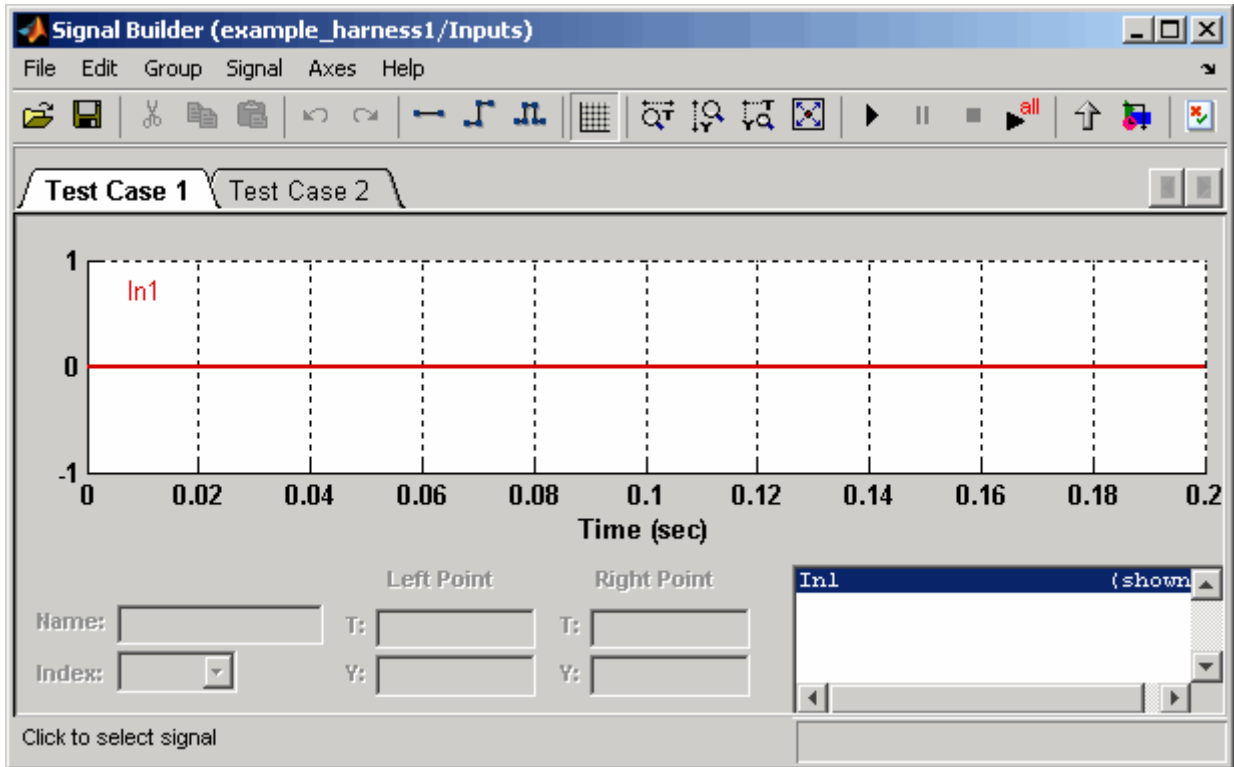
Step	Time	Model Item	Objectives
1	0	Switch	trigger >= threshold false (output is from 3rd input port)

Generated Input Data

Time	0
Step	1
In1	-0.05

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

- 6** To confirm that the test case achieves complete decision coverage, go to the harness model named `example_harness1.mdl`.
- 7** Double-click the Inputs block to open the Signal Builder dialog box.



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

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

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

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

Analyzing Contradictory Models

If the analysis produces the error `The model is contradictory in its current configuration`, the software detected a contradiction in your model and cannot analyze the model. You have a contradiction if your model has Test Objective blocks with incorrect parameters. For example, an objective that states that a signal must be between 0 and 5 when the signal is constant 10.

If the software detects a contradiction, all previous results are invalidated and the software reports that the some of the objectives cannot be satisfied.

Generating Test Cases for a Subsystem

If you have a large model, you can generate test cases for subsystems in the model and review the analysis in smaller, manageable reports. The workflow for generating test cases for a subsystem is:

- 1** Open the model that contains the subsystem.
- 2** Make the subsystem atomic.
- 3** Run the Simulink Design Verifier software using the **Generate Tests for Subsystem** option.
- 4** Review the results.

The tutorial in “Analyzing a Subsystem” on page 1-26 describes how to analyze the Controller subsystem in the Cruise Control Test Generation model.

Proving Properties of a Model

- “About Property Proving” on page 8-2
- “Workflow for Proving Model Properties” on page 8-4
- “Proving Properties in a Model” on page 8-5
- “Proving Properties in a Subsystem” on page 8-28
- “Proving Complex Properties” on page 8-29

About Property Proving

The Simulink Design Verifier software can prove properties of your model. 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 attain a particular value or range of values during simulation. You can then use the Simulink Design Verifier software to prove whether such properties are valid for your model. The software performs a formal analysis of your model to prove or disprove the specified properties. If the software disproves a property, it provides a counterexample that demonstrates a property violation.

Proof Blocks

The Simulink Design Verifier software provides two blocks that allow you to specify property proofs in your Simulink models:

- Proof Objective — Define the values of a signal to prove
- Proof Assumption — Constrain the values of a signal during a proof

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

Proof Functions

The software provides two Embedded MATLAB functions to specify property proving for a Simulink Model or Stateflow chart:

- `sldv.prove` — Specifies a proof objective
- `sldv.assume` — Specifies a proof assumption

These functions:

- Identify mathematical relationships for proving properties in a form that can be more natural than using block parameters

- Support specifying multiple objectives, assumptions, or conditions without complicating the model
- Provide access to the power of MATLAB
- Support separation of verification and model design

For an example using these proof functions, see the `sldv.prove` reference page.

Workflow for Proving Model Properties

A recommended workflow for proving properties of your model is:

- 1** Ensure that your model is compatible for use with the Simulink Design Verifier software (for an example, see “Checking Compatibility of the Example Model” on page 8-7).
- 2** Instrument your model with blocks that specify proof objectives and proof assumptions (for examples, see “Instrumenting the Example Model” on page 8-11 and “Customizing the Example Proof” on page 8-22). Or, you can use Embedded MATLAB functions for proof objectives and assumptions; see “Proof Functions” on page 8-2.
- 3** Specify options that control how Simulink Design Verifier proves the properties of your model (for an example, see “Configuring Property-Proving Options” on page 8-14).
- 4** Execute the Simulink Design Verifier analysis and review the results (for examples, see “Analyzing the Example Model” on page 8-16 and “Reanalyzing the Example Model” on page 8-25).

For an exercise that demonstrates this workflow, see “Proving Properties in a Model” on page 8-5 .

Proving Properties in a Model

In this section...
“About This Example” on page 8-5
“Constructing the Example Model” on page 8-6
“Checking Compatibility of the Example Model” on page 8-7
“Instrumenting the Example Model” on page 8-11
“Configuring Property-Proving Options” on page 8-14
“Analyzing the Example Model” on page 8-16
“Customizing the Example Proof” on page 8-22
“Reanalyzing the Example Model” on page 8-25
“Analyzing Contradictory Models” on page 8-26

About This Example

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

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

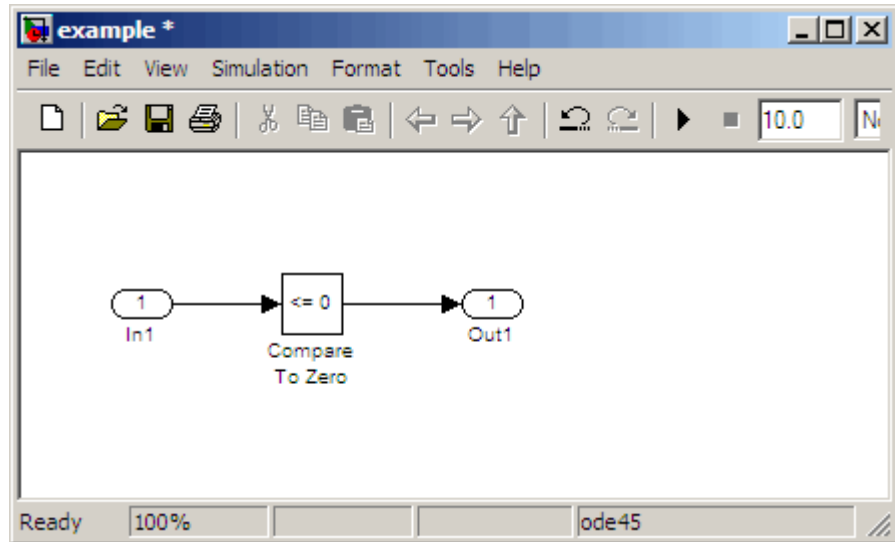
Task	Description	See...
1	Construct the example model.	“Constructing the Example Model” on page 8-6
2	Ensure your model’s compatibility with the Simulink Design Verifier software.	“Checking Compatibility of the Example Model” on page 8-7
3	Add a Proof Objective block to your model to prepare for its proof.	“Instrumenting the Example Model” on page 8-11

Task	Description	See...
4	Configure the Simulink Design Verifier software to prove properties.	“Configuring Property-Proving Options” on page 8-14
5	Prove a property of your model and interpret the results.	“Analyzing the Example Model” on page 8-16
6	Add a Proof Assumption block to customize the proof.	“Customizing the Example Proof” on page 8-22
7	Prove a property of your modified model and interpret the results.	“Reanalyzing the Example Model” on page 8-25

Constructing the Example Model

In this task, you construct a Simulink model that you use throughout the remaining tasks. To complete this task, perform the following steps:

- 1** Create an empty Simulink model.
- 2** Copy the following blocks into your empty model window:
 - From the Sources library, an Inport block to initiate the input signal whose value the Simulink Design Verifier software controls
 - From the Logic and Bit Operations library, a Compare To Zero block to provide simple logic
 - From the Sinks library, an Outport block to receive the output signal
- 3** Connect these blocks such so your model appears similar to the following model:



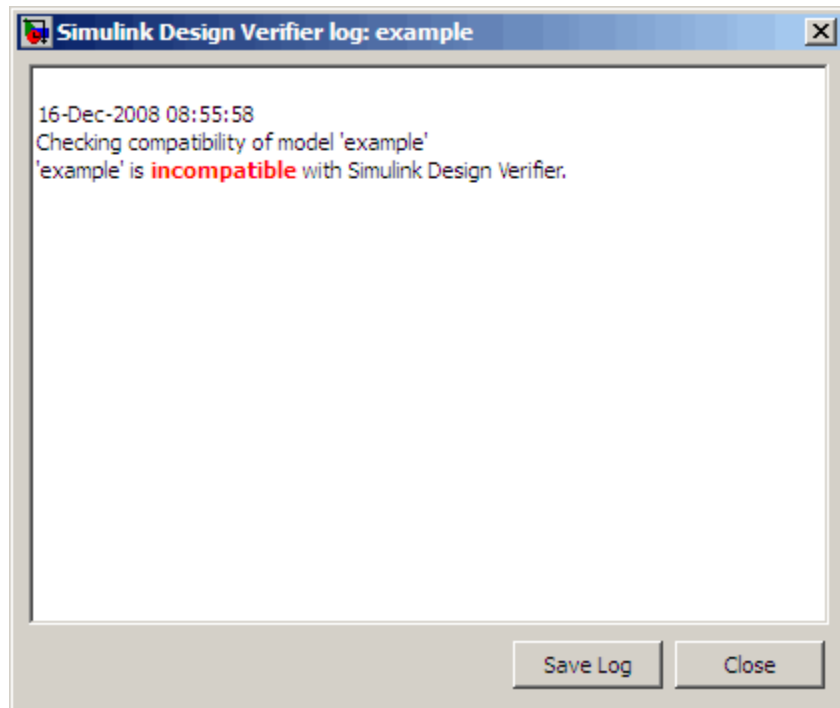
4 Save your model as `example.mdl` for use in the next task.

Checking Compatibility of the Example Model

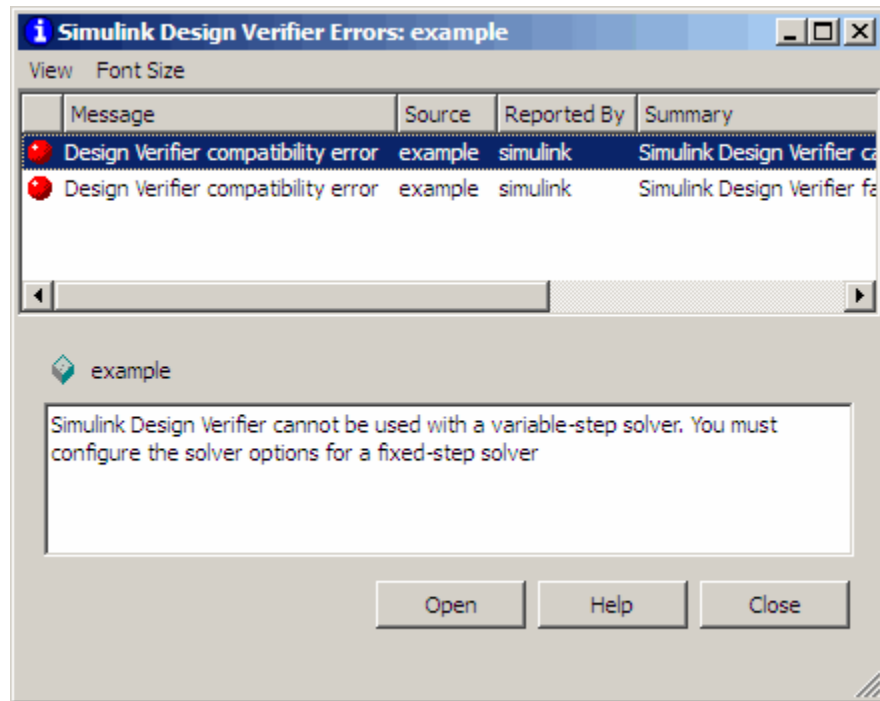
In this task, you ensure that a model is compatible for use with the Simulink Design Verifier software. Specifically, you check the compatibility of the simple Simulink model that you created in the previous task. To complete this task, perform the following steps:

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

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



In the Simulation Diagnostics Viewer, Simulink Design Verifier also displays the following incompatibility error.

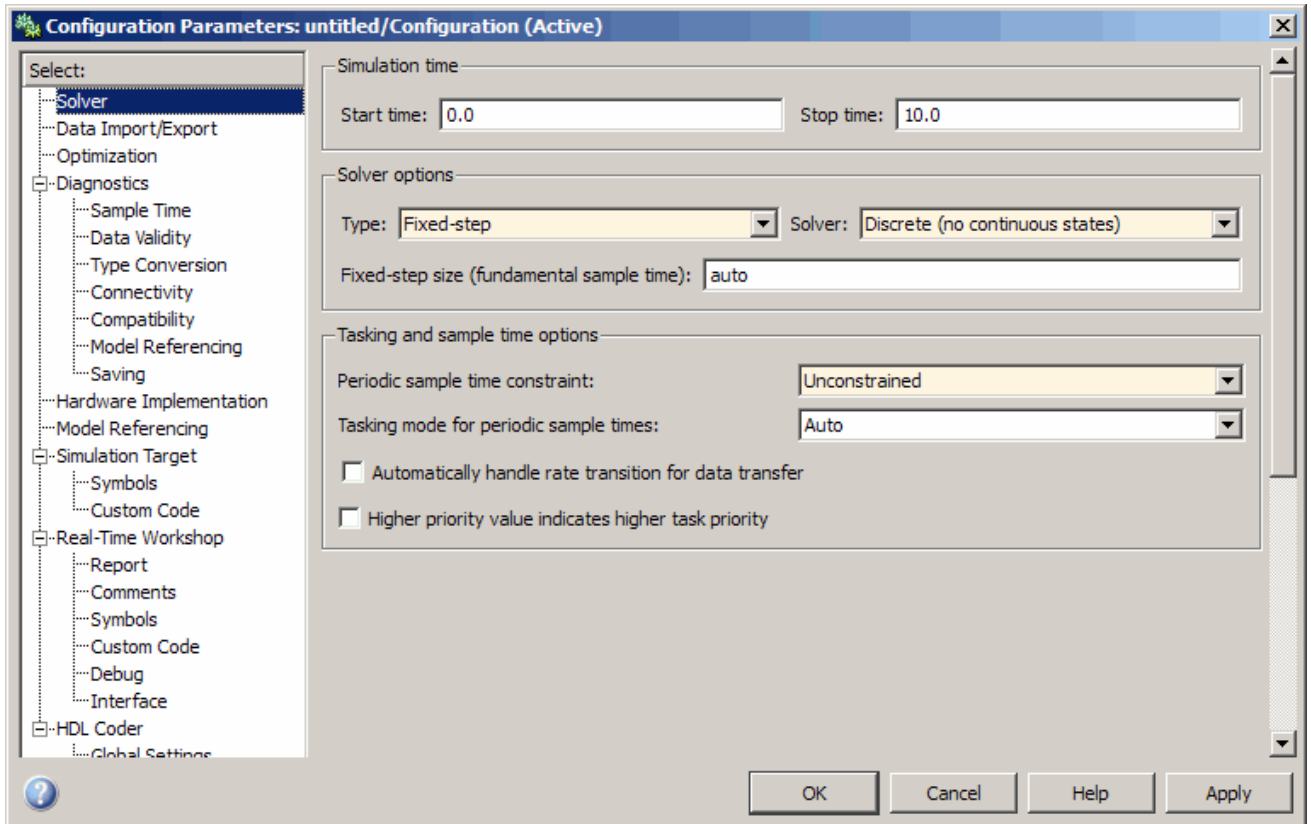


The error message informs you that the Simulink Design Verifier software does not support variable-step solvers. To work around this incompatibility, you must use a fixed-step solver.

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

The Configuration Parameters dialog box appears.

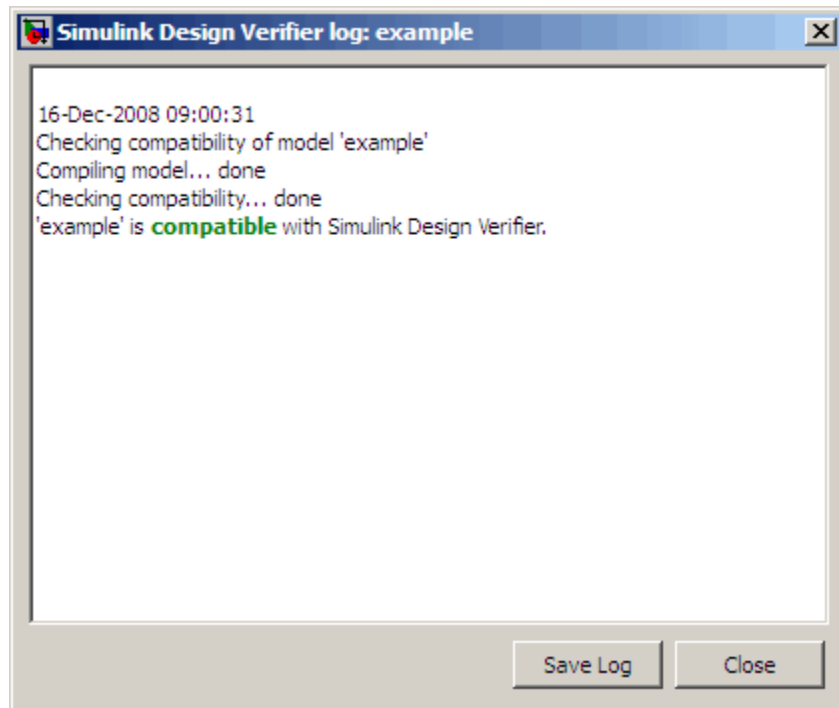
- 3 On the left side of the Configuration Parameters dialog box, in the **Select** tree, click the **Solver** category (if not already selected). Under **Solver options** on the right side, set the **Type** option to **Fixed-step**, and then set the **Solver** option to **Discrete (no continuous states)**.



4 Click **Apply** and then **OK** to apply your changes and close the Configuration Parameters dialog box.

5 Verify the compatibility of your model. In your Simulink model window, select **Tools > Design Verifier > Check Model Compatibility**.

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



6 Save your `example.mdl` model for use in the next task.

What If a Model Is Partially Compatible?

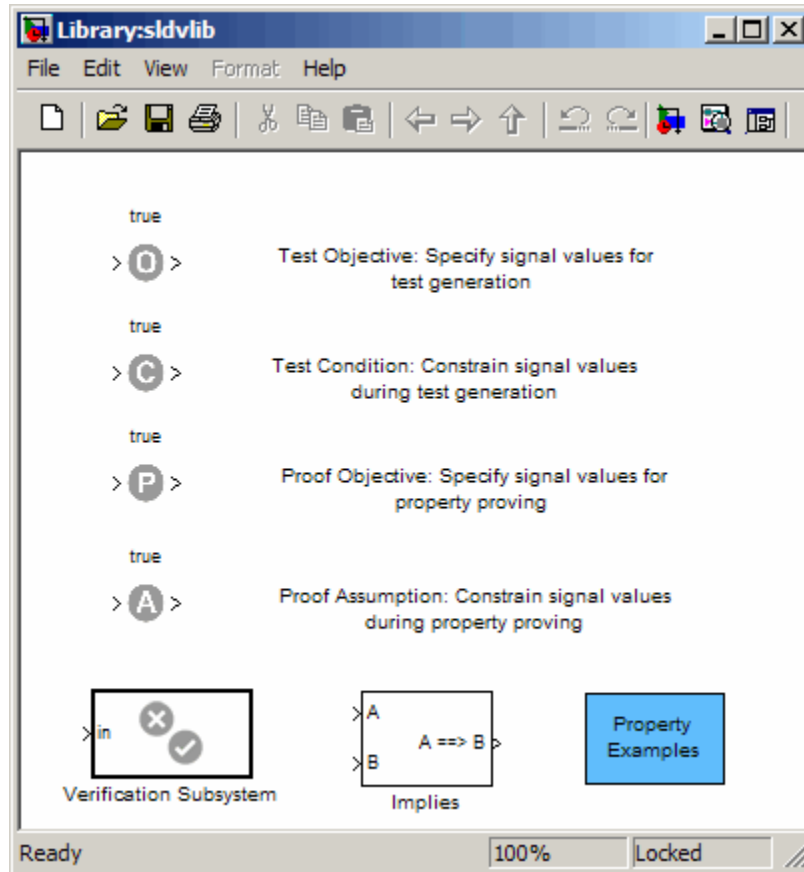
If the compatibility check indicates that your model is partially compatible, your model contains at least one element that is incompatible with the Simulink Design Verifier software. You can continue analyzing a partially compatible model if you turn on automatic stubbing. For details, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

Instrumenting the Example Model

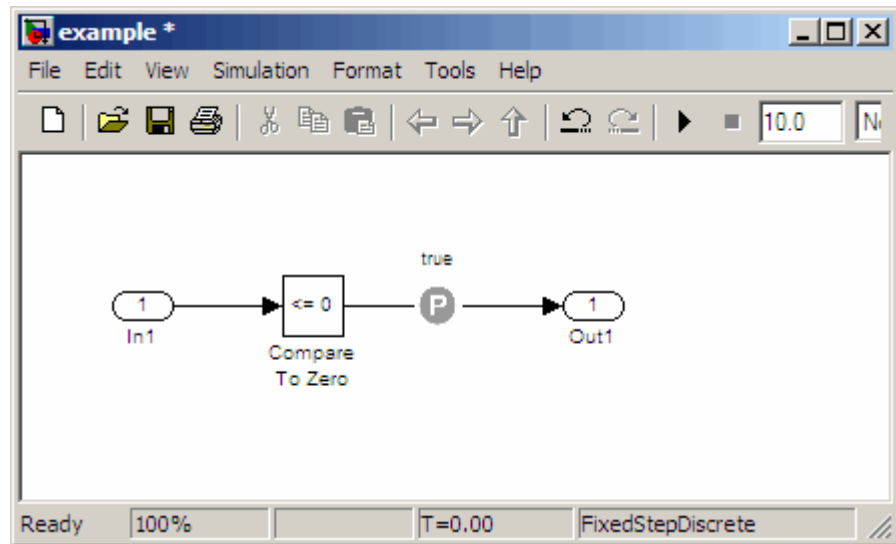
In this task, you prepare your example model so that you can prove its properties with the Simulink Design Verifier software. Specifically, you instrument the model by adding and configuring a Proof Objective block. To complete this task, perform the following steps:

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

The Simulink Design Verifier library appears.



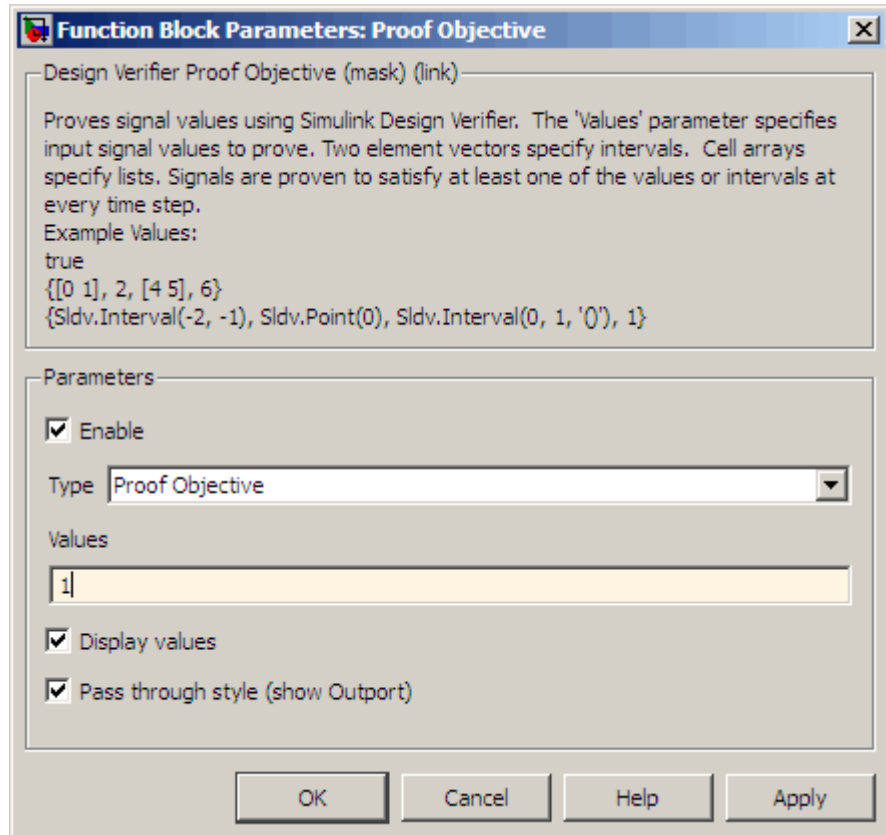
- 2 Copy the Proof Objective block to your model by dragging it from the Simulink Design Verifier library to your model window.
- 3 In your model window, insert the Proof Objective block between the Compare To Zero and Outport blocks. For more information about how to perform this step, (see "Inserting Blocks in a Line" in the Simulink documentation).



- 4 In your model, double-click the Proof Objective block to access its attributes.

The Proof Objective block parameter dialog box opens.

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



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

7 Save your `example.mdl` model for use in the next task.

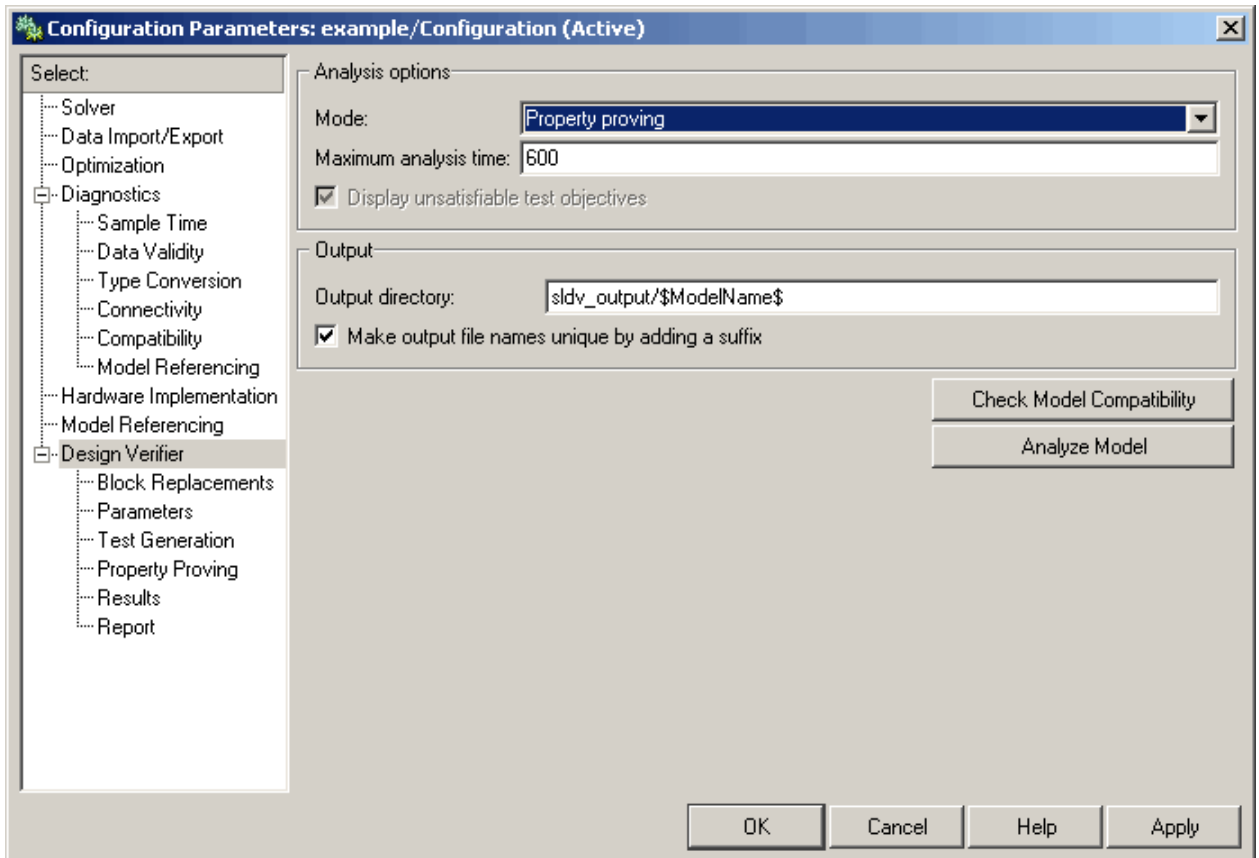
Configuring Property-Proving Options

In this task, you configure the Simulink Design Verifier software to prove properties of the simple Simulink model that you instrumented. To complete this task, perform the following steps:

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

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

- 2 On the left side of the Configuration Parameters dialog box, in the **Select** tree, select the **Design Verifier** category. Under **Analysis options** on the right side, set the **Mode** option to Property proving.



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

Note On the **Property Proving** pane, you can optionally specify values for other parameters that control how the Simulink Design Verifier software proves properties of your model. For more information, see “Property Proving Pane” on page 6-12.

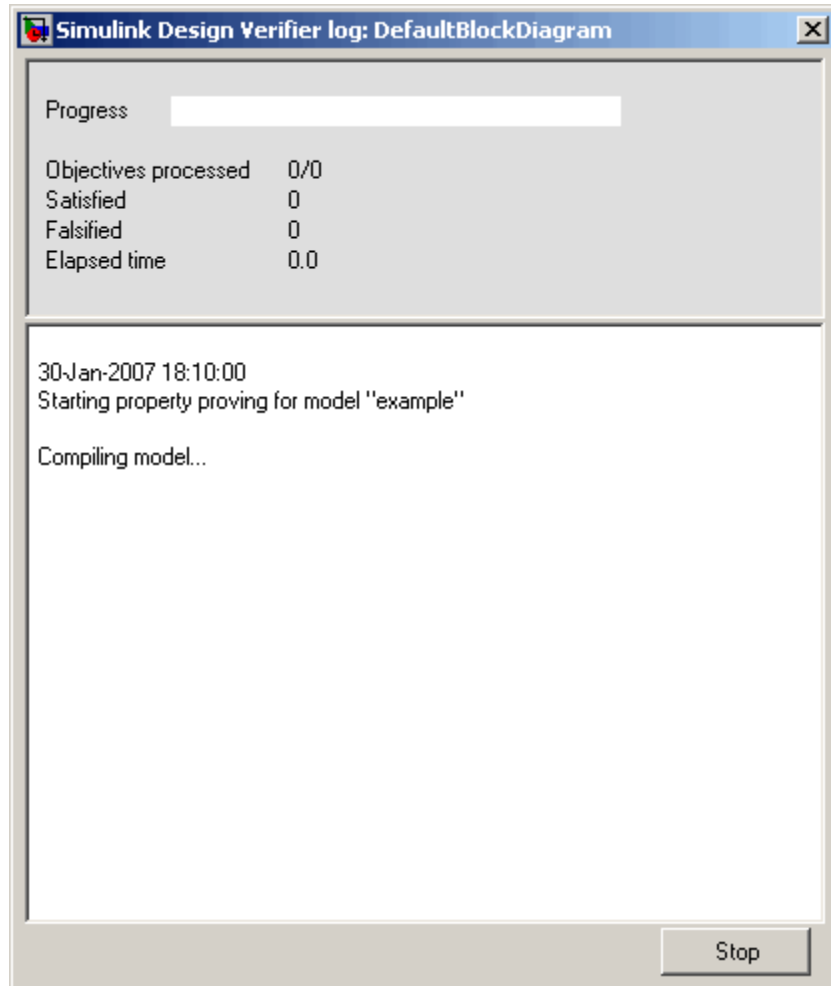
4 Save your `example.mdl` model for use in the next task.

Analyzing the Example Model

In this task, you execute the Simulink Design Verifier analysis. The software proves a property of your example model and produces results for you to interpret. To complete this task, perform the following steps:

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

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



In the Simulink Design Verifier log window, you can see how the proof progresses. You see information such as the number of objectives processed and how many of those objectives were satisfied. Also, in this dialog box you can click **Stop** to terminate the proof at any time.

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

- Simulink Design Verifier report — an HTML report named `example_report.html`.
- Test harness — a harness model named `example_harness.mdl`.
- Signal Builder dialog box — a dialog box that displays signals that falsify the proof objective in this model.

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

- 2 Review the Simulink Design Verifier report. The report includes the following **Table of Contents**. Click an item navigate to particular chapter or section.

Table of Contents	
1. Summary	
2. Analysis Information	
3. Proof Objectives Status	
4. Properties	

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

Chapter 1. Summary	
Analysis Information	
Model:	example
Mode:	PropertyProving
Status:	Completed normally
Objectives Status	
Number of Objectives:	1
Objectives Falsified with Counterexamples:	1

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

- 4 Scroll back to the top of the browser window. In the **Table of Contents**, click **Proof Objectives Status**.

Objectives Falsified with Counterexamples				
#:	Type	Model Item	Description	Counterexample
1	Custom Proof Objective	Proof Objective	Objective: 1	1

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

- 5 In the Objectives Falsified with Counterexamples table, under the **Counterexample** column, click 1.

Proof Objective

Summary

Model Item: [Proof Objective](#)

Property: Objective: 1

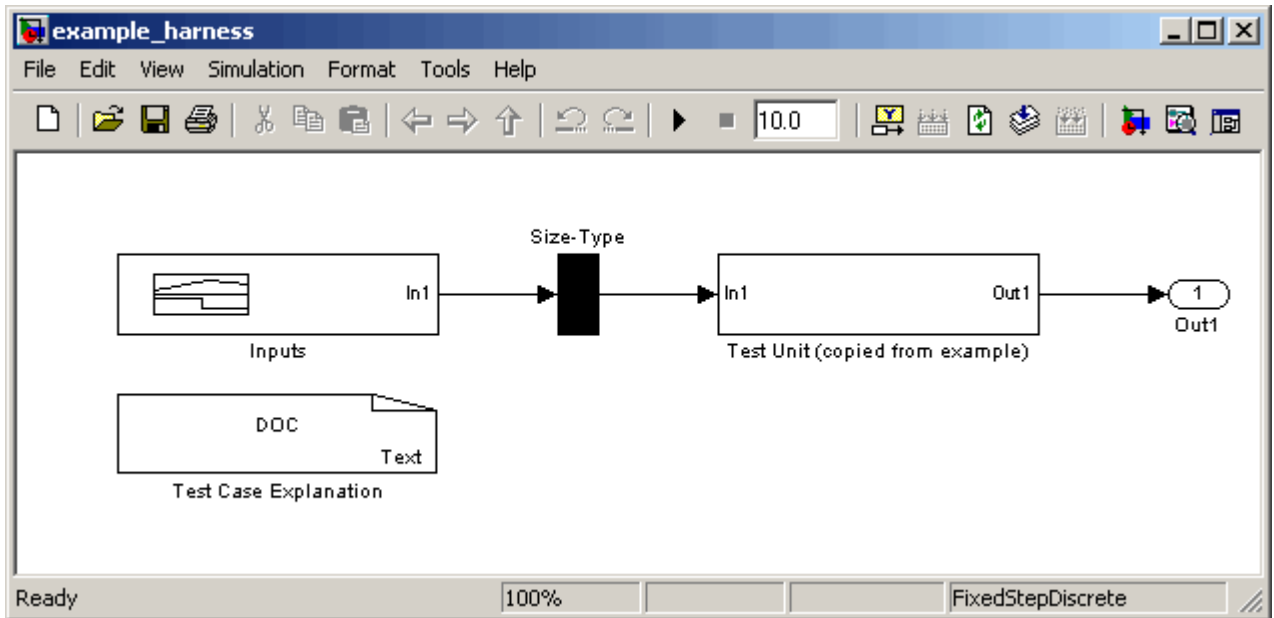
Status: Falsified

Counter Example

Time	0
Step	1
In1	99

This section displays information about proof objective 1 and provides details about the counterexample that the Simulink Design Verifier software generated to disprove that objective. In this counterexample, a signal value of 99 falsifies the objective that you specified using the Proof Objective block. That is, 99 is not less than or equal to 0, which causes the Compare To Zero block to return 0 (false) instead of 1 (true).

6 Review the harness model named `example_harness.mdl`.



The harness model contains the following items:

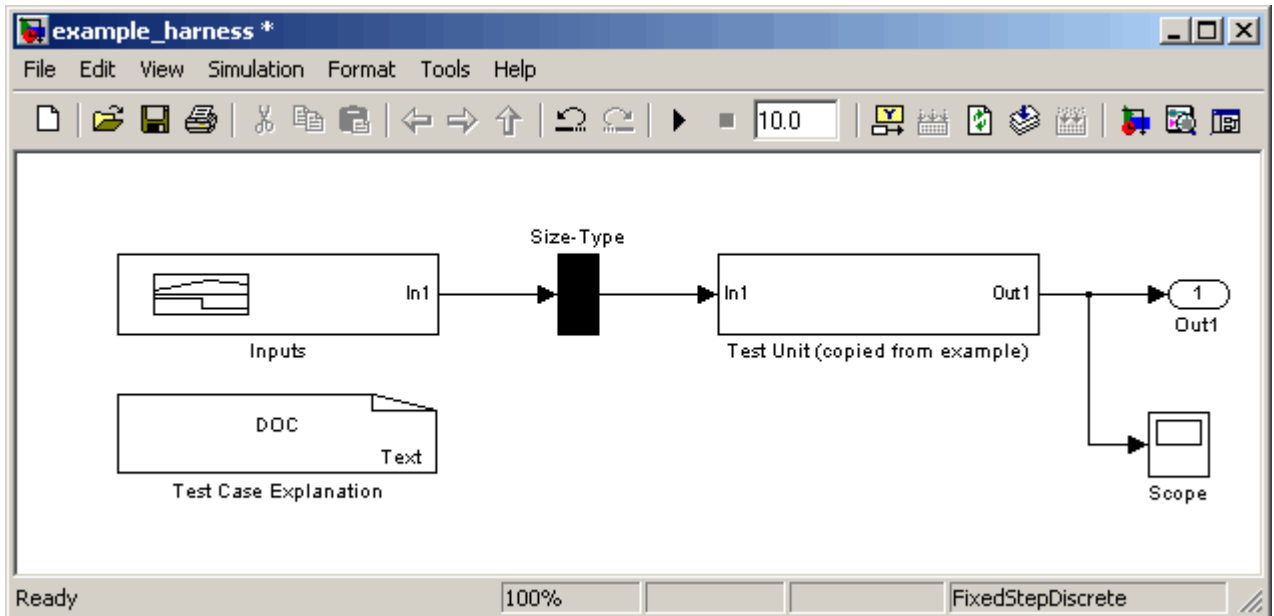
- Signal Builder block named `Inputs` — A group of signals that falsify proof objectives.
- Subsystem block named `Test Unit` — A copy of your model.
- DocBlock named `Test Case Explanation` — A textual description of the counterexamples that the software generates.

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

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

- 7 In the MATLAB Command Window, enter `simulink` to open the Simulink library.
- 8 From the Sinks library, copy a Scope block into your harness model window. The Scope block allows you to see the value of the signal output by the Compare To Zero block in your model.
- 9 In your harness model window, connect the output signal of the Test Unit subsystem to the Scope block.

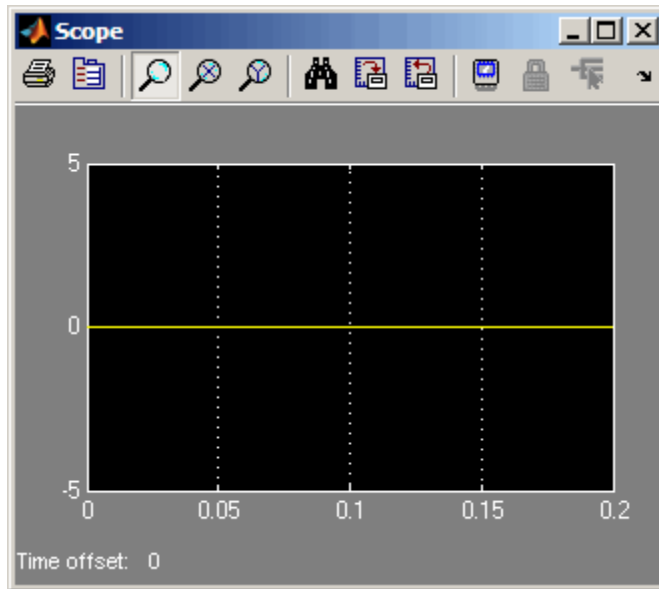
Your model should appear similar to the following model.



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

The Simulink software simulates the harness model.

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



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. Previously, you specified that the proof objective in your model is 1 (true). Therefore, the counterexample that the Signal Builder block supplies falsifies the proof objective.

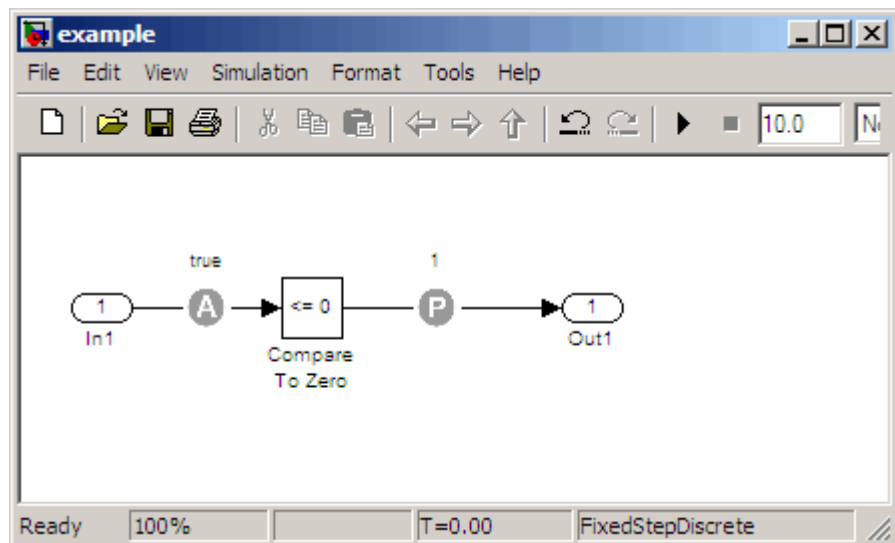
Customizing the Example Proof

In this task, you modify the simple Simulink model whose proof objective the Simulink Design Verifier software disproved in the previous task. 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, in the MATLAB Command Window, type `sldvlib`.

You see the Simulink Design Verifier library.

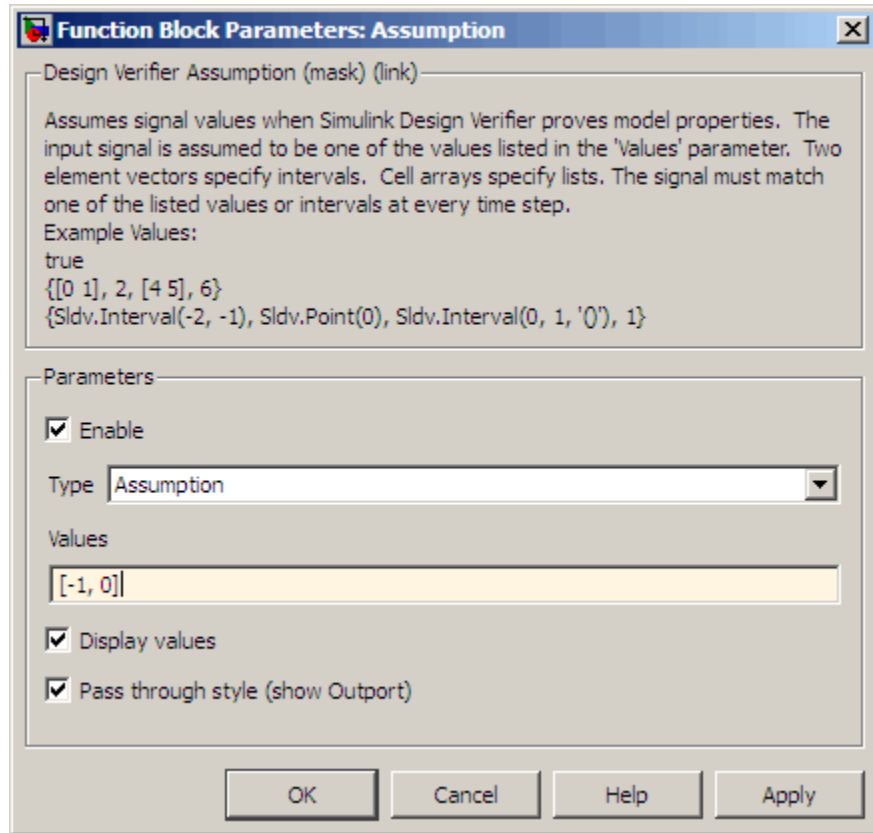
- 2 Copy the Proof Assumption block to your model (`example.mdl`) by dragging it from the Simulink Design Verifier library to your model window.
- 3 In your model window, insert the Proof Assumption block between the Inport and Compare To Zero blocks.



- 4 In your model, double-click the Proof Assumption block to access its attributes.

The Proof Assumption block parameter dialog box opens.

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



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

7 Save your `example.mdl` model for use in the next task.

Simulink Design Verifier blocks are preserved with a model, even if you open the model on a MATLAB installation that does not have a Simulink Design Verifier license. If you then open the model on a system with a Simulink Design Verifier license, the software can analyze the model with the blocks and options that you originally added to the model.

Reanalyzing the Example Model

In this task, you execute the Simulink Design Verifier analysis on the model that you modified. To observe how Proof Assumption blocks affect proofs, compare the result of this analysis to the result that you obtained in a previous task. To complete this task, perform the following steps:

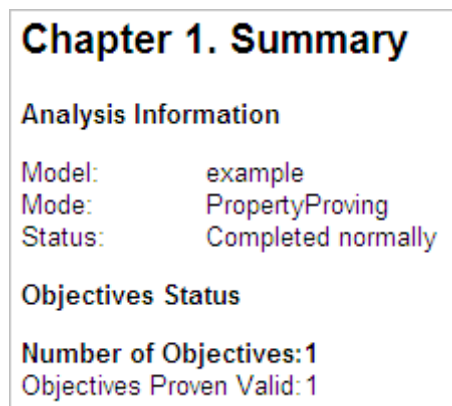
- 1 In your Simulink model window, select **Tools > Design Verifier > Prove Properties**.

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

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

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

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



The screenshot shows a report window titled "Chapter 1. Summary". Under the heading "Analysis Information", the following details are listed: Model: example, Mode: PropertyProving, and Status: Completed normally. Below this, under "Objectives Status", it states "Number of Objectives: 1" and "Objectives Proven Valid: 1".

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

- 4 Scroll back to the top of the browser window. In the **Table of Contents**, click **Proof Objectives Status**.

Objectives Proven Valid				
#:	Type	Model Item	Description	Counterexample
1	Custom Proof Objective	Proof Objective	Objective: 1	n/a

The Objectives Proven Valid table lists the proof objectives that the Simulink Design Verifier software proved to be valid.

- 5 Scroll down to view the Properties chapter or go to the top of the browser window and in the **Table of Contents**, click **Properties**.

Proof Objective	
Summary	
Model Item:	Proof Objective
Property:	Objective: 1
Status:	Proven valid

The Proof Objective summary indicates that the Simulink Design Verifier software proved an objective that you specified in your model. The Proof Assumption block restricts the domain of the input signals to the interval $[-1, 0]$. Therefore, the software proves that this interval does not contain values that are greater than zero, thereby satisfying the proof objective.

Analyzing Contradictory Models

If the analysis produces the error **The model is contradictory in its current configuration**, the software detected a contradiction in your model and it cannot analyze the model. You can have a contradiction if your model has Proof Assumption blocks with incorrect parameters. For example, an assumption could state that a signal must be between 0 and 5 when the signal is constant 10.

If the software detects a contradiction, all previous results are invalidated and the software reports that all the properties are falsified.

Proving Properties in a Subsystem

If you have a large model, you can prove the properties of a subsystem in the model and review the analyses in smaller, manageable reports. The workflow for proving properties in a subsystem is:

- 1 Open the model that contains the subsystem.
- 2 Make the subsystem atomic.
- 3 Run the Simulink Design Verifier software using the **Prove Properties of Subsystem** option.
- 4 Review the results.

The tutorial in “Analyzing a Subsystem” on page 1-26 explains how to generate test cases for the Controller subsystem in the Cruise Control Test Generation model. The steps for proving properties are similar to those for generating test cases, except that you select the **Prove Properties of Subsystem** option instead of the **Generate Tests for Subsystem** option.

Proving Complex Properties

Property-Proving Examples

The Simulink Design Verifier block library includes four Simulink examples that demonstrate how to prove complex properties:

- “Conditions that Trigger a Result” on page 8-31
- “Conditions That Cannot Be True Simultaneously” on page 8-32
- “Increasing or Decreasing Signals” on page 8-32
- “Conditions with One True Element” on page 8-33

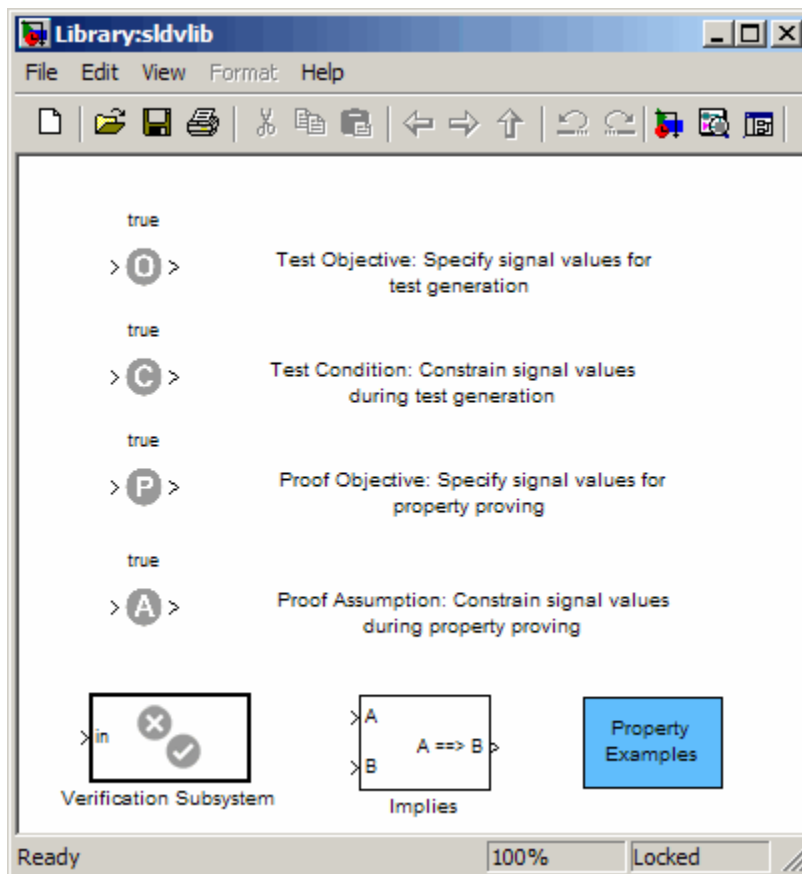
The workflow for using these examples in your model is:

- 1** Copy these examples into your verification subsystem.
- 2** Adapt them, if necessary, for the specific properties that you want to prove.
- 3** Run the Simulink Design Verifier analysis to prove that the assertions in these examples never fail.
- 4** If the assertion fails, the software creates a counterexample that causes the assertion to fail and then generates a harness model.
- 5** On the harness model, execute the counterexample to confirm that the assertion fails with that counterexample.

To view these examples:

- 1** Open the block library. Type:

```
sldvlib
```



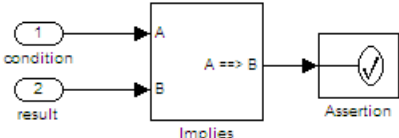
2 Double-click **Property Examples** to open the examples.

slvdemo_properties_spec

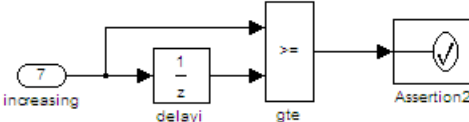
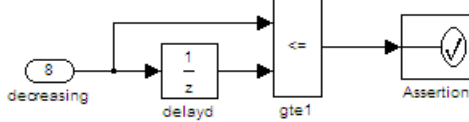
File Edit View Simulation Format Tools Help

10.0 Normal

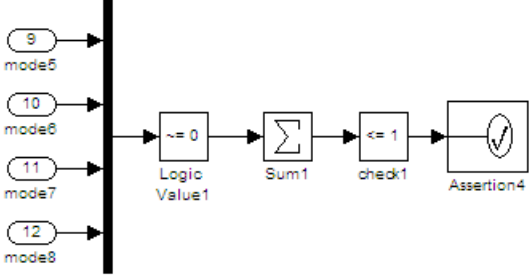
Simulink Design Verifier Property Examples



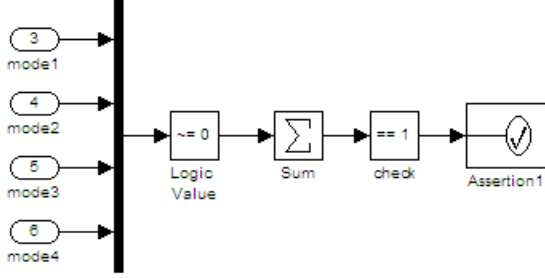
Implies operation describes conditions that should trigger a result.

Increasing and decreasing operations describe signals that should increase or decrease.



Exclusivity operation describes conditions that should never be true at same time.

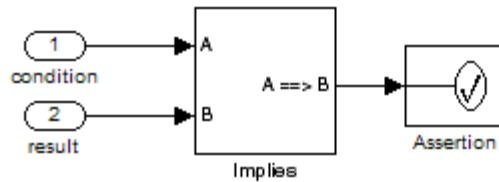


Mutual exclusivity operation describes conditions that should have exactly one true element.

Ready 100% FixedStepDiscrete

Conditions that Trigger a Result

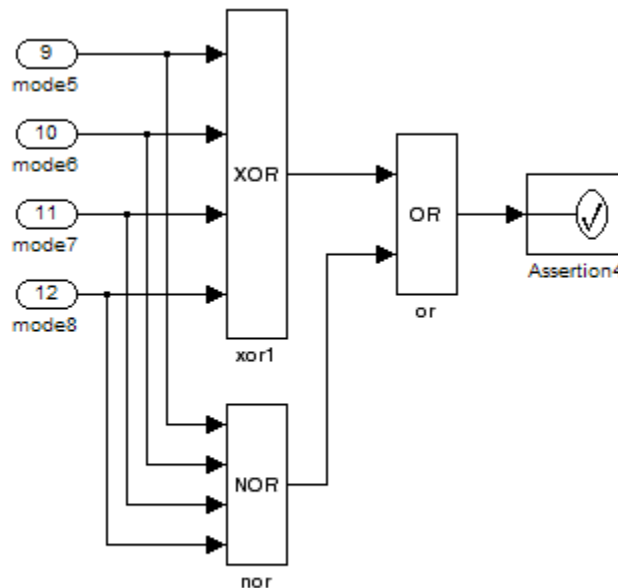
The Simulink Design Verifier Implies block allows you to test for conditions that trigger a result. This example specifies that if condition A is true, result B must always be true.



Implies operation describes conditions that should trigger a result.

Conditions That Cannot Be True Simultaneously

This example specifies that the four inputs must never all be true at the same time.

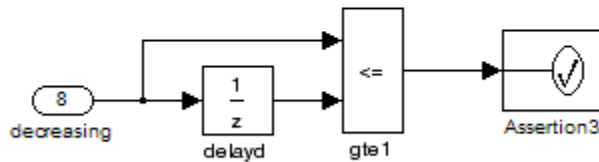
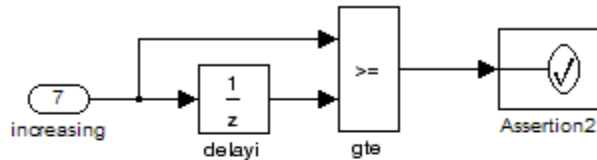


Exclusivity operation describes conditions that should never be true at same time.

Increasing or Decreasing Signals

The two examples in this section specify that a signal is either:

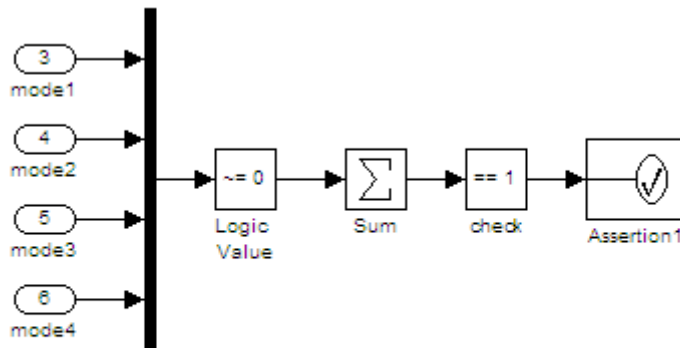
- Always increasing or staying constant
- Always decreasing or staying constant



Increasing and decreasing operations describe signals that should increase or decrease.

Conditions with One True Element

This example specifies that only one of the four input signals can be true.



Mutual exclusivity operation describes conditions that should have exactly one true element.

Reviewing the Results

- “Examining Simulink® Design Verifier Data Files” on page 9-2
- “Exploring Test Harness Models” on page 9-8
- “Creating a SystemTest TEST-File” on page 9-15
- “Understanding Simulink® Design Verifier Reports” on page 9-18

Examining Simulink Design Verifier Data Files

In this section...

“About Simulink® Design Verifier Data Files” on page 9-2

“Overview of the sldvData Structure” on page 9-2

“Model Information Fields in sldvData” on page 9-3

“Simulating Models with Simulink® Design Verifier Data Files” on page 9-7

About Simulink Design Verifier Data Files

When you enable the **Save test data to file** parameter (see “Results Pane” on page 6-14), the Simulink Design Verifier software generates a data file when it completes its analysis. The data file is a MAT-file that contains a structure named `sldvData`. This structure stores all the data the software gathers and produces during the analysis. Although the software displays the same data graphically in the test harness model and report, you can use the data file to conduct your own analysis or to generate a custom report.

Overview of the sldvData Structure

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

- 1 Generate test cases for the `sldvdemo_flipflop` model (see “Analyzing a Model” on page 1-6).
- 2 To load the data file, at the MATLAB prompt, enter the following command:

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

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

- 3 Enter `sldvData` to display the field names that constitute the structure:

```
sldvData =
```

```

ModelInformation: [1x1 struct]
AnalysisInformation: [1x1 struct]
  ModelObjects: [1x2 struct]
  Objectives: [1x12 struct]
  TestCases: [1x4 struct]
  Version: '1.4'

```

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

Model Information Fields in sldvData

The following sections describe the fields in the sldvData structure:

- “ModelInformation Field” on page 9-3
- “AnalysisInformation Field” on page 9-4
- “ModelObjects Field” on page 9-4
- “Objectives Field” on page 9-5
- “TestCases Field / CounterExamples Field” on page 9-5
- “Version Field” on page 9-7

ModelInformation Field

In the sldvData structure, the ModelInformation field contains information about the model you analyzed. The following table describes each subfield of the ModelInformation field.

Subfield Name	Description
Name	String specifying the model name.
Version	String specifying the model number.
Author	String specifying the user name.
SubsystemPath	String representing the full path name of the subsystem (if any) that was analyzed.

Subfield Name	Description
ExtractedModel	String representing the name of the model extracted to analyze the subsystem (if any) specified in SubsystemPath.
ReplacementModel	String specifying the name of the model (if any) that contains the block replacements.

AnalysisInformation Field

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

Subfield Name	Description
Status	String specifying the completion status of the Simulink Design Verifier analysis.
Options	Deep copy of the Simulink Design Verifier options object used during the analysis.
InputPortInfo	Cell array of structures specifying information about each Inport block in the top-level system.
OutputPortInfo	Cell array of structures specifying information about each Outport block in the top-level system.
SampleTimes	For internal use only.

ModelObjects Field

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

Subfield Name	Description
descr	String specifying the full path to a model object, including objects in a Stateflow chart.
typeDesc	String specifies the block type of the model object.

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

Objectives Field

In the s1dvData 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
type	String specifying the type of an objective.
status	String specifying the status of an objective.
descr	String specifying the description of an objective.
label	String specifying the label of an objective.
outcomeValue	Integer specifying an objective's outcome.
coveragePointIdx	Integer representing the index of a coverage point with which an objective is associated.
modelObjectIdx	Integer representing the index of a model object with which an objective is associated.
testCaseIdx	Integer representing the index of a test case or counterexample that addresses an objective.

TestCases Field / CounterExamples Field

In the s1dvData structure, this field can have two names, depending on the type of check:

- If you set the **Mode** parameter to `Test generation`, the `TestCases` field lists information about each test case, such as its signal values and the test objectives it achieves.
- If you set the **Mode** parameter to `Property proving`, the `CounterExamples` field lists information about each counterexample and the proof objective it falsifies.

The following table describes each subfield of the `TestCases / CounterExamples` 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.
<code>paramValues</code>	Structure specifying the parameter values associated with a test case or counterexample. Its fields include: <code>name</code> — String specifying the name of a parameter. <code>value</code> — Number specifying the value of a parameter. <code>noEffect</code> — Logical value specifying whether a parameter's value affects an objective.
<code>stepValues</code>	Vector specifying the number of time steps that comprise signals in a test case or counterexample.
<code>objectives</code>	Structure specifying objectives that a test case or a counterexample addresses. Its fields include: <code>objectiveIdx</code> — Integer representing the index of an objective that a test case achieves or a counterexample falsifies. <code>atTime</code> — Time value at which either a test case achieves an objective or a counterexample falsifies an objective. <code>atStep</code> — Time step at which either a test case achieves an objective or a counterexample falsifies an objective.

Subfield Name	Description
dataNoEffect	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.
expectedOutput	Cell array of vectors specifying the output values that result from simulating the model using the test case signals. Each cell represents the output values associated with a different Output block in the top-level system. This subfield is populated if you select Include expected output values .

Version Field

In the `sldvData` structure, the `Version` field is a string specifying the version of the Simulink Design Verifier software that verified the model.

Simulating Models with Simulink Design Verifier Data Files

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

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

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

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

For more information, see the `sldvruntest` reference page.

Exploring Test Harness Models

In this section...
“About Test Harness Models” on page 9-8
“Anatomy of a Test Harness” on page 9-8
“Configuration of the Test Harness” on page 9-13
“Simulating the Test Harness” on page 9-13

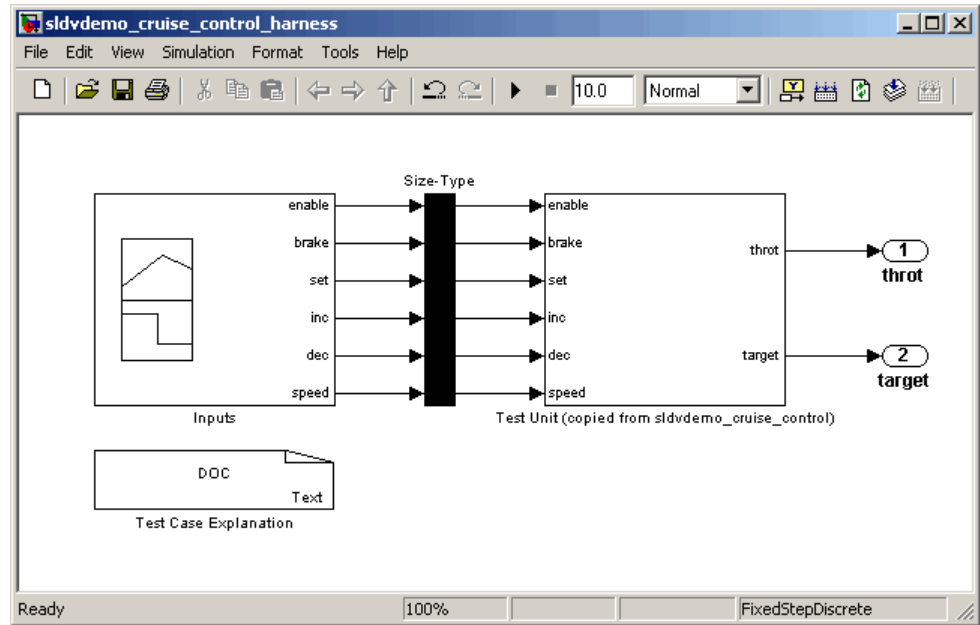
About Test Harness Models

When you enable the **Save test harness as model** parameter (see “Results Pane” on page 6-14), the Simulink Design Verifier software generates a test harness model after it completes its analysis. If the software’s **Mode** parameter specifies **Test generation**, the harness model contains test cases that achieve test objectives. Otherwise, the software’s **Mode** parameter specifies **Property proving** and the harness model contains counterexamples that falsify proof objectives.

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

Anatomy of a Test Harness

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



The harness model contains the following items:

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

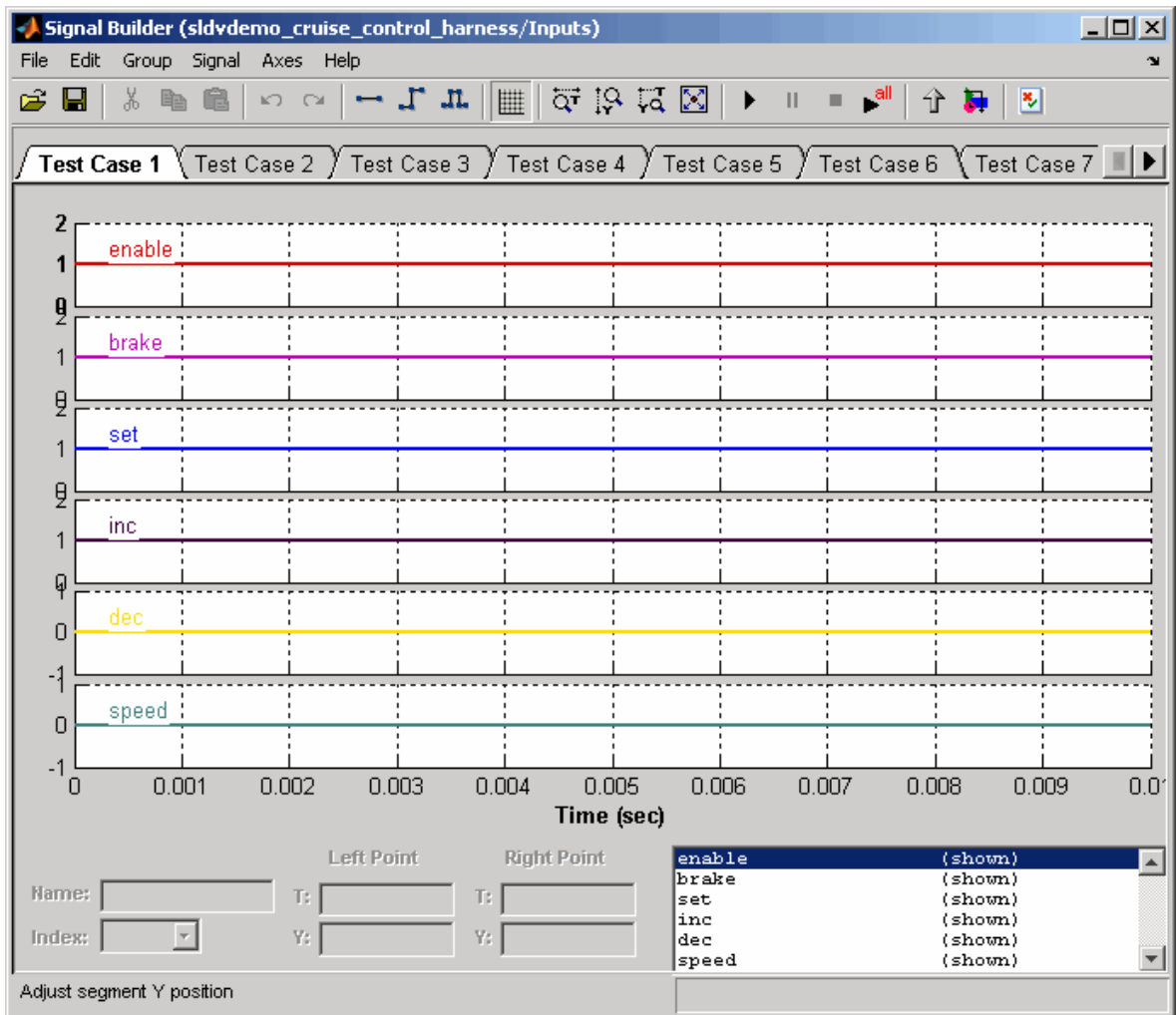
```

1 | Test Case 1 (8 Objectives)
2 |     Parameter values:
3 |
4 |     1. Controller/PI Controller - enable logical value F @ T=0.00
5 |     2. Controller/Switch1 - logical trigger input true (output is from 1s
6 |     3. Controller/Logical Operator1 - Logic: input port 1 T @ T=0.00
7 |     4. Controller/Logical Operator2 - Logic: input port 1 T @ T=0.00
8 |     5. Controller/Logical Operator2 - Logic: MCDC expression for output w
9 |     6. Controller/Logical Operator - Logic: input port 1 T @ T=0.00
10 |    7. Controller/Logical Operator - Logic: input port 2 F @ T=0.00
11 |    8. Controller/Logical Operator - Logic: MCDC expression for output wi
12 |
13 | Test Case 2 (3 Objectives)
14 |     Parameter values:
15 |
16 |     1. Controller/Logical Operator1 - Logic: input port 1 F @ T=0.00
17 |     2. Controller/Logical Operator - Logic: input port 1 F @ T=0.00
18 |     3. Controller/Logical Operator - Logic: MCDC expression for output wi
19 |
20 | Test Case 3 (6 Objectives)
21 |     Parameter values:

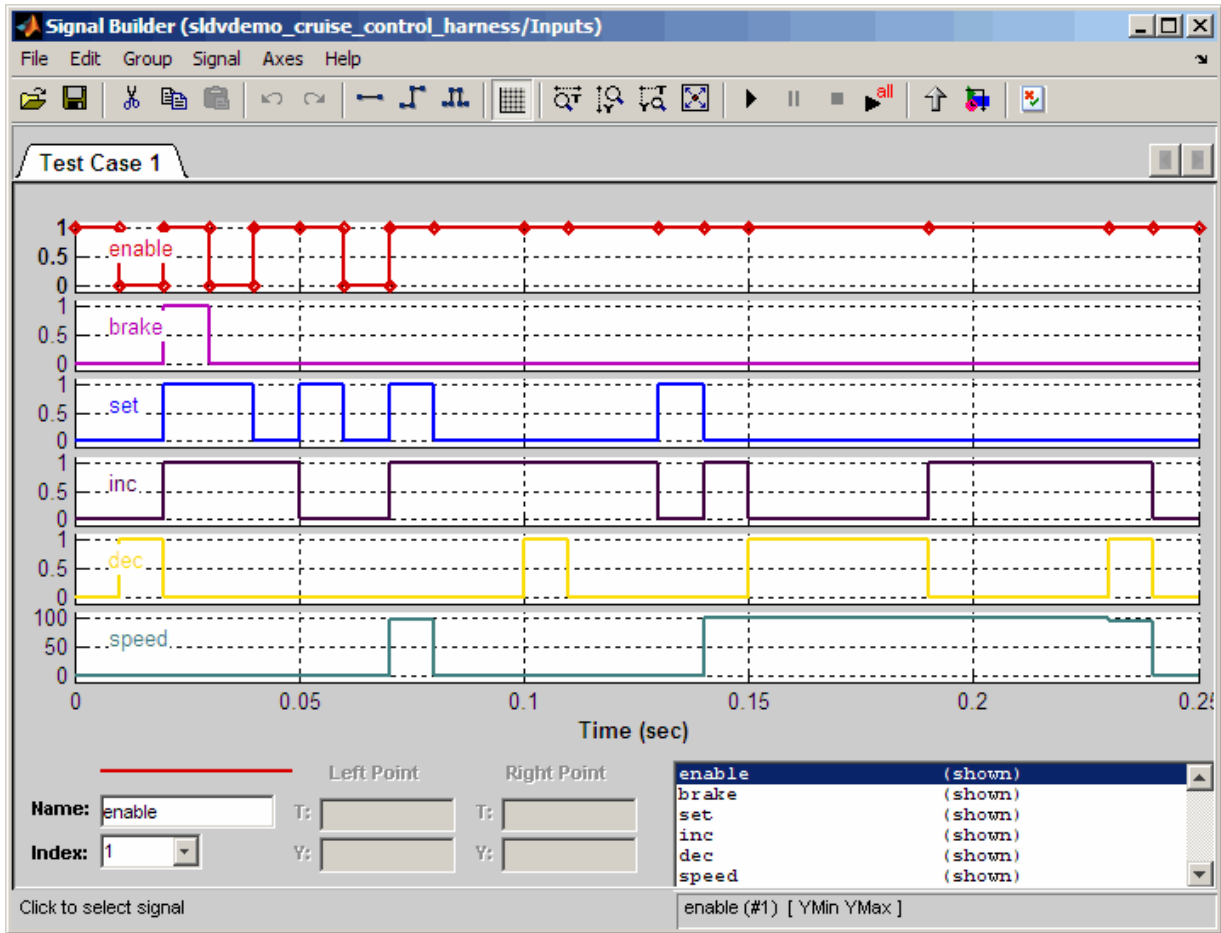
```

- **Inputs** — This Signal Builder block contains signals that comprise the test cases or counterexamples that the Simulink Design Verifier software generated. Double-click the Inputs block to open the Signal Builder dialog box and view its signals. The following Signal Builder block shows the signals for Test Case 8 after analyzing the `sldvdemo_cruise_control` model with the default options.

Each signal group represents a unique test case or counterexample. In the Signal Builder dialog box, select a tab to view the signals associated with a particular test case or counterexample.



If you select the Long test cases option of the **Test suite optimization** parameter, the analysis creates fewer, longer test cases. For example, if you select the Long test cases option for the `sldvdemo_cruise_control` model, the analysis produces 1 long test case instead of 10 shorter test cases. The following Signal Builder dialog box shows the signals for that test case.



Note For more information about the Signal Builder dialog box, see “Working with Signal Groups” in *Simulink User’s Guide*.

- **Size-Type** — This Subsystem block transmits signals from the Inputs block to the Test Unit block. It ensures that the signals are of the appropriate size and data type, which the Test Unit block expects.

- **Test Unit** — This Subsystem block contains a copy of the original model that the Simulink Design Verifier software analyzed.

Configuration of the Test Harness

After the Simulink Design Verifier software generates the test harness, it has the following settings:

- The test harness start time is always 0. If the original model uses a nonzero start time, the software ignores this and always uses 0 for the simulation start time for test cases and counterexamples.
- The test harness stop time always equals the stop time of the longest test case in the Signal Builder dialog box.
- By default, the software enables coverage reporting for test harness models that contain test cases. Although it enables coverage reporting with particular options selected, you can customize the settings to meet your needs. For more information, see “Model Coverage Reporting Options” in the *Simulink Verification and Validation User’s Guide*.

Simulating the Test Harness

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

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


To simulate a single test case or counterexample:

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

The Signal Builder dialog box appears.

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

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


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

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

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

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

The Signal Builder dialog box appears.

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

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

When you click **Run all**, the software simulates all the test cases using the stop time for the test harness model. The stop time equals the stop time for the longest test case, so you may accumulate additional coverage when you simulate the shorter test cases.

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

Creating a SystemTest TEST-File

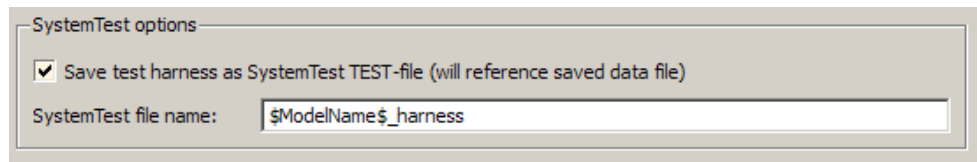
If you have installed the SystemTest software with your MATLAB application, you can specify that the Simulink Design Verifier software create a SystemTest TEST-file when it analyzes a model. Creating a TEST-file allows you to configure and collect model coverage results and run the test cases from inside the SystemTest environment.

Note The option to create a SystemTest TEST-file is only available in test-generation mode; you cannot create this file when running a property-proving analysis.

In addition, if you have a model with a large number of inputs, this feature eliminates the overhead of creating the test harness. However, you can create both a test harness and a TEST-file in the same analysis.

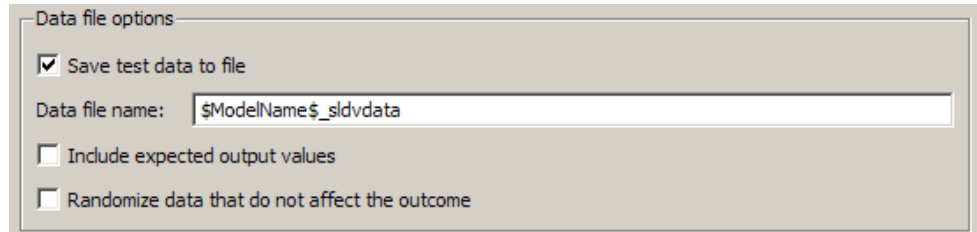
To create a TEST-file for the `sldvdemo_cruise_control` model, perform these steps:

- 1 Type `sldvdemo_cruise_control` at the MATLAB command prompt to open the Cruise Control Test Generation model.
- 2 Select **Simulation > Configuration Parameters** to open the Configuration Parameters dialog box.
- 3 In the **Select** pane, under **Design Verifier**, select **Results**.
- 4 On the **Results** pane, under **SystemTest options**, select **Save test harness as SystemTest TEST-file (will reference saved data file)**.

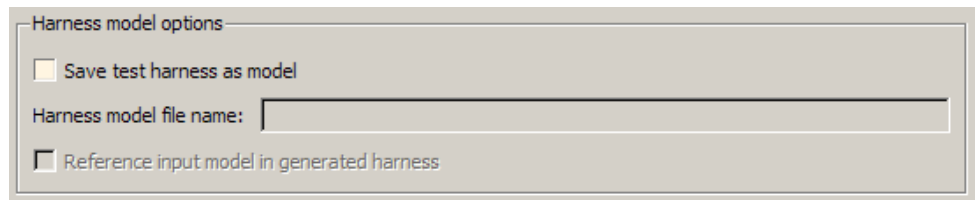


- 5 If you prefer a file name other than the default, specify the **SystemTest file name**.

- 6 Under **Data File options**, verify that **Save test data to file** is selected. You must select this option to generate a TEST-file.

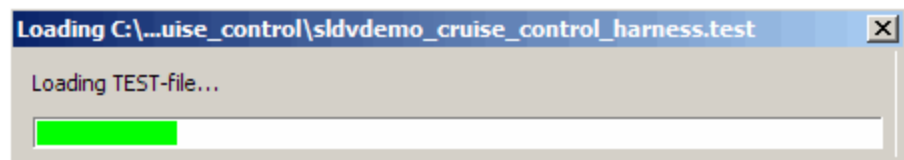


- 7 If you do not need the Simulink Design Verifier test harness in addition to the TEST-file, under **Harness model options**, clear **Save text harness as model**.



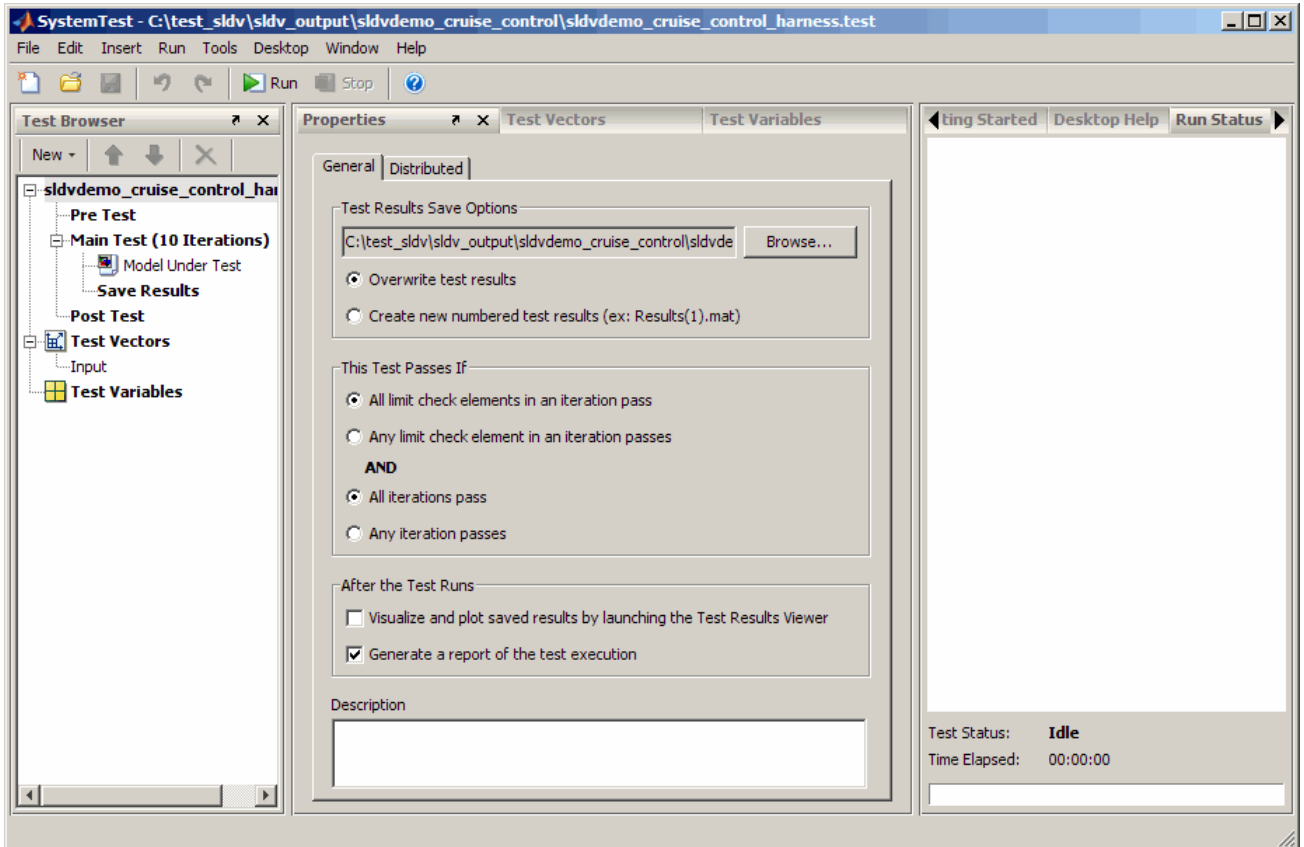
- 8 Click **Apply** and **OK** to save the changes and exit the Configuration Parameters dialog box.
- 9 Double-click the Run block in the `sldvdemo_cruise_control` model to start the analysis.

When the software is creating the TEST-file, the following status box appears.



When the analysis completes, the SystemTest desktop opens the TEST-file, which, for this example, is saved as


```
matlabroot\sldvdemo_output\sldv_cruise_control\sldvdemo_cruise_control_harness.test
```



In the **Test Browser** pane, the 10 iterations under Main Test correspond to the 10 test cases the Simulink Design Verifier software generates and describes in the Test Case Explanation block of the test harness.

For information about running the test cases using the SystemTest software, see “Creating a Simulink Design Verifier Data File Test Vector” in the *SystemTest User’s Guide*.

Understanding Simulink Design Verifier Reports

In this section...

“About Simulink® Design Verifier Reports” on page 9-18

“Front Matter” on page 9-18

“Summary Chapter” on page 9-19

“Analysis Information Chapter” on page 9-20

“Test / Proof Objectives Status Chapter” on page 9-25

“Model Items Chapter” on page 9-29

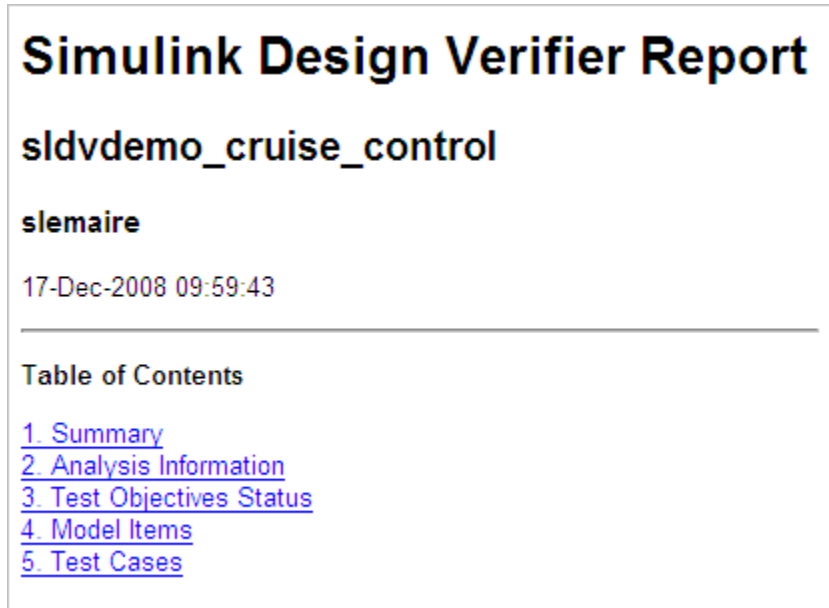
“Test Cases / Properties Chapter” on page 9-30

About Simulink Design Verifier Reports

When you enable the **Generate report of the results** parameter (see “Report Pane” on page 6-17), the Simulink Design Verifier software generates an HTML report after it completes its analysis. If the software’s **Mode** parameter specifies **Test generation**, the report describes the model’s test objectives and any corresponding test cases that result from the analysis. Otherwise, the software’s **Mode** parameter specifies **Property proving**, and the report describes the model’s proof objectives and any counterexamples that result from the analysis.

Front Matter

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



Simulink Design Verifier Report

sldvdemo_cruise_control

slemaire

17-Dec-2008 09:59:43

Table of Contents

- [1. Summary](#)
- [2. Analysis Information](#)
- [3. Test Objectives Status](#)
- [4. Model Items](#)
- [5. Test Cases](#)

The title section lists the following information:

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

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

Summary Chapter

The Summary chapter of the HTML report provides an overview of the Simulink Design Verifier analysis.

Chapter 1. Summary

Analysis Information

Model: sldvdemo_cruise_control
Mode: TestGeneration
Status: Completed normally

Objectives Status

Number of Objectives: 34
Objectives Satisfied: 34

Analysis Information Chapter

The Analysis Information chapter of the HTML report includes the following sections:

- “Model Information” on page 9-20
- “Analysis Options” on page 9-21
- “Unsupported Blocks” on page 9-22
- “Constraints” on page 9-22
- “Block Replacements Summary” on page 9-23
- “Approximations” on page 9-24

Model Information

The Model Information section provides the following information about the current version of the model:

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

Model Information

File:	C:\test_sldv\slvdvdemo_flipflop.mdl
Version:	1.15
Time Stamp:	Fri Jun 27 15:38:15 2008
Author:	slemaire

See “Managing Model Versions” in *Simulink User’s Guide* for details about specifying this information for your models.

Analysis Options

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

The Analysis Options section lists the parameters that affected the Simulink Design Verifier analysis. See “Configuring Simulink® Design Verifier Options” on page 6-5 for more information about the parameters that this section displays.

Analysis Options

Mode:	TestGeneration
Test Suite Optimization:	CombinedObjectives
Maximum Testcase Steps:	500 time steps
Test Conditions:	UseLocalSettings
Test Objectives:	UseLocalSettings
Model Coverage Objectives:	MCDC
Maximum Processing Time:	60s
Block Replacement:	on
Block Replacement Rules:	<FactoryDefaultRules>
Parameters Analysis:	on
Parameters Configuration File:	sldv_params_template.m
Save Data:	on
Save Harness:	on
Save Report:	on

Unsupported Blocks

If your model includes unsupported elements, you can turn on automatic stubbing to allow the analysis to proceed. If you turn on automatic stubbing, the software considers only the interface of the unsupported elements, not their actual behavior. This technique allows the software to complete the analysis. However, the analysis may achieve only partial results if any of the unsupported model elements affect the simulation outcome.

The Unsupported Blocks section appears only if the analysis stubbed any unsupported elements; it lists the unsupported elements in a table, with a link to the element in the model.

Unsupported Blocks

The following blocks are not supported by Simulink Design Verifier. They were abstracted during the analysis. This can lead Simulink Design Verifier to produce only partial results for parts of the model that depends on the output values of these blocks.

Block	Type
Trigonometric Function	Trigonometry

For more information about automatic stubbing, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

Constraints

The Constraints section provides information about any test conditions that the Simulink Design Verifier software applied when it analyzed a model.

You can locate the constraint in your model by clicking constraint; the software highlights the corresponding Test Condition block in your model window and opens a new window showing the block in detail.

Constraints	
Name	Constraint
constraint	[0, 100]

Block Replacements Summary

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

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

- Name of the M-file that represents the block replacement rule and the value of the `BlockType` parameter the rule specifies
- Description of the rule that the `MaskDescription` parameter of the replacement block specifies
- Names of any blocks that the Simulink Design Verifier software replaced in the model

To locate a particular block replacement in your model, click on the name for that replacement in the Replaced Blocks column of the table; the software highlights the affected block in your model window and opens a new window that displays the block in detail.

Block Replacements Summary

Table 2.1. Block Replacements

#:	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_switch_normal.m /Switch	Inserts test objectives for switch blocks that require each switch position be demonstrated when the values of input ports 1 and 3 differ.	Switch
3	sldvdemo_custom_blkrep_rule_sqrt.m /Math	Approximates the mathematical function sqrt using lookup table. The input range is constrained to [0 10000].	Math Function

See Chapter 4, “Working with Block Replacements” for more information.

Approximations

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

Approximations

Simulink Design Verifier performed the following approximations during analysis. These can impact the precision of the results generated by Simulink Design Verifier. Please see the product documentation for further details.

	Type	Description
1	Rational approximation	The model includes floating-point arithmetic. Simulink Design Verifier approximates floating-point arithmetic with rational number arithmetic.

Note Review the analysis results carefully when the software uses approximations. In rare cases, an approximation may result in test cases that fail to achieve test objectives or counterexamples that fail to falsify proof objectives. For example, a floating-point-roundoff error might prevent a signal from exceeding a designated threshold value.

Test / Proof Objectives Status Chapter

The Test / Proof Objectives Status chapter of the HTML report 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 chapter displays each objective in one of the following tables associated with the objective's status:

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

Objectives Undecided

Simulink Design Verifier was not able to process these objectives with the current options.

#:	Type	Model Item	Description	Counterexample
1	Custom Proof Objective	Verify Output/FoutCorrect	Objective: T	n/a
2	Custom Proof Objective	Verify Output/ToutCorrect	Objective: T	n/a

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

Objectives Producing Errors

#:	Type	Model Item	Description	Test Case
4	Decision	Mode switch	logical trigger input true (output is from 1st input port)	n/a
8	Decision	Basic Roll Mode/Integrator	integration result <= lower limit T	n/a
10	Decision	Basic Roll Mode/Integrator	integration result >= upper limit T	n/a

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

- **Objectives Proven Unsatisfiable** — Lists the test objectives that the Simulink Design Verifier software determined could not be satisfied. In this example, the software found that there are no test cases that achieve these objectives.

Objectives Proven Unsatisfiable

Simulink Design Verifier proved that there does not exist any test case exercising these test objectives. This often indicates the presence of dead-code in the model. Other possible reasons can be inactive blocks in the model due to parameter configuration or test constraints such as given using Test Condition blocks. In rare cases, the approximations performed by Simulink Design Verifier can make objectives impossible to achieve.

#:	Type	Model Item	Description	Test Case
1	Custom Test Objective	True	Objective: 100	n/a

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

Objectives Satisfied

Simulink Design Verifier found test cases that exercise these test objectives.

#:	Type	Model Item	Description	Test Case
1	Decision	PI Controller	enable logical value F	2
2	Decision	PI Controller	enable logical value T	1
3	Decision	P Controller	enable logical value F	1
4	Decision	P Controller	enable logical value T	2
5	Decision	mp switch	integer input value = 1 (output is from input port 2)	1
6	Decision	mp switch	integer input value = 2 (output is from input port 3)	2

- **Objectives Satisfied - No Test Case** — Lists test objectives that the Simulink Design Verifier software satisfied without generating test cases. In the following example, nonlinear computation causes the signal to the control input of the Saturation block to be out of range:

Objectives Satisfied - No Test Case

#:	Type	Model Item	Description	Test Case
2	Decision	Saturation	input > lower limit T	n/a
3	Decision	Saturation	input >= upper limit F	n/a
4	Decision	Saturation	input >= upper limit T	n/a

Situations when the software might satisfy an objective but not create a test case are:

- Test objectives that depend on nonlinear computation
- If the test objective creates an arithmetic error, such as division by zero
- If you enable automatic stubbing, and the analysis encounters an unsupported block whose operation it does not understand

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

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

Objectives Proven Valid

#:	Type	Model Item	Description	Counterexample
1	Custom Proof Objective	Proof Objective	Objective: 1	n/a

- **Objectives Falsified with Counterexamples** — Lists the proof objectives that the Simulink Design Verifier software disproved. In this example, the software generated at least one counterexample that falsifies the specified objectives.

Objectives Falsified with Counterexamples

#:	Type	Model Item	Description	Counterexample
1	Assert	Verify True Output/Assertion	assert	1

- **Objectives Falsified - No Counterexample** — Lists the proof objectives that the Simulink Design Verifier software disproved without generating counterexamples. This occurs if, for example, you specified a proof objective on a signal whose value the software cannot control, or the software encountered a divide-by-zero error when instantiating a counterexample.

Objectives Falsified - No Counterexample

#:	Type	Model Item	Description	Counterexample
1	Custom Proof Objective	Proof Objective	Objective: F	n/a
2	Custom Proof Objective	Proof Objective1	Objective: T	n/a

Model Items Chapter

The Model Items chapter of the HTML report includes a table for each object in the model that defines coverage objectives. The table for a particular object lists all of the associated objectives, the objective types, objective descriptions, and the status of each objective at the end of the analysis.

The table for an individual object in the model will look similar to this one for the TK switch in the Roll Reference subsystem.

To highlight a given object in your model, click **View** at the upper-left corner of the table; the software opens a new window that displays the object in detail. To view the details of the test case that was applied to a specific objective, click the test case number in the last column of the table.

Roll Reference/TK switch

[View](#)

#:	Type	Description	Status	Test Case
27	Decision	logical trigger input false (output is from 3rd input port)	Produced error	n/a
28	Decision	logical trigger input true (output is from 1st input port)	Satisfied	1

Test Cases / Properties Chapter

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

- “Test Cases” on page 9-30
- “Properties” on page 9-35

Test Cases

If the Simulink Design Verifier software's **Mode** parameter specifies **Test** generation, the report's Test Cases chapter includes sections that summarize the test cases the analysis generated, one per test case. This section uses the results of analyzing the `sldvdemo_cruise_control` model.

Table of Contents. Each Test Cases section contains a table of contents. Each item in the table of contents links to information about a specific test case.

Summary. The Summary section lists:

- Length of the signals that comprise the test case
- Total number of test objectives that the test case achieves

Summary

Length: 0.06 Seconds (7 sample periods)
Objective Count: 1

Objectives. The Objectives section lists:

- The time step at which the test case achieves that objective.
- The time at which the test case achieves that objective.
- A link to the model item associated with that objective. Clicking the link highlights the model item in the Model Editor.
- The objective that was achieved.

Objectives

Step	Time	Model Item	Objectives
7	0.06	Controller/PI Controller/Discrete-Time Integrator	integration result \geq upper limit T

Generated Input Data. For each input signal associated with the model item, the Generated Input Data section lists the time step and corresponding time at which the test case achieves particular test objectives. If the signal value does not change over those time steps, the table lists the time step and time as ranges.

Generated Input Data

Time	0	0.01-0.02	0.03	0.04-0.05	0.06
Step	1	2-3	4	5-6	7
enable	1	1	1	1	1
brake	0	0	0	0	0
set	1	0	0	0	1
inc	1	1	1	1	-
dec	0	0	1	0	-
speed	97	0	0	0	0

Note The Generated Input Data table displays a dash (–) instead of a number as a signal value when the value of the signal at that time step does not affect the test objective. In the test harness model, the Inputs block represents these values with zeros unless you enable the **Randomize data that does not affect outcome** parameter (see “Randomize data that does not affect outcome” on page 6-16).

Expected Output. If you select the **Include expected output values** on the **Design Verifier, Results** pane of the Configuration Parameters dialog box, the report includes the Expected Output section for each test case. For each output signal associated with the model item, this table lists the expected output value at each time step.

Expected Output These output values are expected assuming that inputs that do not affect the test objectives (- in the table above) are given a default value - 0 for numeric types, and default value for enumerated types.

Time	0	0.01	0.02	0.03	0.04	0.05	0.06
Step	1	2	3	4	5	6	7
throt	0	1.96	1.9898	2.0197	2.0497	2.0798	0.05
target	97	98	99	100	101	102	0

Combined Objectives. If you set the **Test suite optimization** option to `CombinedObjectives` (the default), the Test Cases chapter may include individual information about many test cases.

Chapter 5. Test Cases

Table of Contents

- [Test Case 1](#)
- [Test Case 2](#)
- [Test Case 3](#)
- [Test Case 4](#)
- [Test Case 5](#)
- [Test Case 6](#)
- [Test Case 7](#)
- [Test Case 8](#)
- [Test Case 9](#)
- [Test Case 10](#)

This section contains detailed information about each generated test case.

Test Case 1

Summary

Length: 0 Seconds (1 sample periods)

Objective Count: 3

Long Test Cases. If you set the **Test suite optimization** option to LongTestcases, the Test Cases chapter in the report includes fewer sections about longer test cases.

Chapter 5. Test Cases

Table of Contents

[Test Case 1](#)

This section contains detailed information about each generated test case.

Test Case 1

Summary

Length: 0.23 Seconds (24 sample periods)
Objective Count: 34

Properties

If the Simulink Design Verifier software's **Mode** parameter specifies Property proving, the report's Properties chapter includes a series of sections that summarize the proof objectives and any counterexamples the software generated.

If the software proves an objective is valid, this report chapter displays a summary section similar to this one.

Proof Objective

Summary

Model Item: [Proof Objective](#)
Property: Objective: 1
Status: Proven valid

If the software falsifies an objective, this report chapter has a summary section similar to the one in the following figure.

To highlight the proof objective in your model, click the Model Item name in the Summary section.

Verify True Output/Assertion

Summary

Model Item: [Verify True Output/Assertion](#)
Property: assert
Status: Falsified

Counter Example

Time0	
Step 1	
raw	128

Analyzing Large Models and Improving Performance

- “Sources of Model Complexity” on page 10-2
- “Analyzing a Large Model” on page 10-3
- “Generating Reports for Large Models” on page 10-8
- “Managing Model Data to Simplify the Analysis” on page 10-9
- “Partitioning Model Inputs and Generating Tests Incrementally” on page 10-13
- “Analyzing the Model Using a Bottom-Up Approach” on page 10-15
- “Analyzing Logical Operations” on page 10-16
- “Handling Models with Large State Spaces” on page 10-17
- “Handling Problems with Counters and Timers” on page 10-18
- “Techniques for Proving Properties of Large Models” on page 10-20

Sources of Model Complexity

Some model characteristics can cause problems with a Simulink Design Verifier analysis in the following ways:

- Complexity of model inputs due to:
 - Large number of inputs (The number of inputs can vary, depending on the individual model.)
 - Types of inputs (floating-point values, for example)
 - The way the inputs affect the model state and the objectives of the analysis
- Number of possible simulation paths through a model
- Portions of the model that cannot be reached
- Large signal count in the model

The following sections describe techniques designed to reduce the impact of this complexity and achieve the best performance from the Simulink Design Verifier software.

Most of these techniques focus on test generation for large models. However, you can use many of them to prove the properties of a large model and generate counterexamples when a property is disproved. In addition, “Techniques for Proving Properties of Large Models” on page 10-20 describes specific techniques for proving properties in a large model.

Analyzing a Large Model

In this section...
“Types of Large Model Problems” on page 10-3
“Using the Default Parameter Values” on page 10-4
“Modifying the Analysis Parameters” on page 10-5
“Using the Large Model Optimization” on page 10-6
“Stopping the Analysis Before Completion” on page 10-6

Types of Large Model Problems

The Simulink Design Verifier software may encounter some of these problems when analyzing a large model:

- Unsatisfiable objectives — The software proved there are no test cases that exercise these test objectives, and thus did not generate any test cases.
- Undecided objectives — The software was not able to satisfy or falsify these objectives.
- Objectives with errors — The most common error occurs when a model component uses nonlinear arithmetic, which can affect a test objective.
- Cannot complete the analysis in the time allotted — This problem may indicate an area of your model where the software encountered problems, or you may need to increase value of the **Maximum analysis time** parameter.
- Analysis hangs — If the number of objectives processed remains constant for a considerable length of time, the software has likely encountered complexity between the model and its objectives.
- Does not achieve a high percentage of model coverage — When you ran the test cases on the test harness, the percentage of model coverage was insufficient for your design.

The next few sections describe the initial steps to take when analyzing a large model. Although these steps address test generation, you can use a similar approach when proving properties in a model.

Using the Default Parameter Values

When you generate test cases for a model, whether large or small, the first step is to analyze the model using the Simulink Design Verifier default parameter values:

- 1** Check to see if your model is compatible with the Simulink Design Verifier software, as described in Chapter 3, “Ensuring Compatibility with the Simulink® Design Verifier Software”.
- 2** Using the default parameter values, analyze the model. The following table lists three of the default parameter values.

Parameter	Default Value	Description
Maximum analysis time	600 (seconds)	If the analysis does not finish within the specified time, the analysis times out and terminates.
Test suite optimization	Combined objectives	Generates test cases that address more than one test objective (if possible).
Model coverage objectives	MCDC	Generates test cases that achieve modified condition/decision coverage (MCDC), which includes decision coverage (DC) and condition coverage (CC).

- 3** Review the following information in the Simulink Design Verifier log window while the analysis runs:
 - Number of objectives processed — How many objectives were processed? Did the analysis hang after processing a certain number of objectives? The answers to these questions might give you a clue about where a problem might lie.
 - Number of objectives satisfied/Number of objectives falsified — Which objectives were falsified?

- Time elapsed — Did the analysis time out, or did it finish within the specified maximum analysis time?
- 4 When the analysis completes, review the Simulink Design Verifier report. This report contains links to the model elements for satisfied and falsified objectives so you can see what portions of the model might have problems.
 - 5 If all the test objectives have been satisfied, run the test cases on the test harness to determine model coverage.

If model coverage is sufficient, you do not need to do anything else. If the coverage is not sufficient, take additional steps to improve the analysis performance, as described in the following sections.

Note A large percentage of falsified objectives and poor model coverage often indicates that you need to change model parameter values to get complete coverage. This occurs when you have tunable parameters in Constant blocks that are connected to enabled subsystems or the trigger input of Switch blocks. In these situations, configure Simulink Design Verifier parameter support as described in Chapter 5, “Specifying Parameter Configurations”.

Modifying the Analysis Parameters

If the analysis satisfied most but not all of the objectives, try the following steps:

- 1 Increase the **Maximum analysis time** parameter. Such an increase gives the analysis more time to satisfy all the objectives.
- 2 Set the **Model coverage objectives** parameter to **Decision**. Selecting this option generates only test cases that achieve decision coverage. These test cases are a subset of the **MCDC** option.
- 3 Rerun the analysis and review the report.

If the results are not satisfactory, try the techniques described in the following sections.

Using the Large Model Optimization

Set the **Test suite optimization** parameter to `Large model`, and rerun the Simulink Design Verifier analysis.

The large model optimization strategy is designed for large, complex models. It may or may not improve the results of your analysis enough to fully test your design.

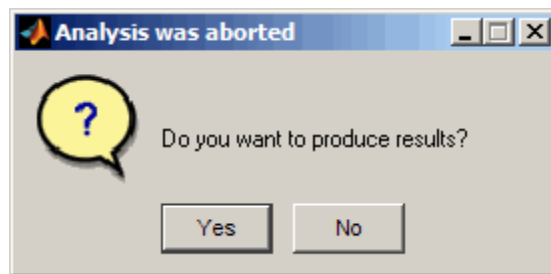
If there are outstanding test cases you want the software to generate, or additional properties you need to prove, continue with the following techniques.

Stopping the Analysis Before Completion

Watch the **Objectives processed** value in the log window. If about 50 percent of the **Maximum analysis time** parameter has elapsed and this value does not increase, the model analysis may have trouble processing certain objectives. If the analysis does not progress, take the following steps:

- 1 Click **Stop** in the log window.

The following dialog box opens.



- 2 Click **Yes** to save the results.

The software creates a test harness and an HTML report.

- 3 Review the results. In the HTML report, review the **Objectives Undecided when the Analysis was Stopped** and **Objectives Producing Errors** sections to identify the model elements that are causing problems.

- 4** Review the model elements that have undecided objectives or objectives with errors to see if any of the following problems are present. Consult the respective sections for specific techniques to improve the analysis:
- Floating-point inputs
See “Managing Model Data to Simplify the Analysis” on page 10-9.
 - Nonlinear operations
See “Analyzing the Model Using a Bottom-Up Approach” on page 10-15 and “Analyzing Logical Operations” on page 10-16.
 - Large state spaces
See “Handling Models with Large State Spaces” on page 10-17.
 - Large timers and time delays
See “Handling Problems with Counters and Timers” on page 10-18.

Generating Reports for Large Models

When you analyze a model with a large root-level input signal count, you may encounter an insufficient memory error when the Simulink Design Verifier software is generating the report.

When this occurs, you need to increase the amount of memory the Sun Java Virtual Machine (JVM™) software can allocate. For steps on how to increase this memory, see “Increasing the MATLAB JVM Memory Allocation Limit” in the MATLAB® Report Generator™ documentation.

Managing Model Data to Simplify the Analysis

In this section...
“Simplifying Data Types” on page 10-9
“Constraining Data” on page 10-9

Simplifying Data Types

One way to simplify your model is to use for the designated signal data type a data type requiring the smallest space for the expected data. For example, do not use an `int` data type for Boolean data, because only 1 bit is required for Boolean data.

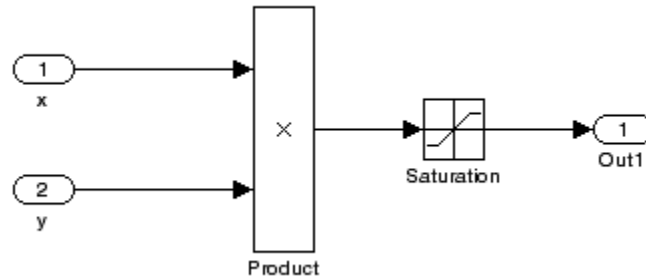
In another example, suppose you have a Sum block with two inputs that are always going to be integers between -10 and 10 . In this example, set the **Output data type** parameter to `int8`, rather than `int32` or `double`, or any other data type that requires more space than necessary.

To display the signal data types in the model window, select **Format > Port/Signal Displays > Port Data Types**.

Constraining Data

Another effective technique for reducing complexity is to restrict the inputs to a set of representative values or, ideally, a single constant value. This process, called *discretization*, treats the input as if it were an enumeration. Discretization allows you to handle nonlinear arithmetic from multiplication and division in the simplest way possible.

The following model has a Product block feeding a Saturation block.

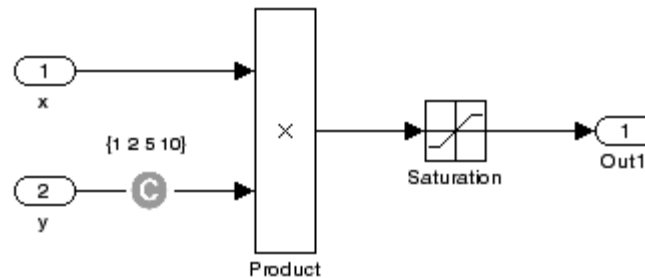


The Simulink Design Verifier software generates errors when attempting to satisfy the upper and lower limits of the Saturation block, because the software does not support nonlinear arithmetic. To work around these errors, restrict one of the inputs to a set of discrete values.

Identify discrete values that are required to satisfy your testing needs. For example, you may have an input for model speed, and your design contains paths of execution that are conditioned on speed above or below thresholds of 80, 150, 600, and 8000 RPM. For an effective analysis, constrain speed values to be 50, 100, 200, 1000, 5000, or 10000 RPM so that every threshold can be either active or inactive.

If you need to use more than two or three values, consider specifying the constrained values using an expression like `num2cell(minval:increment:maxval)`.

Using the previous example model, restrict the second input (*y*) to be either 1, 2, 5, or 10 using the Test Condition block as shown. The Simulink Design Verifier software produces test cases for all inputs.



You can also constrain signals that are intermediate or output values of the model. Constraining such signals makes it easier to work around multiplication or divisions inside lower-level subsystems that do not depend on model inputs.

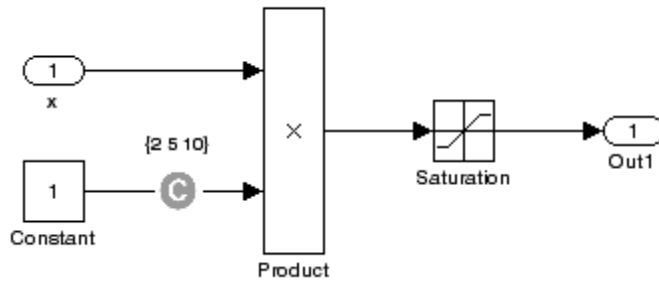
Note Discretization is best limited to a small number of inputs (less than 10). If your model requires discretization of many inputs, try to achieve model coverage through successive simulations as described in “Partitioning Model Inputs and Generating Tests Incrementally” on page 10-13.

Test Condition blocks do not need to be placed exactly on the inputs. In deciding where to place the constraints in your model, consider the following guidelines:

- Favor constraints on the input values because the software can process inputs easier.
- If you need to place constraints on both the input and the output, for example, to avoid nonlinear arithmetic, one of the constraints should be a range such as [minval maxval]. The software first tests the values at both ends of the range and can return a test case, even if the underlying calculations are nonlinear.
- Make sure that constraints at corresponding input and output points are not contradictory. Do not constrain the output signals to values that are not achievable because of the constraints on the input values.

- Avoid creating constraints that contradict the model. Such contradictions occur when a constraint can never be satisfied because it contradicts some aspect of the model or another constraint. Analyzing contradictory models can cause the Simulink Design Verifier software to hang.

The next figure shows a simple example of a contradictory model. The second input to the Multiply block is the constant 1, but the Test Condition block constrains it to a value of 2, 5, or 10. The software cannot achieve all the test objectives in this model.



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

Partitioning Model Inputs and Generating Tests Incrementally

As described in “Constraining Data” on page 10-9, you can constrain the values of model inputs using the Simulink Design Verifier Test Condition block.

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.

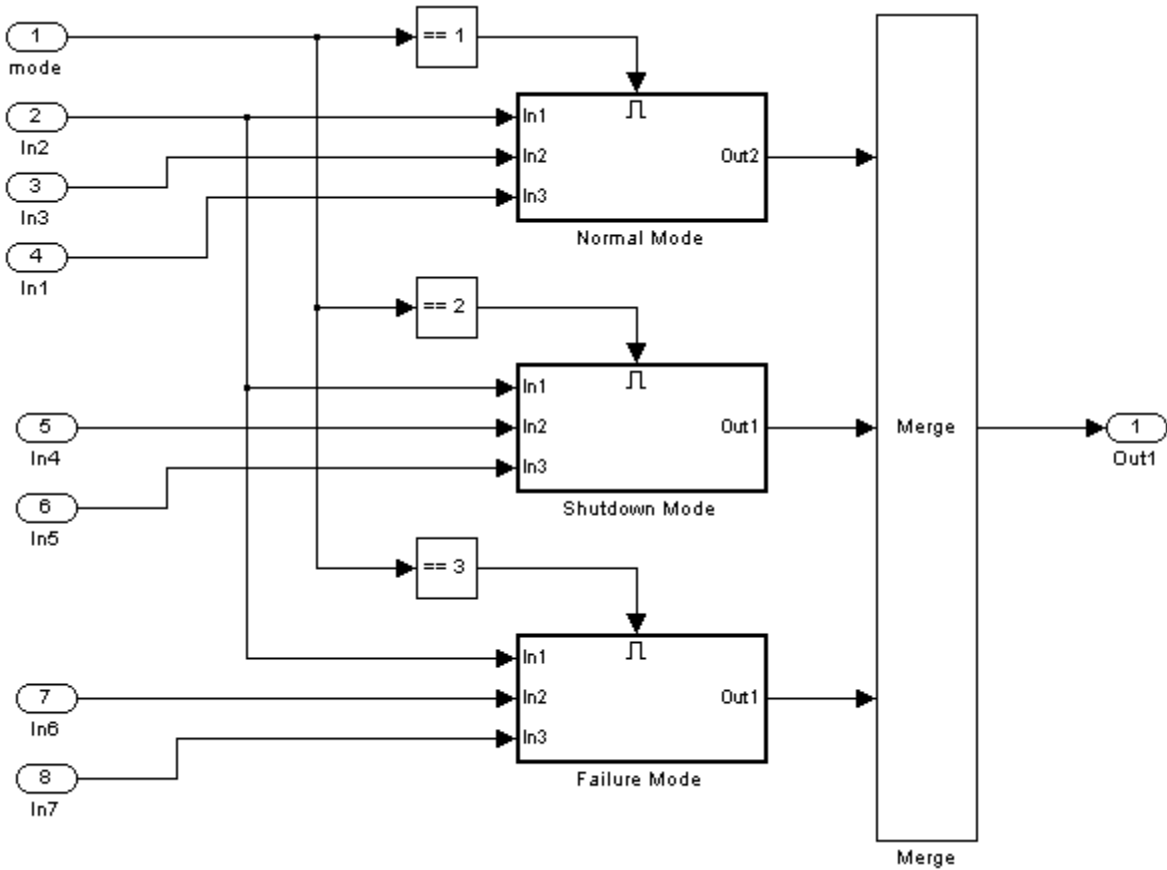
As an advanced technique for experienced MATLAB programmers, you can use parameterized constraints and successive runs of the Simulink Design Verifier software to implement an incremental test-generation technique:

- 1** Partition model inputs so that some are held constant, some are constrained to sets of constants using the Test Condition block, and some are free.
- 2** Generate test cases and run those test cases to collect model coverage.
- 3** Choose new values and partition the inputs with these new values.
- 4** Generate test cases for missing coverage using `sldvgencov` and the current test coverage.

Note The Extending an Existing Test Suite demo shows how to extend a test suite so that it satisfies missing model coverage.

- 5** Repeat steps 3 and 4 until you have generated sufficient coverage.

Partition the model inputs that enable further simplification when an analysis runs. Consider the following model, which has three mutually independent enabled subsystems—Normal Mode, Shutdown Mode, and Failure Mode.



You can incrementally generate test cases for each subsystem by constraining the first input to the appropriate constant value before running an analysis. In this way, as you create test cases for each subsystem, the software ignores the complexity of the other two subsystems.

Analyzing the Model Using a Bottom-Up Approach

Simulink Design Verifier software works most effectively at analyzing large models using a bottom-up approach. In this approach, the software analyzes smaller model components first, which can be faster than using the Large model test suite optimization.

The bottom-up approach offers several advantages:

- It allows you to solve the problems that slow down test generation or property proving in a controlled environment.
- Solving problems with small model components before analyzing the model as a whole is more efficient, especially if you have unreachable components in your model that you can only discover in the context of the model.
- You can iterate more quickly—find a problem and fix it, find another problem and fix it, and so on.
- If one model component has a problem, for example, it's unreachable, that situation can prevent the software from generating tests for *all* the objectives in a large model.

Try this workflow with your large model:

- 1** Break down the model into components of 100–1000 objectives each. Use the `sldvextract` function to extract components into a new model for analysis purposes.
- 2** Analyze the individual components, starting with the lowest level subsystems.
- 3** Fix any problems by adding constraints or specifying block replacements.
- 4** After you analyze the smaller components, reapply the necessary constraints and substitutions to the original model and analyze the full model.

When you finish a bottom-up analysis, you should have a top-level model that the Simulink Design Verifier software can analyze quickly.

Analyzing Logical Operations

If you have a model with both logical and arithmetic operations, consider analyzing only the logical operations.

The Simulink Design Verifier software does not support nonlinear arithmetic of floating-point numbers, as occurs with multiplication or division, unless one of the multiply operands or the divisor is a constant.

To simplify models that contain integers or floating-point numbers, the software maps the model computations into expressions of Boolean variables. For example, the software might represent an 8-bit number as a set of 8 Boolean values, with one for each digit. It might represent a bitwise OR operation of two 8-bit integers as 8 separate logical OR operations.

Mapping problems of one data type into Boolean variables is complex, and this complexity increases when the software performs such mapping. The software handles models with predominantly logical signals more efficiently than it does those with large integer or floating-point signals.

Note Simulink Design Verifier software can handle floating-point inputs when their values impact the design through linear inequalities such as $x < y$ or $a > 0$.

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

Handling Models with Large State Spaces

Persistent design variables (variables that are assigned in one time step and used in a later time step during simulation) affect the complexity of analysis in much the same way as input complexity. You can use one or more of the following techniques to simplify the complexity of the state space you want to search:

- Apply constraints to input signals that are delayed.
- Constraint the inputs to states that are contained within conditionally executed subsystems.
- Limit the number of test case steps by setting the **Maximum test case step** parameter to 20.
- Increase the sample time for part or all of the model. (This procedure is similar to reducing timer thresholds, as described in “Handling Problems with Counters and Timers” on page 10-18.) A test case you generate at a lower sample rate often has similarities to the test case with a high sample rate that you need to achieve an objective.

States that are computed from previous state values present a special challenge. For example, if you want to restrict the integrator value in a PID controller, you can only use a set of values that includes all reachable values from the initial value. Otherwise, the input must be forced to 0. Neither of these limitations is practical and would probably make test generation or property proving less complete.

Alternatively, you can use any existing simulation data to help satisfy your testing needs. If you have existing test data, run it on your model and collect model coverage. By using the `sldvgen cov` function, you can ignore model coverage objectives that have already been satisfied in simulation when you supply a coverage data object.

Note For more information on satisfying missing model coverage, see the [Extending an Existing Test Suite](#) demo.

Handling Problems with Counters and Timers

Complexity from states occurs from both the size of the state representation and the number of time steps required to transition from one state to another. The Simulink Design Verifier software searches through sequences of time steps, starting from the default configuration, to find input values that reach a state that satisfies an objective.

Note For the purposes of Simulink Design Verifier analysis, the term *configuration* refers to a set of values for all the persistent information in your model.

The search process investigates all configurations that can be reached in a single time step before considering any of the configurations that can be reached in two time steps. Likewise, the search investigates all configurations that can be reached in two time steps before it considers any configuration that requires three or more time steps, etc.

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

You may see similar effects 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 causes the test sequences to contain more time steps, resulting in longer test cases that are more difficult to identify.

Some basic techniques you can use to improve performance in models that have delays include:

- 1 Make time delays tunable parameters. Choose very small values when running a Simulink Design Verifier analysis. A system with a logical error when a time delay is set to 2000 steps usually demonstrates that error if the time delay is changed to 2 steps. If your system has several delays, choose small but unique values for each of them so that your delays are progressively satisfied.

- 2** Choose higher-frequency cutoffs for filters and fewer samples to average to minimize filtering delays.
- 3** Make the initial values of counters and timers parameter values that the Simulink Design Verifier software can modify. The software finds initial values that allow shorter test cases to exceed thresholds.

Techniques for Proving Properties of Large Models

Property proving uses the same underlying techniques as test generation and suffers from the same performance limitations. However, unlike test generation, you often cannot simplify the problem without compromising the validity of the results.

You can quickly prove simple proof objectives that are not affected by model dynamics. However, a successful proof requires that the Simulink Design Verifier software search through all reachable configurations of your model—even the ones that are reached only after long time delays. The computation time and memory required to search a model completely often make an exhaustive proof impractical.

Simulink Design Verifier software offers a bounded model-checking capability to examine properties in larger, more complicated models. Bounded model checking restricts the search for property violations to a predefined limit of time steps. If a violation is not detected, it is impossible to violate the property with any input sequence having fewer time steps than the specified limit. However, you cannot prove that the property is true because there might be a counterexample within more time steps than the specified limit.

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

Note For more information about the parameters for property proving, see “Property Proving Pane” on page 6-12.

Use the following technique for proving properties in large model combines proving and searching for violations:

- 1 On the **Design Verifier > Property Proving** pane, set the **Strategy** parameter to **Prove**.

- 2** On the **Design Verifier** pane, use a relatively short value for the **Maximum analysis time** parameter, such as 5–10 minutes. If there are trivial counterexamples—or if your properties do not depend on model dynamics—the analysis should complete in that amount of time.
- 3** Change the **Strategy** parameter to **Find violation**, and choose a small bound for the **Maximum violation steps** parameter, such as 4, 5, or 6. If your properties have simple counterexamples, the software should discover them.
- 4** If you do not find any violations with a small bound, increase the bound and look for longer counterexamples.
 - a** 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.
 - b** In addition, consider the dynamics of your model and the number of time steps needed to transition between an arbitrary pair of configurations. If you choose too large a bound, the violation search can be more complex than the unbounded proof.
- 5** If you can run violation searches with relatively large bounds, e.g., 30–50 time steps, switch back to the **Prove** strategy, and use a longer time limit, such as several hours.

Function Reference

sldv.assume

Purpose Proof assumption

Note `dv.assume`, which is similar to `sldv.assume`, will be removed in a future release. Use `sldv.assume` instead.

Syntax `sldv.assume(expr)`

Description `sldv.assume(expr)` specifies that *expr* be true for every evaluation while proving properties. Use any valid Boolean Embedded MATLAB expression for *expr*.

This function has no output and no impact on its parenting function, other than any indirect side effects of evaluating *expr*. If you issue this function from the MATLAB command line, the function has no effect.

Intersperse `sldv.assume` proof assumptions within Embedded MATLAB code or separate the assumptions into a verification script.

The **Proof assumptions** option in the **Property proving** pane applies to proof assumptions represented with the `sldv.assume` function, as well as with the Proof Assumption block.

Examples Add property proof objective and proof assumption, using Embedded MATLAB expressions.

- 1 Open the `sldvdemo_sbr_verification` model and save it as `my_sldvdemo_sbr_verification`.

my_sldvdemo_sbr_verification *

File Edit View Simulation Format Tools Help

10.0 Normal

Simulink Design Verifier Property Proving Workflow for Seat Belt Reminder

Part 1: Finding property violations

The diagram illustrates the workflow for finding property violations. It consists of three main blocks:

- Design Model:** A block named 'sldvdemo_sbr_design' with 'Inputs' on the left and 'SeatBeltIcon' on the right. A '1' in a circle is next to the 'Inputs' port.
- Safety Properties:** A block with 'Inputs' on the left and 'SeatBeltIcon' on the right. It contains a checkmark icon and a cross icon.
- Input Constraints:** A block with 'Inputs' on the left. It also contains a checkmark icon and a cross icon.

 Arrows show the flow of data from the Design Model to both the Safety Properties and Input Constraints blocks.

This model verifies the seat belt reminder design model referenced in the top block above. The Safety Properties block below it contains a property specified in MATLAB that indicates when the icon should be active. Simulink Design Verifier analyzes the design model and safety property to prove correctness or to identify counterexamples.

In this model, the property is violated because the design implicitly assumes that the KEY input starts at 0 and changes by increments of 1..

Run (double-click) View Options (double-click) Open Fixed Model (double-click)

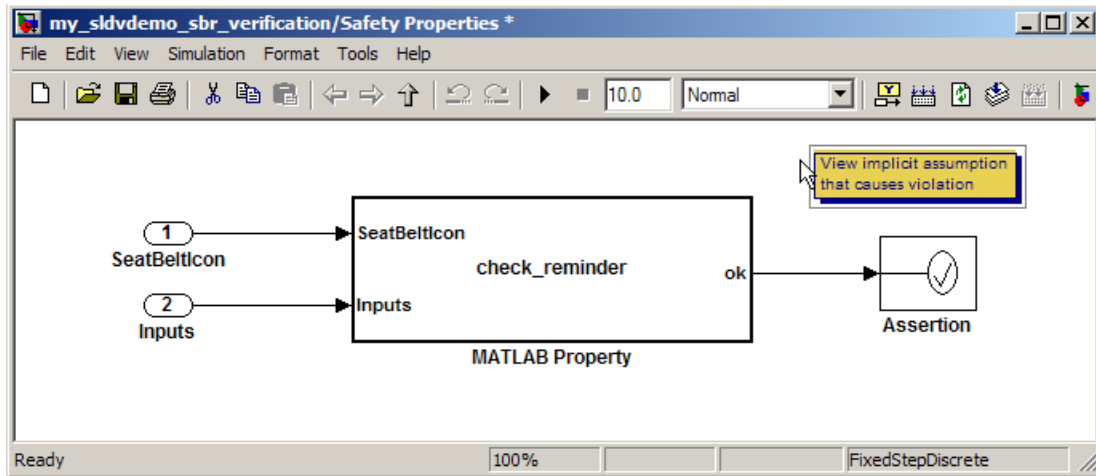
Run Simulink Design Verifier View Simulink Design Verifier Options Open Fixed Model

Copyright 2006-2008 The MathWorks, Inc.

Ready 100% FixedStepDiscrete

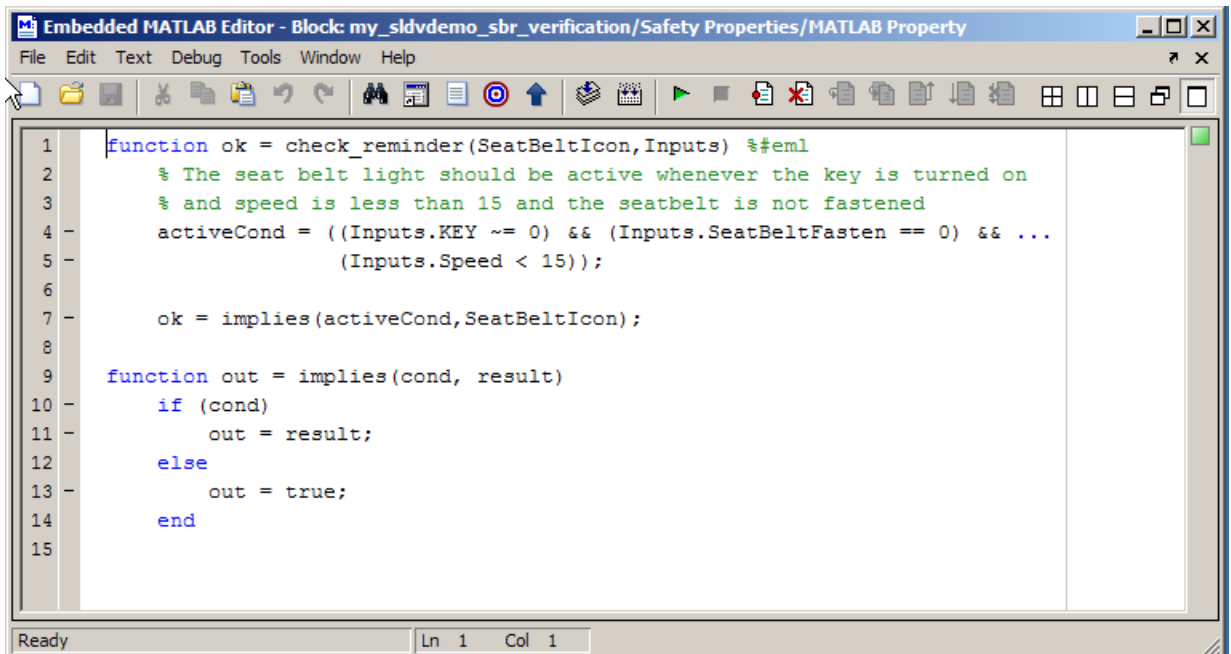
sldv.assume

2 Open the Safety Properties subsystem.



3 Remove the Assertion block. Instead of the Assertion block, this example uses `sldv.prove` and `sldv.assume`.

4 Open the MATLAB Property Embedded MATLAB Function block.



```

1  function ok = check_reminder(SeatBeltIcon,Inputs) %#eml
2      % The seat belt light should be active whenever the key is turned on
3      % and speed is less than 15 and the seatbelt is not fastened
4      activeCond = ((Inputs.KEY ~= 0) && (Inputs.SeatBeltFasten == 0) && ...
5                  (Inputs.Speed < 15));
6
7      ok = implies(activeCond,SeatBeltIcon);
8
9      function out = implies(cond, result)
10     if (cond)
11         out = result;
12     else
13         out = true;
14     end
15

```

5 Delete `ok = implies(activeCond,SeatBeltIcon)`.

6 To add a proof objective and proof assumption, add these two lines to the `check_reminder` code.

```

sldv.prove(implies(activeCond,SeatBeltIcon)
sldv.assume(Inputs.KEY==0 | 1)

```

7 In the Embedded MATLAB editor, save the updated code.

8 Prove the safety properties: from the Safety Properties subsystem, select **Tools > Design Verifier > Prove Properties**.

Alternatives

Instead of using the `sldv.assume` function, you can insert a Proof Assumption block in your model. However, using `sldv.assume` instead

of a Proof Assumption block offers several benefits, described in “About Property Proving” on page 8-2.

You can also constrain signal values when proving models by using Embedded MATLAB without using the `sldv.assume` function. However, using `sldv.assume` instead of directly using Embedded MATLAB eliminates the need to:

- Express the assumption with a Simulink block
- Explicitly connect the assumption output to a Simulink block

See Also

`sldv.condition` | `sldv.prove` | `sldv.test` | Proof Assumption | Proof Objective | Test Condition | Test Objective

Tutorials

- “Proving Properties in a Model” on page 8-5

How To

- “Workflow for Proving Model Properties” on page 8-4

Purpose

Replace model blocks for analysis

Syntax

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

Description

[*status*, *newmodel*] = sldvblockreplacement(*model*) copies *model* and replaces specified model blocks and other model components for a Simulink Design Verifier analysis. sldvblockreplacement replaces the blocks of the model according to the block-replacement rules in the model configuration settings. sldvblockreplacement returns a handle to the new model in *newmodel*. If the operation is successful, sldvblockreplacement returns a *status* of 1. Otherwise, it returns 0.

[*status*, *newmodel*] = sldvblockreplacement(*model*, *options*) replaces the blocks of *model* according to the block replacement rules specified in the sldvoptions object *options*, and returns a handle to the new model in *newmodel*.

[*status*, *newmodel*] = sldvblockreplacement(*model*, *options*, *showUI*) performs the same tasks as sldvblockreplacement(*model*, *options*). If *showUI* is true, errors appear in the Simulation Diagnostics Viewer. Otherwise, errors appear at the MATLAB command line.

Examples

Replace the blocks in sldvdemo_blockreplacement_unsupportedblocks using the block-replacement rules specified in opts:

```
opts = sldvoptions;  
opts.BlockReplacement = 'on'  
opts.BlockReplacementRulesList = ...  
'<FactoryDefaultRules>, custom_rule_switch';  
[status, newmodel] = sldvblockreplacement(...  
    'sldvdemo_blockreplacement_unsupportedblocks', opts);
```

See Also

sldvoptions

sldvblockreplacement

Tutorials

- “Example: Replacing Multiport Switch Blocks” on page 4-8

How To

- Chapter 4, “Working with Block Replacements”

Purpose

Check model for compatibility with analysis

Syntax

```
status = sldvcompat(model)
status = sldvcompat(block)
status = sldvcompat(subsystem, options)
status = sldvcompat(model, options, showUI, startCov)
```

Description

`status = sldvcompat(model)` returns a `status` of 1 if `model` is compatible with Simulink Design Verifier software. Otherwise, `sldvcompat` returns 0.

`status = sldvcompat(block)` converts the Simulink `block` into a temporary model and checks the compatibility of that model with Simulink Design Verifier software. After the compatibility check, `sldvcompat` closes the temporary model.

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

`status = sldvcompat(model, options, showUI, startCov)` checks the compatibility of the model with Simulink Design Verifier software. If `showUI` is true, errors appear in the Simulation Diagnostics Viewer. Otherwise, errors appear at the MATLAB command line. The analysis ignores all model coverage objectives satisfied in `startCov`, a `cvdata` object.

Examples

Check the `sldvdemo_flipflop` model to see if it is compatible with Simulink Design Verifier software:

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

Alternatives

To check the compatibility of a model, in the Model Editor window, select **Tools > Design Verifier > Check Model Compatibility** to check if a model is compatible with the Simulink Design Verifier software.

To check the compatibility of a subsystem, right-click the subsystem and select **Check Subsystem Compatibility**.

sldvcompat

See Also

sldvoptions | sldvrun

How To

- “Checking Compatibility of the Example Model” on page 7-7

Purpose Test condition

Note `dv.condition`, which is similar to `sldv.condition`, will be removed in a future release. Use `sldv.condition` instead.

Syntax `sldv.condition(expr)`

Description

`sldv.condition(expr)` Specifies that *expr* is true for every time step in a generated text case. Use any valid Boolean Embedded MATLAB expression for *expr*.

This function has no output and no impact on its parenting function, other than any indirect side effects of evaluating *expr*. If you issue this function from the MATLAB command line, the function has no effect.

Intersperse `sldv.condition` test conditions within Embedded MATLAB code or separate the conditions into a verification script.

The **Test conditions** option in the **Test generation** pane applies to test conditions represented with the `sldv.condition` function, as well as with the Test Condition block.

Examples

Add a test objective and test conditions, using Embedded MATLAB expressions.

- 1 Open the `sldvdemo_cruise_control` model and save it as `my_sldvdemo_cruise_control`.

The screenshot shows the Simulink Design Verifier window titled "my_sldvdemo_cruise_control". The main area displays a block diagram for "Cruise Control Test Generation". The diagram features a central "Controller" block with several inputs and outputs. On the left, there are six numbered input ports: 1 (enable), 2 (brake), 3 (set), 6 (speed), 4 (inc), and 5 (dec). The "speed" input (6) is connected to a circular constraint icon labeled "[0 100]". The Controller block has two internal labels: "throt" and "target". On the right, there are two numbered output ports: 1 (throt) and 2 (target). Below the diagram is a text box explaining the test generation configuration: "This model is configured to generate test cases that achieve complete model coverage. By default Simulink Design Verifier generates test cases that satisfy objectives in the fewest steps. One of the test objectives forces the discrete integrator in the PI controller to exceed its upper limit. When you run Simulink Design Verifier without constraints the limit is exceeded in a single step by forcing speed to be 500. The constraint on speed limits the values in test cases between 0 and 100. This forces the test cases to take several samples to exceed the integrator limit." At the bottom of the window, there are three blue buttons: "Run (double-click)", "Toggle Speed Constraint (double-click)", and "View Options (double-click)". Below these buttons are the labels "Run Simulink Design Verifier", "Toggle Constraint", and "View Simulink Design Verifier Options". The status bar at the bottom shows "Ready", "100%", and "FixedStepDiscrete".

Simulink Design Verifier
Cruise Control Test Generation

1 enable
2 brake
3 set
6 speed
4 inc
5 dec

[0 100]

Controller

1 throt
2 target

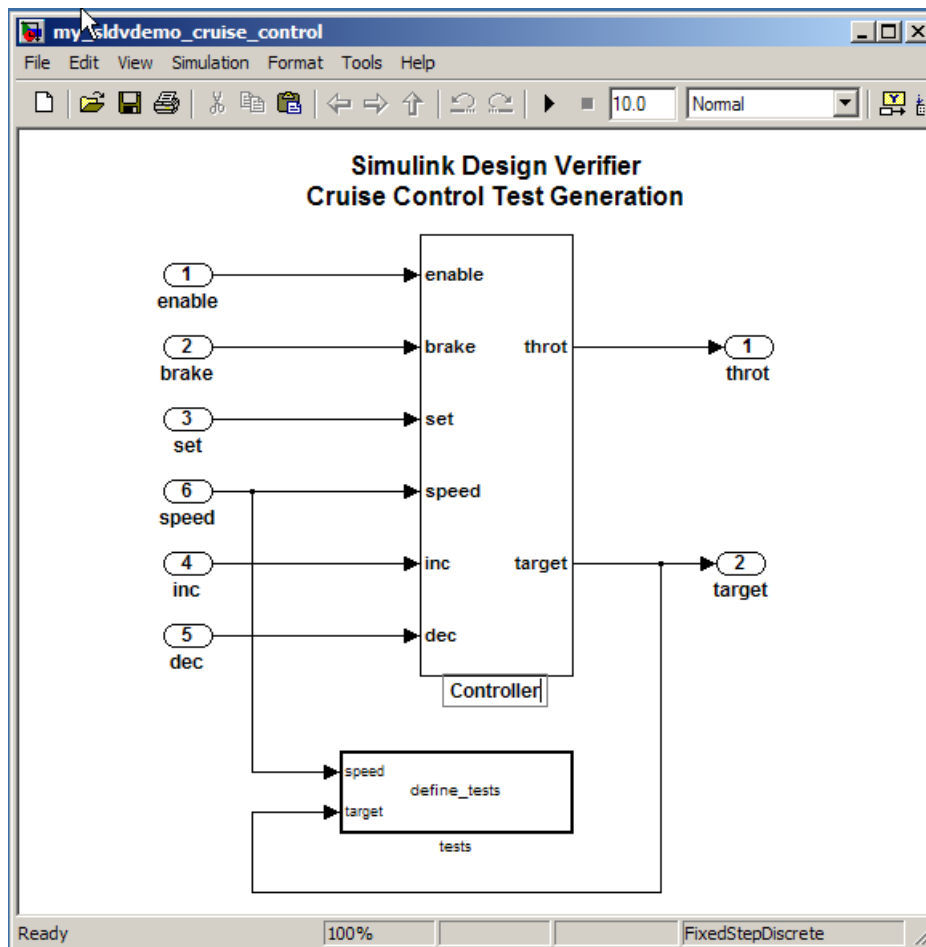
This model is configured to generate test cases that achieve complete model coverage. By default Simulink Design Verifier generates test cases that satisfy objectives in the fewest steps. One of the test objectives forces the discrete integrator in the PI controller to exceed its upper limit. When you run Simulink Design Verifier without constraints the limit is exceeded in a single step by forcing speed to be 500. The constraint on speed limits the values in test cases between 0 and 100. This forces the test cases to take several samples to exceed the integrator limit.

Run (double-click)
Toggle Speed Constraint (double-click)
View Options (double-click)

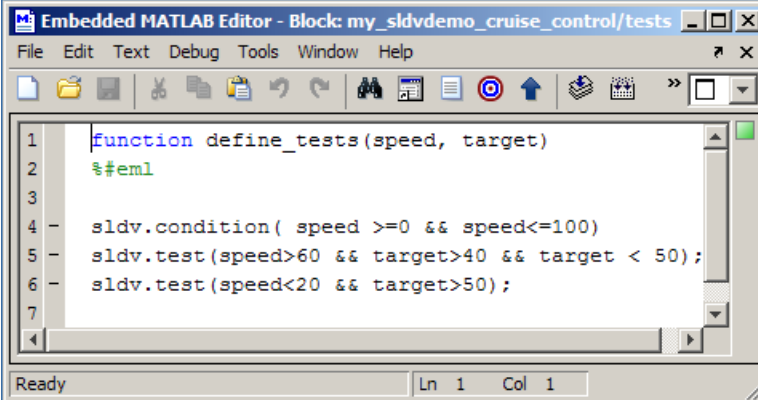
Run Simulink Design Verifier Toggle Constraint View Simulink Design Verifier Options

Ready 100% FixedStepDiscrete

- 2** Remove the Test Condition block for the `speed` block signal. Instead of the Test Condition block, this example uses `sldv.test` and `sldv.condition`.
- 3** From the User-Defined Functions library, add an Embedded MATLAB Function block and:
 - a** Name the block “tests.”
 - b** Connect the block to the signal for the `speed` block and to the signal for the `target` block.



4 Add the following code to the tests Embedded MATLAB Function block.



```

1 function define_tests(speed, target)
2 %#eml
3
4 - sldv.condition( speed >=0 && speed<=100)
5 - sldv.test(speed>60 && target>40 && target < 50);
6 - sldv.test(speed<20 && target>50);
7

```

5 In the Embedded MATLAB editor, save the updated code.

6 Generate the test: select **Tools > Design Verifier > Generate Tests**.

Alternatives

Instead of using the `sldv.condition` function, you can insert a Test Condition block in your model. However, using `sldv.condition` instead of a Test Condition block offers several benefits, described in “About Test Case Generation” on page 7-2.

You can also specify test conditions by using Embedded MATLAB without using the `sldv.condition` function. However, using `sldv.condition` instead of directly using Embedded MATLAB eliminates the need to:

- Express the constraints with Simulink blocks
- Explicitly connect the condition output to a Simulink block

See Also

`sldv.assume` | `sldv.prove` | `sldv.test` | Proof Assumption | Proof Objective | Test Condition | Test Objective

Tutorials

- “Generating Test Cases for a Model” on page 7-5

How To

- “Workflow for Generating Test Cases” on page 7-4

Purpose

Extract subsystem contents into new model for analysis

Syntax

```
[status, newmodel] = sldvextract(block)
[status, newmodel] = sldvextract(block, showModel)
[status, newmodel] = sldvextract(block, showModel, showUI)
[status, newmodel] = sldvextract(block, showModel, showUI,
    isvalid)
```

Description

[*status*, *newmodel*] = sldvextract(*block*) extracts the contents of the Atomic Subsystem block *block* and creates a model for the Simulink Design Verifier software to analyze. The sldvextract function returns the handle of the new model in *newmodel*. sldvextract uses the subsystem name for the model name, appending a numeral to the model name if that model name already exists. If the operation is successful, sldvextract returns a *status* of 1. Otherwise, it returns 0.

[*status*, *newmodel*] = sldvextract(*block*, *showModel*) opens the extracted model if you set *showModel* to true.

[*status*, *newmodel*] = sldvextract(*block*, *showModel*, *showUI*) performs the same tasks as sldvextract(*block*, *showModel*). If you set *showUI* to true, the output in the Simulation Diagnostics Viewer; otherwise, output appears in the MATLAB command window.

[*status*, *newmodel*] = sldvextract(*block*, *showModel*, *showUI*, *isvalid*) performs the same tasks as sldvextract(*block*, *showModel*, *showUI*). The *isvalid* argument is for internal use.

Examples

Extract the Atomic Subsystem block, Bus Counter, from the sldemo_mdhref_conversion model and copy it into a new model:

```
open_system('sldemo_mdhref_conversion');
[status, newmodel] = sldvextract('sldemo_mdhref_conversion/Bus Counter',...
    true);
```

sldvgencov

Purpose

Analyze models to obtain missing model coverage

Syntax

```
[status, cvdo] = sldvgencov(model, options, showUI, startCov)
[status, cvdo] = sldvgencov(block, options, showUI, startCov)
[status, cvdo, filenames] = sldvgencov(model, options, showUI, startCov)
[status, cvdo, filenames, newmodel] = sldvgencov(block, options, showUI, startCov)
```

Description

[*status*, *cvdo*] = `sldvgencov(model, options, showUI, startCov)` analyzes *model* using the `sldvoptions` object *options*.

[*status*, *cvdo*] = `sldvgencov(block, options, showUI, startCov)` analyzes the Atomic Subsystem block *block* using the `sldvoptions` object *options*.

[*status*, *cvdo*, *filenames*] = `sldvgencov(model, options, showUI, startCov)` analyzes *model* and returns the file names that the software created in *filenames*.

[*status*, *cvdo*, *filenames*, *newmodel*] = `sldvgencov(block, options, showUI, startCov)` analyzes *block* using the `sldvoptions` object *options*. The software returns a handle to *newmodel*, which contains a copy of the *block* subsystem.

Inputs

block

Handle to an Atomic Subsystem block in a Simulink model

model

Handle to a Simulink model

Default: []

options

`sldvoptions` object that specifies analysis parameters

Default: []

showUI

Logical value indicating where to display messages during analysis

 true to display messages in the log window

 false (default) to display messages in the MATLAB command window

startCov

cvdata object. The analysis ignores all model coverage objectives already satisfied in *startCov*.

Default: []

Outputs

cvdo

cvdata object containing coverage data for new tests

filenames

A structure whose fields list the file names with the analysis results:

DataFile	MAT-file with raw input data
HarnessModel	Simulink harness model
SystemTestFile	SystemTest TEST-file
Report	HTML report of the results
ExtractedModel	Simulink model extracted from subsystem
BlockReplacementModel	Simulink model obtained after block replacements

status

Logical value that indicates whether the analysis collected model coverage successfully

```
true  
false
```

Examples

Analyze the Cruise Control model and simulate a version of that model using data from test cases from the previous analysis. Compare the model coverage data, and collect the coverage missing from the `sldvdemo_cruise_control_mod` model analysis:

```
opts = sldvoptions;  
opts.Mode = 'TestGeneration';           % generate test cases  
opts.ModelCoverageObjectives = 'MCDC';  % MCDC coverage  
opts.SaveHarnessModel = 'off';          % Don't create the harness model  
opts.SaveReport = 'off';                % or report  
open_system('sldvdemo_cruise_control');  
[ status, files ] = sldvrun('sldvdemo_cruise_control', opts);  
open_system('sldvdemo_cruise_control_mod.mdl');  
[ outData, startCov ] = sldvrntest('sldvdemo_cruise_control_mod',...  
    files.DataFile, [], true);  
cvhtml('Coverage with the original test suite', startCov);  
[ status, covData, files ] = sldvgencov('sldvdemo_cruise_control_mod',...  
    opts, false, startCov);
```

See Also

`sldvharnessmerge` | `sldvoptions` | `sldvrun`

Tutorials

- “Generating Test Cases for a Model” on page 7-5

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 in *models*. sldvharnessmerge saves the data and initialization commands in *name*, which is a handle to a model.

initialization_commands is a cell array of strings the same length as *models*. It defines parameter settings for the test cases of each test harness model.

If the operation is successful, sldvharnessmerge returns a *status* of 1. Otherwise, it returns 0.

sldvharnessmerge requires that *name* and the models in *models* have only one Signal Builder block at the top level.

Use sldvharnessmerge with sldvgencov to combine test cases that use different sets of parameter values.

Examples

Analyze sldvdemo_cruise_control for decision and full coverage and merge the two test harnesses:

```
model = 'sldvdemo_cruise_control';  
open_system(model)  
% Collect decision coverage  
harness1 = 'first_harness';  
opts1 = sldvoptions;  
opts1.Mode = 'TestGeneration';  
opts1.ModelCoverageObjectives = 'Decision';  
opts1.HarnessModelFileName = harness1;  
sldvrun(model, opts1);  
% Collect full coverage  
harness2 = 'second_harness';  
opts2.Mode = 'TestGeneration';  
opts2 = sldvoptions;
```

sldvharnessmerge

```
opts2.ModelCoverageObjectives = 'ConditionDecision';
opts2.HarnessModelFileName = harness2;
sldvrun(model, opts2);
% Merge the two harness files:
status = sldvharnessmerge(harness1, harness2);
```

See Also

sldvgencov | sldvmakeharness | sldvrun

Purpose Check if Simulink Design Verifier software is analyzing model

Syntax

```
status = sldvisactive
status = sldvisactive(model)
status = sldvisactive(block)
```

Description `status = sldvisactive` checks if the Simulink Design Verifier software is actively analyzing the current Simulink model. If the operation is successful, `sldvisactive` returns 1. Otherwise, it returns 0.

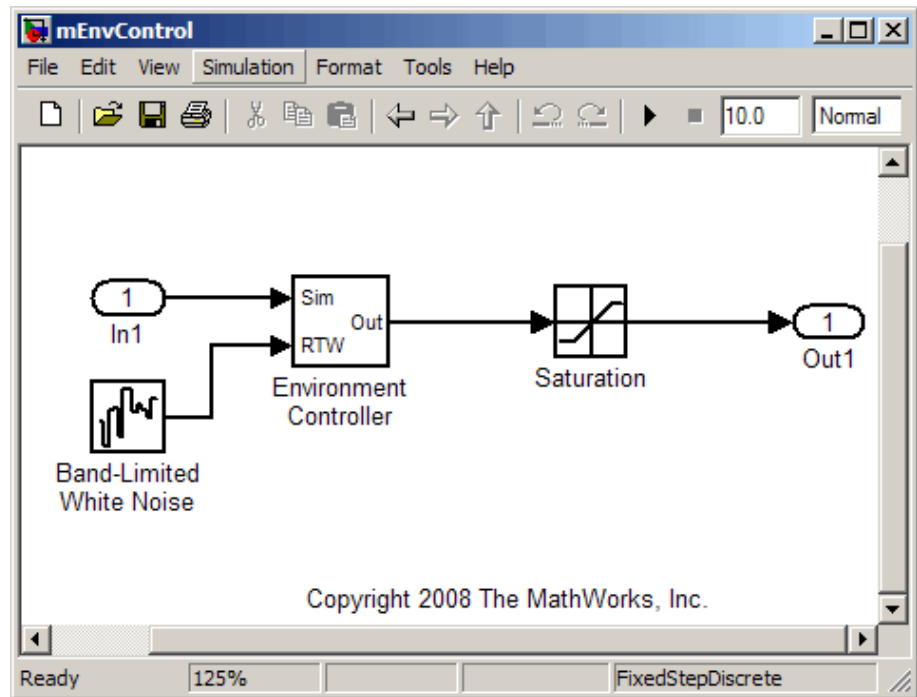
`status = sldvisactive(model)` checks if the Simulink Design Verifier software is actively analyzing `model`.

`status = sldvisactive(block)` checks if the Simulink Design Verifier software is actively analyzing the model that contains `block`.

`sldvisactive` customizes the model analysis in block and model callback functions, or mask initialization.

Examples Eliminate blocks that are incompatible with the Simulink Design Verifier software:

- 1 Create a Simulink model and save it as `mEnvControl`.



- 2 Right-click the Environment Controller block and select **View Mask**.
- 3 Click the **Initialization** tab and add the following command, if it does not exist:

```
switch_mode = rtwenvironmentmode(bdroot(gcbh)) || ...  
    (exist('sldvisactive','file')~=0 && ...  
    sldvisactive(bdroot(gcbh)));
```

The Simulink Design Verifier software does not support Band-Limited White Noise blocks. If the software is analyzing the mEnvControl model the mask initialization of the Environment Controller block:

- Sets the pass-through mode to pass the Sim signal to the output port.
- Eliminates the RTW port, which is incompatible with the Simulink Design Verifier software.

sldvmakeharness

Purpose

Generate test harness model

Syntax

```
[status, savedHarnessFilePath] = sldvmakeharness
[status, savedHarnessFilePath] = sldvmakeharness(model)
[status, savedHarnessFilePath] = sldvmakeharness(model,
    sldvDataFile)
[status, savedHarnessFilePath] = sldvmakeharness(model, [],
    harnessFilePath)
[status, savedHarnessFilePath] = sldvmakeharness(model, [],
    [], modelRefHarness)
```

Description

[*status*, *savedHarnessFilePath*] = `sldvmakeharness` generates a test harness from the current Simulink model. If `sldvmakeharness` generates the test harness successfully, it returns *status* = 1. It also returns the path and file name for the generated harness model in *savedHarnessFilePath*. `sldvmakeharness` creates a harness model with no data; the harness includes one default test case that specifies default values for all input signals.

[*status*, *savedHarnessFilePath*] = `sldvmakeharness(model)` generates a test harness from *model*, which is a handle to a Simulink model or a string with the model name.

[*status*, *savedHarnessFilePath*] = `sldvmakeharness(model, sldvDataFile)` generates a test harness from the data file *sldvDataFile*. If the model name and data in *sldvDataFile* do not match the name and data of *model*, `sldvmakeharness` displays a warning, but it generates the harness and returns *status* = 1.

[*status*, *savedHarnessFilePath*] = `sldvmakeharness(model, [], harnessFilePath)` generates a test harness from *model* and stores it in *harnessFilePath*. If *harnessFilePath* is invalid, `sldvmakeharness` cannot save the harness, but it creates and opens it.

[*status*, *savedHarnessFilePath*] = `sldvmakeharness(model, [], [], modelRefHarness)` generates a test harness from *model*. If *modelRefHarness* is true, the test harness model includes *model* in a Model block. If *modelRefHarness* is false, the test harness model includes a copy of *model*.

Successful harness generation does not imply that the model is compatible with the Simulink Design Verifier software.

Examples

Create a test harness for the `sldvdemo_cruise_control` model:

```
[status, harnessfile] = sldvmakeharness('sldvdemo_cruise_control',...  
    '', '', 'true');
```

Alternatives

`sldvmakeharness` creates a test harness model without analyzing the model:

- 1** In the Model Editor, select **Simulation > Configuration Parameters**.
- 2** In the **Design Verifier Results** pane, under **Harness model options**, set the options.
- 3** Click **OK**.
- 4** Select **Tools > Design Verifier > Test Generation** to run an analysis.

See Also

`sldvharnessmerge` | `sldvrun`

sldvoptions

Purpose Create design verification options object

Syntax `options = sldvoptions`
`options = sldvoptions(model)`

Description `options = sldvoptions` returns an object `options` that contains default values for the design verification parameters.
`options = sldvoptions(model)` returns the object `options` attached to `model`.

Outputs `options`

The following table describes the parameters that comprise a Simulink Design Verifier options object.

Parameter	Values
Assertions	'EnableAll' 'DisableAll' 'UseLocalSettings' (default)
AutomaticStubbing	'on' 'off' (default)
BlockReplacement	'on' 'off' (default)
BlockReplacementModel- FileName	string '\$modelName\$_replacement' (default)
BlockReplacementRulesList	string '<FactoryDefaultRules>' (default)

DataFileName	string '\$ModelName\$_sldvdata' (default)
DisplayReport	'on' (default) 'off'
DisplayUnsatisfiable-Objectives	'on' (default) 'off'
HarnessModelFileName	string '\$ModelName\$_harness' (default)
MakeOutputFilesUnique	'on' (default) 'off'
MaxProcessTime	double '600' (default)
MaxTestCaseSteps	int32 '500' (default)
MaxViolationSteps	int32 '20' (default)
Mode	'TestGeneration' (default) 'PropertyProving'
ModelCoverageObjectives	'None' 'Decision' 'ConditionDecision' 'MCDC' (default)

sldvoptions

ModelReferenceHarness	'on' 'off' (default)
OutputDir	string 'sldv_output/\$modelName\$' (default)
Parameters	'on' (default) 'off'
ParametersConfigFileName	string 'sldv_params_template.m' (default)
ProofAssumptions	'EnableAll' 'DisableAll' 'UseLocalSettings' (default)
ProvingStrategy	'FindViolation' 'Prove' (default) 'ProveWithViolationDetection'
RandomizeNoEffectData	'on' 'off' (default)
ReportFileName	string '\$modelName\$_report' (default)
ReportIncludeGraphics	'on' 'off' (default)
SaveDataFile	'on' (default) 'off'

SaveExpectedOutput	'on' 'off' (default)
SaveHarnessModel	'on' (default) 'off'
SaveReport	'on' (default) 'off'
SaveSystemTestHarness	'on' 'off' (default)
SystemTestFileName	string '\$ModelName\$_harness'
TestConditions	'EnableAll' 'DisableAll' 'UseLocalSettings' (default)
TestObjectives	'EnableAll' 'DisableAll' 'UseLocalSettings' (default)
TestSuiteOptimization	'CombinedObjectives' (default) 'IndividualObjectives' 'LargeModel' 'LongTestCases'

Examples

Create an options object and set several parameters:

```
opts = sldvoptions;  
opts.AutomaticStubbing = 'on';  
opts.Mode = 'TestGeneration';
```

sldvoptions

```
opts.ModelCoverageObjectives = 'MCDC';  
opts.ReportIncludeGraphics = 'on';  
opts.SaveHarnessModel = 'off';  
opts.SaveReport = 'off';  
opts.TestSuiteOptimization = 'LongTestCases';
```

Get the options object for the `sldvdemo_cruise_control` model:

```
sldvdemo_cruise_control  
optsModel = sldvoptions(bdroot);  
optsCopy = optsModel.deepCopy;  
optsCopy.MaxProcessTime = 120;
```

Alternatives

In the Model Editor window, select **Tools > Design Verifier > Options** to set the Simulink Design Verifier analysis options.

See Also

`sldvblockreplacement` | `sldvcompat` | `sldvgencov` | `sldvruntime`

How To

- “Configuring Simulink® Design Verifier Options” on page 6-5

Purpose Proof objective

Note `dv.prove`, which is similar to `sldv.prove`, will be removed in a future release. Use `sldv.assume` instead.

Syntax `sldv.prove(expr)`

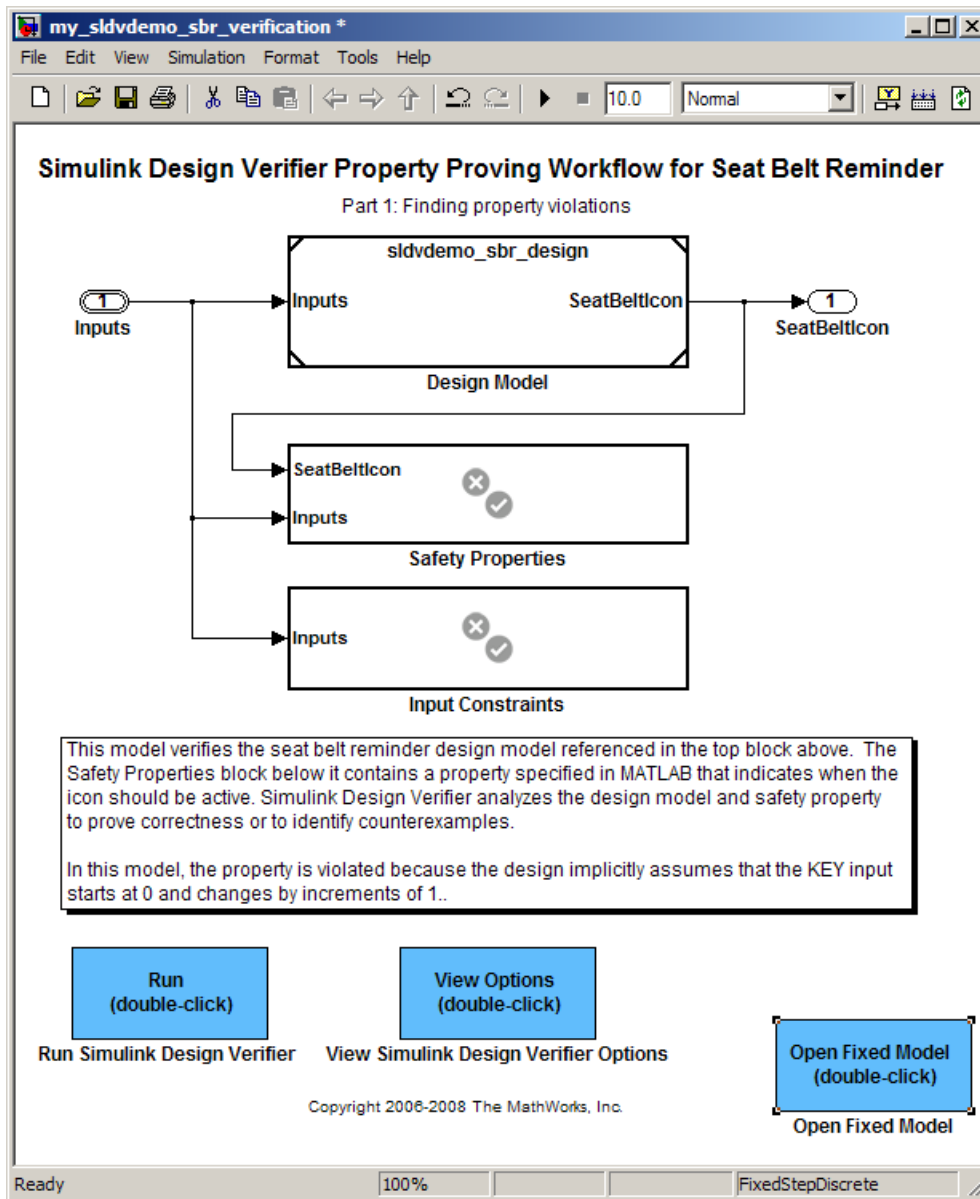
Description `sldv.prove(expr)` specifies that *expr* be true for every evaluation while proving properties. Use any valid Boolean Embedded MATLAB expression for *expr*.

This function has no output and no impact on its parenting function, other than any indirect side effects of evaluating *expr*. If you issue this function from the MATLAB command line, the function has no effect.

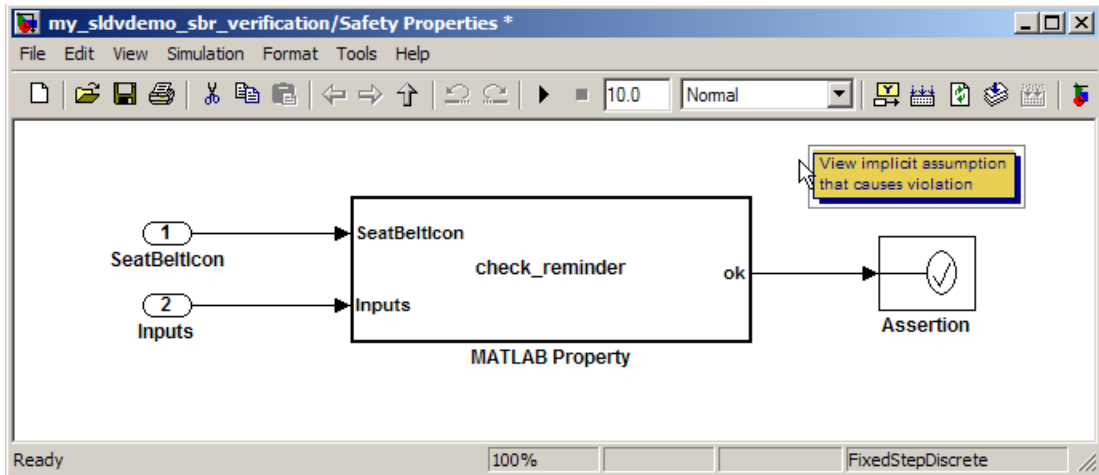
Intersperse `sldv.prove` proof assumptions within Embedded MATLAB code or separate the assumptions into a verification script.

Examples Add property proof objective and proof assumption, using Embedded MATLAB expressions.

- 1 Open the `sldvdemo_sbr_verification` model and save it as `my_sldvdemo_sbr_verification`.

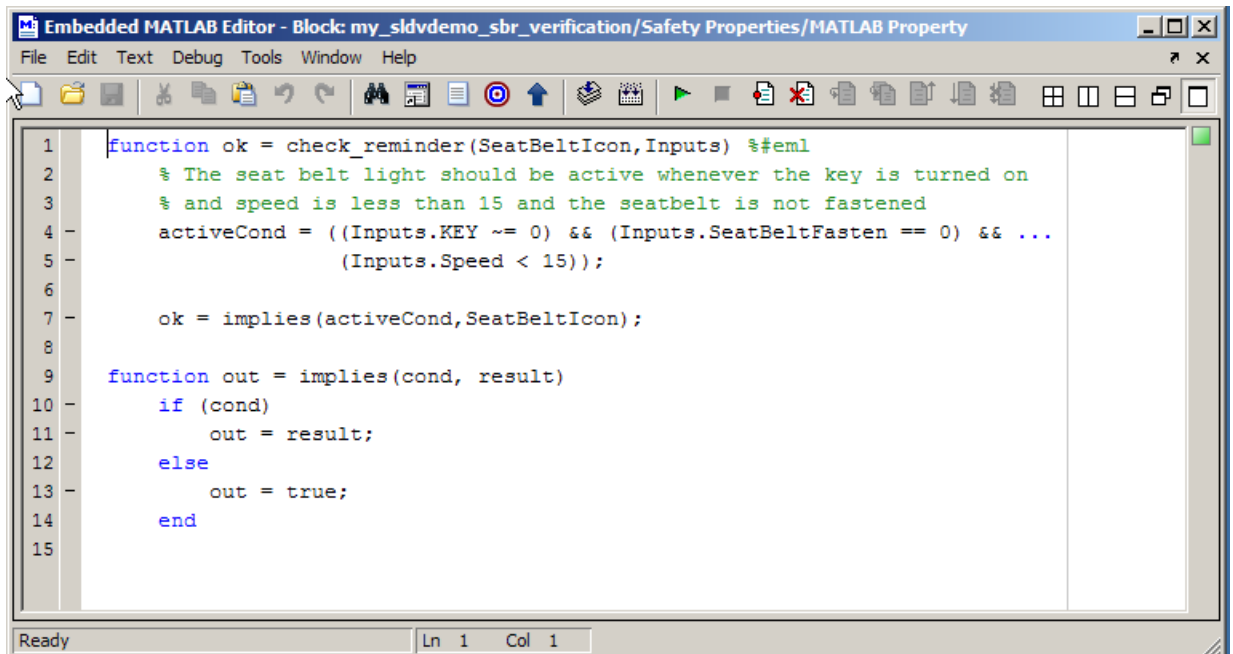


2 Open the Safety Properties subsystem.



3 Remove the Assertion block. Instead of the Assertion block, this example uses `sldv.prove` and `sldv.assume`.

4 Open the MATLAB Property Embedded MATLAB Function block.



```
1 function ok = check_reminder(SeatBeltIcon,Inputs) %#eml
2     % The seat belt light should be active whenever the key is turned on
3     % and speed is less than 15 and the seatbelt is not fastened
4     activeCond = ((Inputs.KEY ~= 0) && (Inputs.SeatBeltFasten == 0) && ...
5                 (Inputs.Speed < 15));
6
7     ok = implies(activeCond,SeatBeltIcon);
8
9 function out = implies(cond, result)
10    if (cond)
11        out = result;
12    else
13        out = true;
14    end
15
```

5 Delete `ok = implies(activeCond,SeatBeltIcon)`.

6 To add a proof objective and proof assumption, add these two lines to the `check_reminder` code.

```
sldv.prove(implies(activeCond,SeatBeltIcon))
sldv.assume(Inputs.KEY==0 | 1)
```

7 In the Embedded MATLAB editor, save the updated code.

8 Prove the safety properties: from the Safety Properties subsystem, select **Tools > Design Verifier > Prove Properties**.

Alternatives

Instead of using the `sldv.prove` function, you can insert a Proof Objective block in your model.

However, using `sldv.prove` instead of a Proof Objective block offers several benefits, described in “About Property Proving” on page 8-2.

You can also specify a proof objective by using Embedded MATLAB without using the `sldv.prove` function. However, using `sldv.prove` instead of directly using Embedded MATLAB eliminates the need to:

- Express the objective with a Simulink block
- Explicitly connect the proof output to a Simulink block

See Also

`sldv.condition` | `sldv.prove` | `sldv.test` | Proof Assumption | Proof Objective | Test Condition | Test Objective

Tutorials

- “Proving Properties in a Model” on page 8-5

How To

- “Workflow for Proving Model Properties” on page 8-4

sldvreport

Purpose Generate report

Syntax

```
[status, reportFilePath] = sldvreport(sldvDataFile)
[status, reportFilePath] = sldvreport(sldvDataFile,
    {reportOption1, reportOption2, ...})
[status, reportFilePath] = sldvreport(sldvDataFile,
    {reportOption1, reportOption2, ...}, reportFilePath,
    showUI)
```

Description

[*status*, *reportFilePath*] = sldvreport(*sldvDataFile*) generates a complete HTML report from the data in *sldvDataFile*. *status* returns true if sldvreport created the report successfully. *reportFilePath* contains the actual name of the HTML report created.

[*status*, *reportFilePath*] = sldvreport(*sldvDataFile*, {*reportOption1*, *reportOption2*, ...}) generates a report from *sldvDataFile* based on the specified options. *options* is a cell array of strings.

[*status*, *reportFilePath*] = sldvreport(*sldvDataFile*, {*reportOption1*, *reportOption2*, ...}, *reportFilePath*, *showUI*) generates a report and saves it in the location *reportFilePath*.

Inputs *sldvDataFile*

Name of the data file that contains the analysis results

Default: ''

options

Cell array of strings that specify options for the report:

'summary'	Include summary analysis data only
'objectives'	Include test objective data

'object'	Include data about all model objects
'testcases'	Include data about all generated test cases
'properties'	Include data about all properties proven or falsified

Default: {}

reportFilePath

The path and file name for the generated HTML report

Default: ''

showUI

Logical value indicating where to display messages during analysis

true to display messages in the log window

false (default) to display messages in the MATLAB command window

Examples

Analyze the model and create the report using `sldvreport`:

```

opts = sldvoptions; % Create options structure
opts.Mode = 'TestGeneration'; % Do test-gen analysis
opts.SaveReport = 'off'; % Don't save HTML report
open_system 'sldvdemo_cruise_control'; % Open the model
[ status, files ] = sldvrun('sldvdemo_cruise_control', opts); %Analyze model
[ status, reportFilePath] = sldvreport(files.DataFile,...
    {'objectives', 'objects', 'testcases'} ); % Create report

```

Alternatives

The Simulink Design Verifier software can create an HTML report after analyzing a model. In the **Design Verifier** category of the

sldvreport

Configuration Parameters dialog box, in the **Report** pane, select **Generate report of the results**.

See Also

sldvrun

Purpose

Analyze model

Syntax

```
status = sldvrun
status = sldvrun(model)
status = sldvrun(block)
status = sldvrun(model, options)
[status, filenames] = sldvrun(model, options)
[status, filenames] = sldvrun(model, options, showUI,
    startCov)
```

Description

`status = sldvrun` analyzes the current model to generate test cases that provide model coverage or prove the model properties.

`status = sldvrun(model)` analyzes *model* to generate test cases that provide model coverage or prove the model properties

`status = sldvrun(block)` converts *block* into a new model and runs a design verification analysis on the new model.

`status = sldvrun(model, options)` analyzes *model* using the `sldvoptions` object *options*.

`[status, filenames] = sldvrun(model, options)` analyzes *model* and returns the file names the software created during the analysis.

`[status, filenames] = sldvrun(model, options, showUI, startCov)` opens the log window during the analysis if you set *showUI* to true. If you set *showUI* to false (the default), it directs output to the MATLAB command line.

Inputs

model

Handle to a Simulink model

Default: []

block

Handle to a block in a Simulink model

Default: []

options

sldvoptions object specifying the analysis options

Default: []

showUI

Logical value indicating where to display messages during the analysis

true to display messages in the log window

false (default) to display messages in the MATLAB command window

startCov

cvdata object specifying model coverage objects for the software to ignore

Default: []

Outputs

filenames

A structure whose fields list the file names that the Simulink Design Verifier software generates:

DataFile	MAT-file with raw input data
HarnessModel	Simulink harness model
SystemTestFile	SystemTest TEST-file
Report	HTML report with the results
ExtractedModel	Simulink model extracted from subsystem
BlockReplacementModel	Simulink model obtained after block replacements

<i>status</i>	
-1	Analysis exceeded the maximum processing time
0	Error
1	Preprocessing completed normally

Examples

Set `sldvoptions` parameters, open the Cruise Control model, and analyze the model using the specified options:

```

opts = sldvoptions;
opts.Mode = 'TestGeneration';           % Perform test-generation analysis
opts.ModelCoverageObjectives = 'MCDC';  % MCDC coverage
opts.SaveHarnessModel = 'off';          % Don't save the harness as model file
opts.SaveReport = 'on';                 % Save the HTML report
open_system('sldvdemo_cruise_control');
[ status, files ] = sldvrun('sldvdemo_cruise_control', opts);

```

Alternatives

In the Model Editor window, select either **Tools > Design Verifier > Generate Tests** or **Tools > Design Verifier > Prove Properties** to run a Simulink Design Verifier analysis option.

See Also

`sldvcompat` | `sldvoptions` | `sldvgencov`

Tutorials

- “Generating Test Cases for a Model” on page 7-5
- “Proving Properties in a Model” on page 8-5

sldvrntest

Purpose

Simulate model using test case

Syntax

```
data = sldvrntest(model, sldvDataFile, testIdx)
data = sldvrntest(model, sldvDataFile)
[data, cvdo] = sldvrntest(model, sldvDataFile, testIdx,
    withCoverage)
[data, cvdo] = sldvrntest(model, sldvDataFile, [],
    withCoverage)
[data, cvdo] = sldvrntest(model, sldvDataFile,
    testIdx, cvt)
[data, cvdo] = sldvrntest(model, sldvDataFile, [], cvt)
[data, cvdo] = sldvrntest(model, sldvDataFile, testIdx,
    withCoverage, outputFormat)
```

Description

`data = sldvrntest(model, sldvDataFile, testIdx)` simulates *model* using input signals associated with a single test case. *testIdx* specifies the index of the test case in *sldvDataFile*. *data* is a structure whose fields contain the simulation results.

`data = sldvrntest(model, sldvDataFile)` simulates *model* using all test cases in the *sldvDataFile* MAT-file. For each test case, the software uses the stop time associated with that particular test case.

`[data, cvdo] = sldvrntest(model, sldvDataFile, testIdx, withCoverage)` simulates *model* using the test case that *testIdx* indexes in *sldvDataFile*. The Simulink Verification and Validation software collects model coverage information during the simulation, which the function returns in the cvdata object *cvdo*.

`[data, cvdo] = sldvrntest(model, sldvDataFile, [], withCoverage)` simulates *model* using all test cases in *sldvDataFile*. If you omit the *testIdx* argument, `sldvrntest` uses all the test cases in the simulation. The Simulink Verification and Validation software collects model coverage information during the simulation, which the function returns in the cvdata object *cvdo*.

`[data, cvdo] = sldvrntest(model, sldvDataFile, testIdx, cvt)` runs a simulation using the test case that *testIdx* indexes in the

sldvDataFile MAT-file, and records coverage data. The coverage tool uses the settings in the *cvtest* object *cvtest*.

`[data, cvdo] = sldvrntest(model, sldvDataFile, [], cvt)` runs simulations for each test case and records the cumulative coverage. The coverage tool uses the settings in *cvtest*.

`[data, cvdo] = sldvrntest(model, sldvDataFile, testIdx, withCoverage, outputFormat)` stores the output values of the model in *model* in the structure *data*.

Inputs

cvtest

cvtest object

model

Handle to a Simulink model

Default: ''

sldvDataFile

Name of the data file that contains the analysis results

Default: ''

testIdx

Test case index in the data file

Default: []

withCoverage

If true, the Simulink Verification and Validation software collects model coverage information during the simulation in *cvdo*.

Default: 0

outputFormat

Specifies format of output values.

'Timeseries' (default)	sldvrntest stores the output values in the Timeseries format.
'StructureWithTime'	sldvrntest stores the output values in the Structure with time format, if the model output signals do not include bus signals.

Outputs

cvdo

cvdata object that contains the model coverage information collected during the simulation

data

A structure whose fields contain the simulation results.

T	Simulation time vector
X	Simulation state matrix
Y	Simulation output captured in time-series objects. If the Output block specifies a bus object, output captured in time-series array objects.

Examples

Run a simulation, using data from any test case in the test suite, to check for full coverage of the model `sldvdemo_cruise_control`:

```
opts = sldvoptions;  
opts.Mode = 'TestGeneration';  
opts.ModelCoverageObjectives = 'MCDC';  
opts.SaveHarnessModel = 'off';  
opts.SaveReport = 'off';  
open_system 'sldvdemo_cruise_control';
```



```
[ status, files ] = sldvrun('sldvdemo_cruise_control', opts);  
[ outData, initialCov ] = sldvrntest('sldvdemo_cruise_control',...  
    files.DataFile, [], true);  
cvhtml('Initial coverage', initialCov);
```

See Also

[cvsim](#) | [cvtest](#) | [sim](#)

sldv.test

Purpose Test objective

Note `dv.test`, which is similar to `sldv.test`, will be removed in a future release. Use `sldv.test` instead.

Syntax `sldv.test(expr)`

Description `sldv.test(expr)` Specifies that *expr* should be made true when generating tests. Use any valid Boolean Embedded MATLAB expression for *expr*.

This function has no output and no impact on its parenting function, other than any indirect side effects of evaluating *expr*. If you issue this function from the MATLAB command line, the function has no effect.

Intersperse `sldv.test` test objectives within Embedded MATLAB code or separate the objectives into a verification script.

The **Test objectives** option in the **Test generation** pane applies to test objectives represented with the `sldv.test` function, as well as with the Test Objective block.

Examples Add a test objective and test conditions, using Embedded MATLAB expressions.

- 1 Open the `sldvdemo_cruise_control` model and save it as `my_sldvdemo_cruise_control`.

Simulink Design Verifier
Cruise Control Test Generation

1 enable
 2 brake
 3 set
 6 speed [0 100]
 4 inc
 5 dec

throt
 target

1 throt
 2 target

Controller

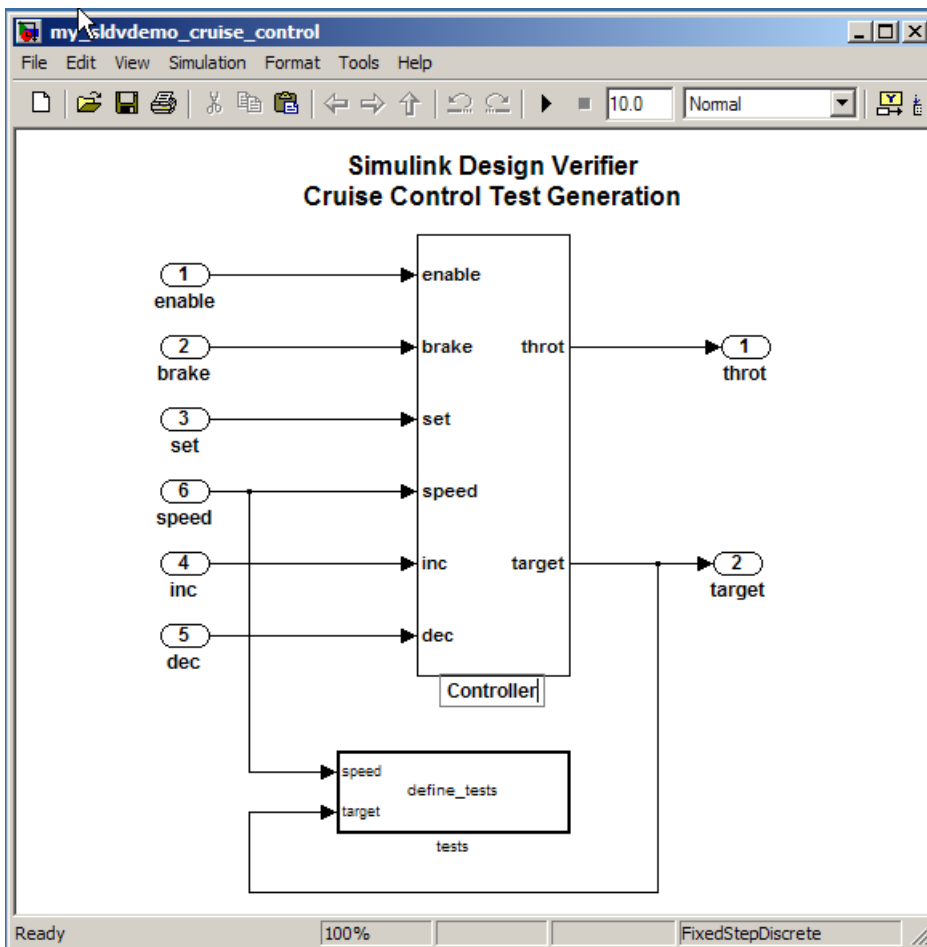
This model is configured to generate test cases that achieve complete model coverage. By default Simulink Design Verifier generates test cases that satisfy objectives in the fewest steps. One of the test objectives forces the discrete integrator in the PI controller to exceed its upper limit. When you run Simulink Design Verifier without constraints the limit is exceeded in a single step by forcing speed to be 500. The constraint on speed limits the values in test cases between 0 and 100. This forces the test cases to take several samples to exceed the integrator limit.

Run (double-click)
 Toggle Speed Constraint (double-click)
 View Options (double-click)

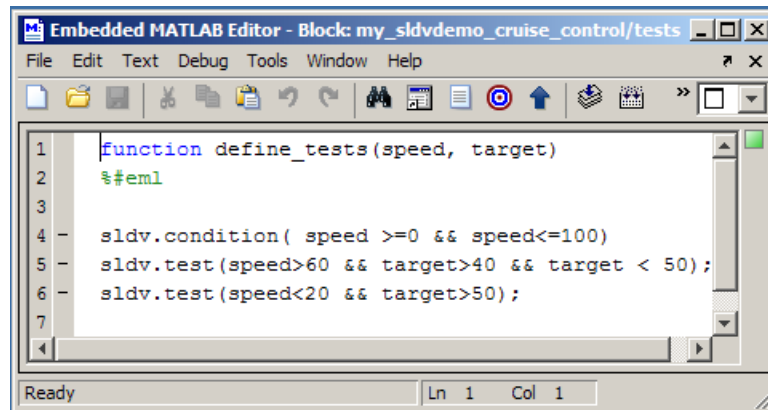
Run Simulink Design Verifier Toggle Constraint View Simulink Design Verifier Options

Ready 100% FixedStepDiscrete

- 2** Remove the Test Condition block for the `speed` block signal. Instead of the Test Condition block, this example uses `sldv.test` and `sldv.condition`.
- 3** From the User-Defined Functions library, add an Embedded MATLAB Function block and:
 - a** Name the block “tests.”
 - b** Connect the block to the signal for the `speed` block and to the signal for the `target` block.



- 4 Add the following code to the tests Embedded MATLAB Function block.



- 5 In the Embedded MATLAB editor, save the updated code.
- 6 Generate the test: select **Tools > Design Verifier > Generate Tests**.

Alternatives

Instead of using the `sldv.test` function, you can insert a Test Objective block in your model.

However, using `sldv.test` instead of a Test Objective block offers several benefits, described in “About Test Case Generation” on page 7-2.

See Also

`sldv.assume` | `sldv.condition` | `sldv.prove` | Proof Assumption | Proof Objective | Test Condition | Test Objective

Tutorials

- “Generating Test Cases for a Model” on page 7-5

How To

- “Workflow for Generating Test Cases” on page 7-4

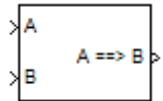
Block Reference

Implies

Purpose Specify conditions that produce certain responses

Library Simulink Design Verifier

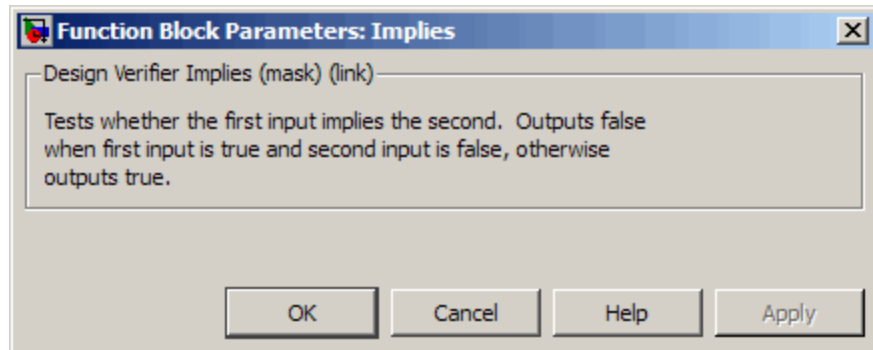
Description



The Implies block lets you specify a condition to produce a given response, for example, when you press the brake pedal on a car, the cruise control mechanism becomes disabled. If input A is true and input B is false, the output is false; for all other pairs of inputs, the output is true.

You can use the Implies block in any model, not just when you run the Simulink Design Verifier software.

Parameters and Dialog Box



Purpose

Constrain signal values when proving model properties

Library

Simulink Design Verifier

Description



When operating in property-proving mode, the Simulink Design Verifier software proves that properties of your model satisfy specified criteria (see Chapter 8, “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. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software proves or disproves that the properties of your model satisfy specified criteria.

The block’s parameter dialog box also allows you to:

- Enable or disable the assumption.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink and Real-Time Workshop software ignore the Proof Assumption block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Proof Assumption block only when proving model properties.

Specifying Proof Assumptions

Use the **Values** parameter to constrain signal values in property proofs. Specify any combination of scalars and intervals in the form of a MATLAB cell array. (For information about cell arrays, see “Cell Arrays” in the MATLAB documentation.)

Proof Assumption

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

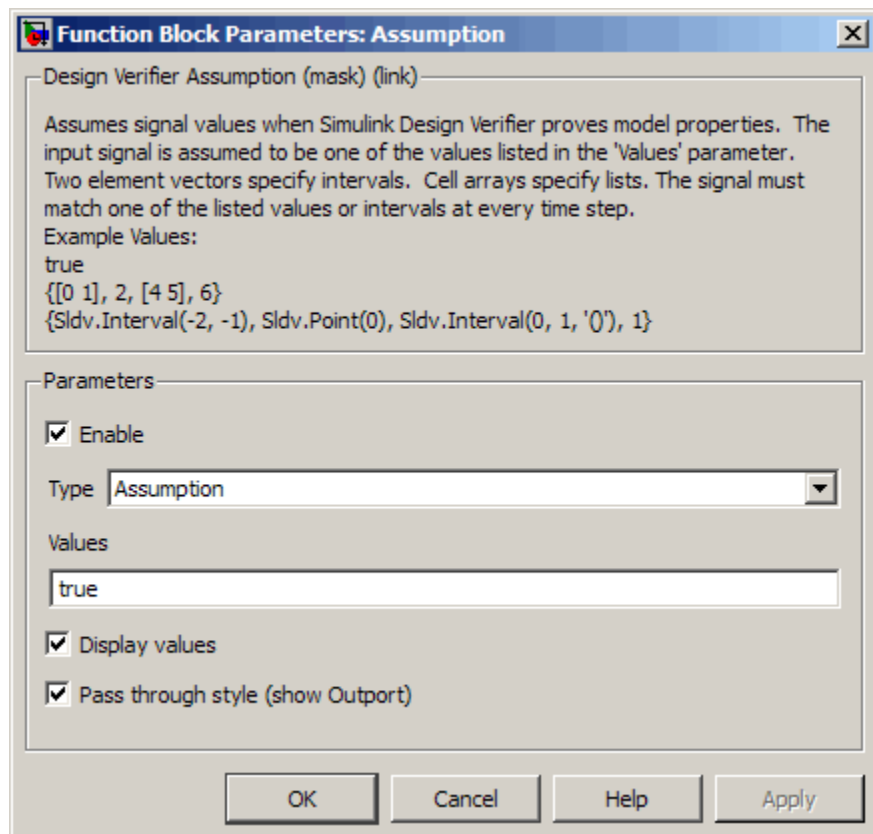
If you specify multiple scalars and intervals for a Proof Assumption block, the Simulink Design Verifier software combines them using a logical OR operation during the property proof. In this case, the software considers the entire assumption to be satisfied if any single scalar or interval is satisfied.

Data Type Support

The Proof Assumption block accepts signals of all built-in data types supported by the Simulink software. For a discussion on the data types supported by the Simulink software, see “Data Types Supported by Simulink” in *Simulink User’s Guide*.

Proof Assumption

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), the Simulink Design Verifier software uses the block when proving properties of a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Proof Assumption block did not exist. If this option is not selected, the block appears grayed out in the model editor.

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

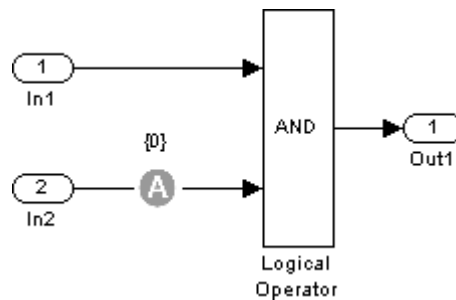
Specify the proof assumption (see “Specifying Proof Assumptions” on page 12-3).

Display values

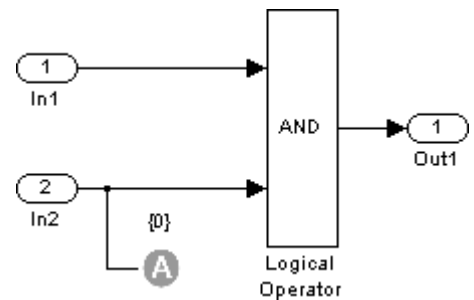
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

Pass through style

Specify whether the block displays an output port in the model editor. If selected (the default), the block displays its output port, allowing its input signal to pass through as the block output. If not selected, the block hides its output port and terminates the input signal. The following figure illustrates the appearance of the block in each case.



Pass through style: selected



Pass through style: deselected

See Also

Proof Objective, Test Condition

Proof Objective

Purpose Define objectives that signals must satisfy when proving model properties

Library Simulink Design Verifier

Description When operating in property-proving mode, the Simulink Design Verifier software proves that properties of your model satisfy specified criteria (see Chapter 8, “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 acceptable values for the block’s input signal. If a signal value deviates from the acceptable values in *any* time step, a property violation occurs and the proof objective is falsified. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software proves or disproves that the properties of your model satisfy specified criteria.

The block’s parameter dialog box allows you to

- Enable or disable the objective.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink and Real-Time Workshop software ignore the Proof Objective block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Proof Objective block only when proving model properties.

Specifying Proof Objectives

Use the **Values** parameter to define values that a signal must achieve during a proof simulation. Specify any combination of scalars and intervals in the form of a MATLAB cell array. (For information about cell arrays, see “Cell Arrays” in the MATLAB documentation.)

Tip If the **Values** parameter specifies only one scalar value, you do not need to enter it in the form of a MATLAB cell array.

Scalar values each comprise a single cell in the array, for example:

```
{0, 5}
```

A closed interval comprises a two-element vector as a cell in the array, where each element specifies an interval endpoint:

```
{[1, 2]}
```

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

- `'()'` — Defines an open interval.
- `'[]'` — Defines a closed interval.
- `'(]'` — Defines a left-open interval.
- `'[])'` — Defines a right-open interval.

Note By default, `Sldv.Interval` considers an interval to be closed if you omit its third input argument.

As an example, the **Values** parameter

```
{0, [1, 3]}
```

specifies:

Proof Objective

- 0 — a scalar
- [1, 3] — a closed interval

The **Values** parameter

```
{Sldv.Interval(0, 1, '['), Sldv.Point(1)}
```

specifies:

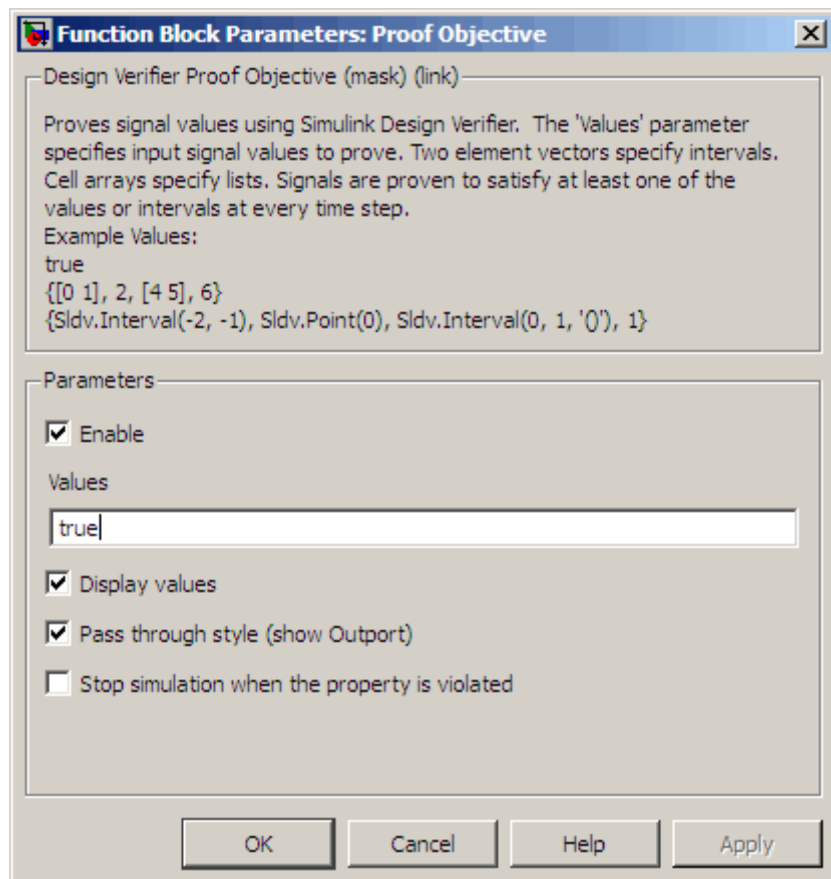
- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

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

Data Type Support

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

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), the Simulink Design Verifier software uses the block when proving properties of a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Proof Objective block did not exist. If this option is not selected, the block appears grayed out in the model editor.

Proof Objective

Values

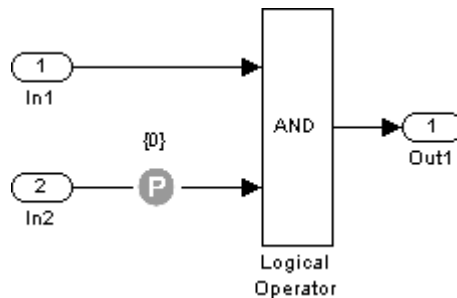
Specify the proof objective (see “Specifying Proof Objectives” on page 12-8).

Display values

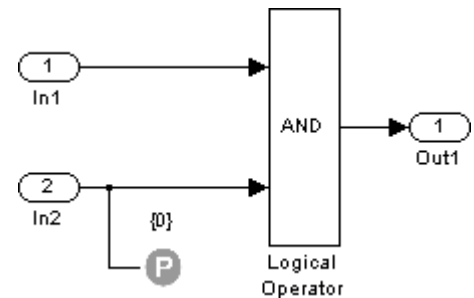
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

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

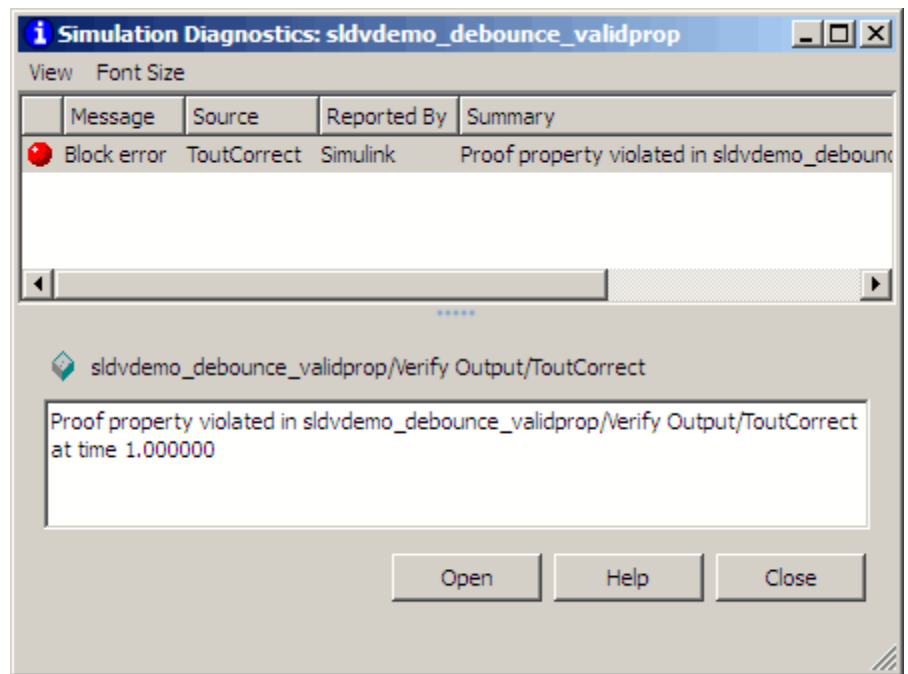


Pass through style: deselected

Stop simulation when the property is violated

Specify whether to stop the simulation if the simulation encounters a signal that violates the property specified in the **Values** parameter.

If you select this parameter and simulate the model, the simulation stops if it encounters a violation of the specified property, as in the following figure.



See Also

Proof Assumption, Test Objective

Test Condition

Purpose Constrain signal values in test cases

Library Simulink Design Verifier

Description



When operating in test generation mode, the Simulink Design Verifier software produces test cases that satisfy specified criteria (see Chapter 7, “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. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software attempts to produce test cases that satisfy the condition.

The block’s parameter dialog box also allows you to

- Enable or disable the condition.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink and Real-Time Workshop software ignore the Test Condition block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Test Condition block only when generating test cases for a model.

Specifying Test Conditions

Use the **Values** parameter to constrain signal values in test cases. Specify any combination of scalars and intervals in the form of a MATLAB cell array. (For information about cell arrays, see “Cell Arrays” in the MATLAB documentation.)

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:

Test Condition

- 0 — a scalar
- [1, 3] — a closed interval

The **Values** parameter

```
{Sldv.Interval(0, 1, '['), Sldv.Point(1)}
```

specifies:

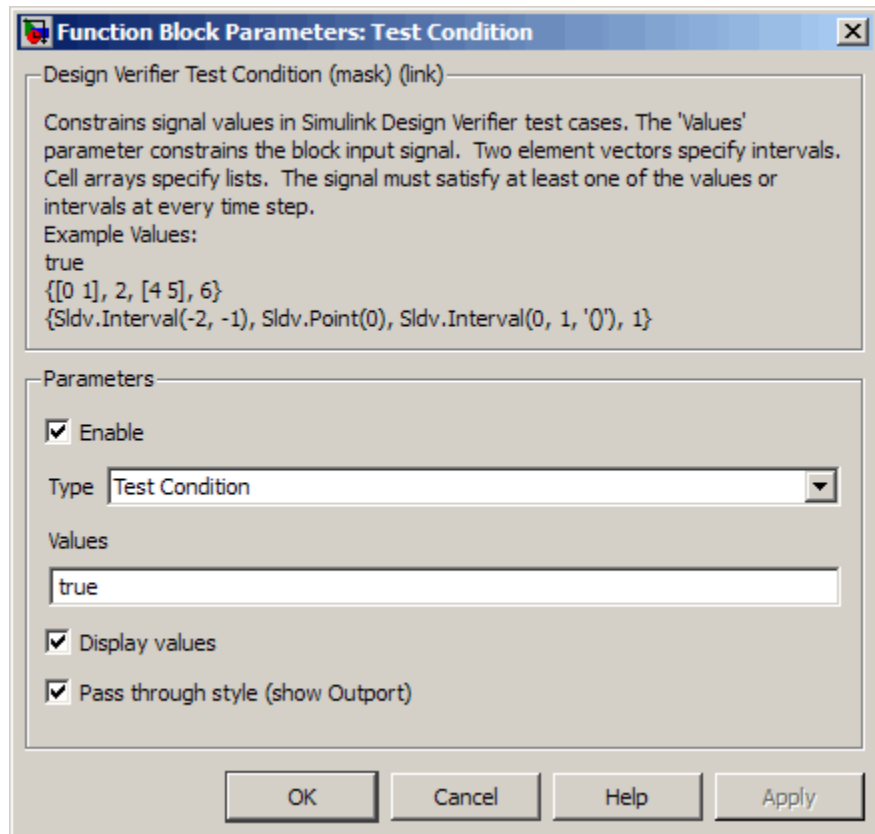
- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

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

Data Type Support

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

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), Simulink Design Verifier software uses the block when generating tests for a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Test Condition block did not exist. If this option is not selected, the block appears grayed out in the model editor.

Test Condition

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

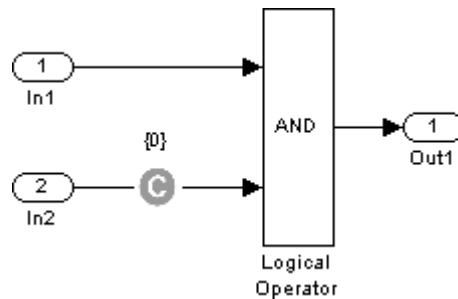
Specify the test condition (see “Specifying Test Conditions” on page 12-14).

Display values

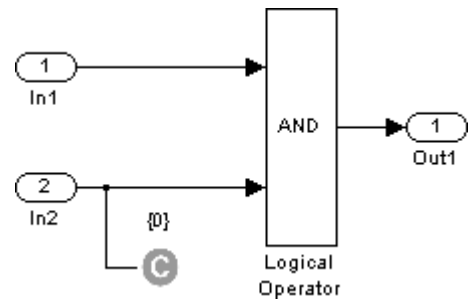
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

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

Define custom objectives that signals must satisfy in test cases

Library

Simulink Design Verifier

Description



When operating in test generation mode, the Simulink Design Verifier software produces test cases that satisfy specified criteria (see Chapter 7, “Generating Test Cases”). In this mode, you can use Test Objective blocks to define custom test objectives for signals in your model. The **Values** parameter lets you specify values that a signal must achieve for at least one time step during a test case simulation. The block applies the specified **Values** parameter to its input signal, and the Simulink Design Verifier software attempts to produce test cases that satisfy the objective.

The block’s parameter dialog box also allows you to

- Enable or disable the objective.
- Specify that the block should display its **Values** parameter in the model editor.
- Specify that the block should display its output port.

Note The Simulink and Real-Time Workshop software ignore the Test Objective block during model simulation and code generation, respectively. The Simulink Design Verifier software uses the Test Objective block only when generating test cases for a model.

Specifying Test Objectives

Use the **Values** parameter to define custom objectives that signals must satisfy in test cases. Specify any combination of scalars and intervals in the form of a MATLAB cell array. (For information about cell arrays, see “Cell Arrays” in the MATLAB documentation.)

Test Objective

Tip If the **Values** parameter specifies only one scalar value, you do not need to enter it in the form of a MATLAB cell array.

Scalar values each comprise a single cell in the array, for example:

`{0, 5}`

A closed interval comprises a two-element vector as a cell in the array, where each element specifies an interval endpoint:

`{[1, 2]}`

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

- `'()'` — Defines an open interval.
- `'[]'` — Defines a closed interval.
- `'(]'` — Defines a left-open interval.
- `'[)'` — Defines a right-open interval.

Note By default, `Sldv.Interval` considers an interval to be closed if you omit its third input argument.

As an example, the **Values** parameter

`{0, [1, 3]}`

specifies:

- 0 — a scalar
- [1, 3] — a closed interval

The **Values** parameter

```
{Sldv.Interval(0, 1, '[') }, Sldv.Point(1)}
```

specifies:

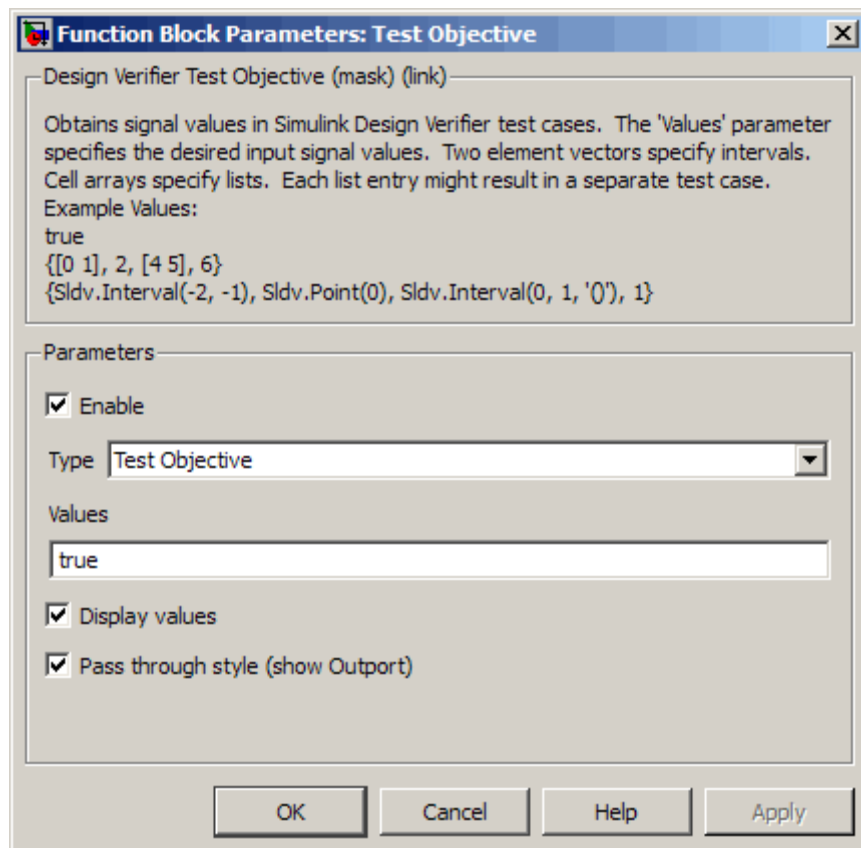
- `Sldv.Interval(0, 1, '[')` — the right-open interval [0, 1)
- `Sldv.Point(1)` — a scalar

Data Type Support

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

Test Objective

Parameters and Dialog Box



Enable

Specify whether the block is enabled. If selected (the default), the Simulink Design Verifier software uses the block when generating tests for a model. Clearing this option disables the block, that is, causes the Simulink Design Verifier software to behave as if the Test Objective block did not exist. If this option is not selected, the block appears grayed out in the model editor.

Type

Specify whether the block behaves as a Test Objective or Proof Objective block. Select **Proof Objective** to transform the Test Objective block into a Proof Objective block.

Values

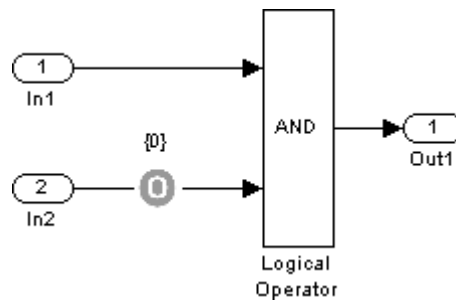
Specify the test objective (see “Specifying Test Objectives” on page 12-19).

Display values

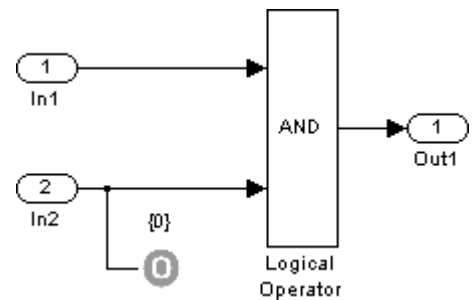
Specify whether the block displays the contents of its **Values** parameter in the model editor. By default, this option is selected.

Pass through style

Specify whether the block displays an output port in the model editor. If selected (the default), the block displays its output port, allowing its input signal to pass through as the block output. If not selected, the block hides its output port and terminates the input signal. The following figure illustrates the appearance of the block in each case.



Pass through style: selected



Pass through style: deselected

See Also

Proof Objective, Test Condition

Verification Subsystem

Purpose

Represent subsystem that specifies proof or test objectives without impacting simulation results or generated code

Library

Simulink Design Verifier

Description



This block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem that specifies proof or test objectives for use with the Simulink Design Verifier software. The Real-Time Workshop software ignores Verification Subsystem blocks during code generation, behaving as if the subsystems do not exist. A Verification Subsystem block allows you to add Simulink Design Verifier components to a model without affecting its generated code.

To create a Verification Subsystem in your model:

- 1 Copy the Verification Subsystem block from the Simulink Design Verifier library into your model.
- 2 Open the Verification Subsystem block by double-clicking it.
- 3 In the Verification Subsystem window, add blocks that specify proof or test objectives. Use Inport blocks to represent input from outside the subsystem.

The Verification Subsystem block in the Simulink Design Verifier library is preconfigured to work correctly. For correct behavior, a Verification Subsystem block must

- Contain no Outport blocks.
- Enable its **Treat as Atomic Unit** parameter.
- Specify its **Mask type** parameter as `VerificationSubsystem`.

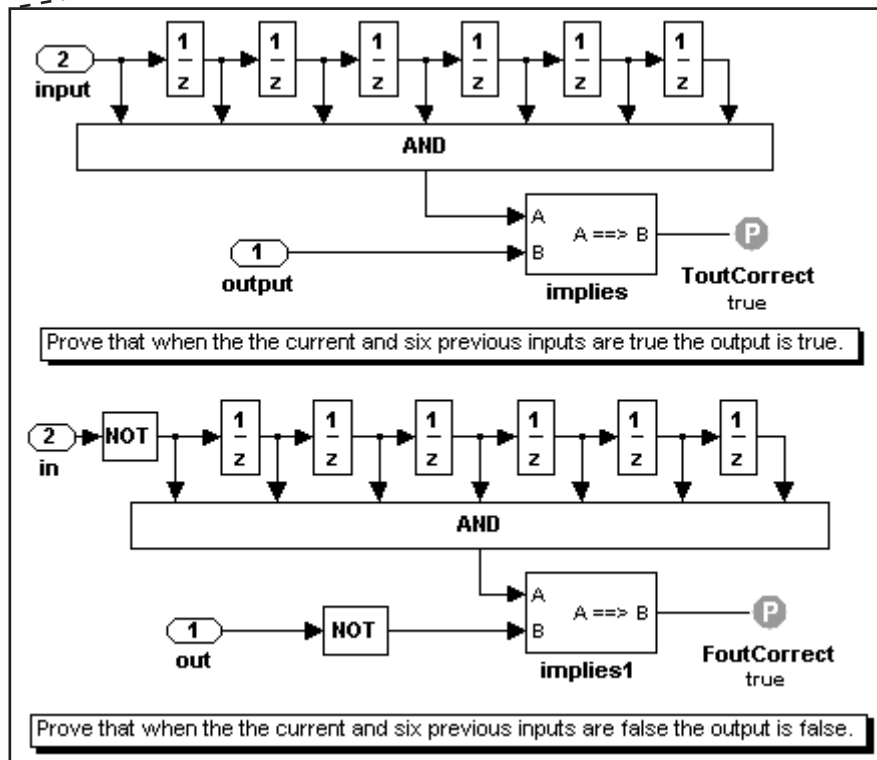
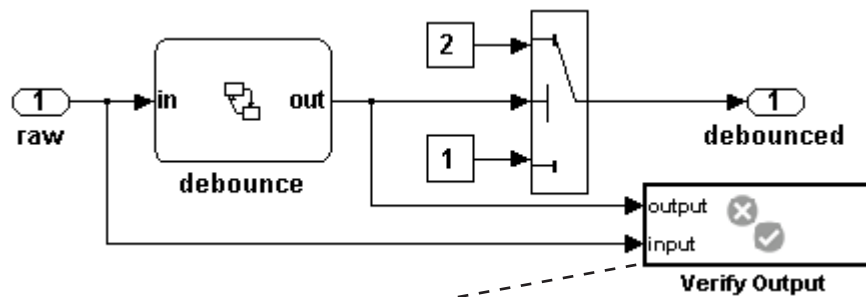
Note If you alter a Verification Subsystem block so that it no longer behaves correctly, the Simulink Design Verifier software displays a warning.

See the Subsystem block in the *Simulink Reference* and “Creating Subsystems” in *Simulink User’s Guide* for more information.

Examples

The `sldvdemo_debounce_validprop` demo model includes a Verification Subsystem that specifies two proof objectives, as shown in the following figure.

Verification Subsystem



See Also Proof Assumption, Proof Objective, Test Condition, Test Objective

Verification Subsystem

Configuration Parameters

- “Design Verifier Pane” on page 13-2
- “Design Verifier Pane: Block Replacements” on page 13-10
- “Design Verifier Pane: Parameters” on page 13-15
- “Design Verifier Pane: Test Generation” on page 13-18
- “Design Verifier Pane: Property Proving” on page 13-26
- “Design Verifier Pane: Results” on page 13-32
- “Design Verifier Pane: Report” on page 13-46
- “Parameter Command-Line Information Summary” on page 13-52

Design Verifier Pane

The image shows a configuration window for the Design Verifier Pane, divided into two sections: "Analysis options" and "Output".

Analysis options

- Mode: Test generation (selected in a dropdown menu)
- Maximum analysis time (s): 600 (text input field)
- Display unsatisfiable test objectives
- Automatic stubbing of unsupported blocks and functions

Output

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

In this section...

“Design Verifier Pane Overview” on page 13-3

“Mode” on page 13-3

“Maximum analysis time” on page 13-5

“Display unsatisfiable test objectives” on page 13-6

“Automatic stubbing of unsupported blocks and functions” on page 13-7

“Output directory” on page 13-8

“Make output file names unique by adding a suffix” on page 13-9

Design Verifier Pane Overview

Specify analysis options and configure Simulink Design Verifier output.

Mode

Specify whether the Simulink Design Verifier software generates test cases or proves properties.

Settings

Default: Test generation

Test generation

Generates test cases for a model.

Property proving

Proves properties of a model.

Tip

The Simulink Design Verifier software specifies the value of this option automatically when you select the **Tools > Design Verifier > Generate Tests** or **Tools > Design Verifier > Prove Properties**.

Dependency

Selecting Test generation enables the **Display unsatisfiable test objectives** parameter.

The **Generate Tests** button changes to **Prove Properties** if the **Mode** parameter changes from Test generation to Property proving.

Command-Line Information

Parameter: DVMode

Type: string

Value: 'TestGeneration' | 'PropertyProving'

Default: 'TestGeneration'

See Also

- Generating Test Cases
- Proving Properties of a Model

Maximum analysis time

Specify the maximum time (in seconds) that the Simulink Design Verifier software spends analyzing a model.

Settings

Default: 600

The value you enter represents the maximum number of seconds the Simulink Design Verifier software analyzes your model.

Command-Line Information

Parameter: DVMaxProcessTime

Type: double

Value: any valid value

Default: 600

Display unsatisfiable test objectives

Specify whether to display a warning for unsatisfiable test objectives. For more information about using this option, see “Display unsatisfiable test objectives” on page 6-6.

Settings

Default: On



On

Displays a warning in the Simulation Diagnostics Viewer when the Simulink Design Verifier software is unable to satisfy a test objective.



Off

Does not display a warning when the Simulink Design Verifier software is unable to satisfy a test objective.

Command-Line Information

Parameter: `DVDisplayUnsatisfiableObjectives`

Type: string

Value: 'on' | 'off'

Default: 'on'

Automatic stubbing of unsupported blocks and functions

Specify whether or not Simulink Design Verifier software should ignore unsupported blocks and functions and proceed with the analysis.

Settings

Default: Off



On

Ignores unsupported blocks and functions and proceeds with the analysis.



Off

Displays a warning when the Simulink Design Verifier software encounters an unsupported block or function and asks if you want to continue the analysis.

Command-Line Information

Parameter: AutomaticStubbing

Type: string

Value: 'on' | 'off'

Default: 'off'

Output directory

Specify a folder to which the Simulink Design Verifier software writes its output.

Settings

Default: `sldv_output/$ModelName$`

- Enter a path that is either absolute or relative to the current folder.
- `$ModelName$` is a token that represents the model name.

Tip

You can use the following parameters to customize the names and locations of Simulink Design Verifier output:

- **Data file name**
- **Harness model file name**
- **SystemTest file name**
- **Report file name**
- **File path of the output model**

Command-Line Information

Parameter: `DVOutputDir`

Type: `string`

Value: any valid path

Default: `'sldv_output/$ModelName$'`

Make output file names unique by adding a suffix

Specify whether the Simulink Design Verifier software makes its output file names unique by appending a numeric suffix.

Settings

Default: On



On

Appends an incremental numeric suffix to Simulink Design Verifier output file names. Selecting this option prevents the software from overwriting existing files that have the same name.



Off

Does not append a suffix to Simulink Design Verifier output file names. In this case, the software might overwrite existing files that have the same name.

Command-Line Information

Parameter: DVMakeOutputFilesUnique

Type: string

Value: 'on' | 'off'

Default: 'on'

Design Verifier Pane: Block Replacements

The image shows a configuration window titled "Block replacements". It contains a checkbox labeled "Apply block replacements" which is currently unchecked. Below the checkbox is a large empty rectangular area labeled "List of block replacement rules (in order of priority)". At the bottom of the window, there is a section titled "Output model" containing a text input field labeled "File path of the output model:".

In this section...

“Block Replacements Pane Overview” on page 13-11

“Apply block replacements” on page 13-12

“List of block replacement rules” on page 13-13

“File path of the output model” on page 13-14

Block Replacements Pane Overview

Specify options that control how the Simulink Design Verifier software preprocesses the models it analyzes.

See Also

[Working with Block Replacements](#)

Apply block replacements

Specify whether the Simulink Design Verifier software replaces blocks in a model before its analysis.

Settings

Default: Off



On

Replaces blocks in a model before the Simulink Design Verifier software analyzes it.



Off

Does not replace blocks in a model before the Simulink Design Verifier software analyzes it.

Dependencies

This parameter enables **List of block replacement rules** and **File path of the output model**.

Command-Line Information

Parameter: DVBlockReplacement

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Working with Block Replacements

List of block replacement rules

Specify a list of block replacement rules that the Simulink Design Verifier software executes before its analysis.

Settings

Default: <FactoryDefaultRules>

- Specify block replacement rules as a list delimited by spaces, commas, or carriage returns.
- The Simulink Design Verifier software processes block replacement rules in the order that you list them.
- If you specify the default value, the Simulink Design Verifier software uses its factory default block replacement rules.

Dependency

This parameter is enabled by **Apply block replacements**.

Command-Line Information

Parameter: DVBlockReplacementRulesList

Type: string

Value: any rules

Default: '<FactoryDefaultRules>'

See Also

Working with Block Replacements

File path of the output model

Specify a folder and file name for the model that results after applying block replacement rules.

Settings

Default: \$modelName\$_replacement

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name for the model that results after applying block replacement rules.
- \$modelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Apply block replacements**.

Command-Line Information

Parameter: DVBlockReplacementModelFileName

Type: string

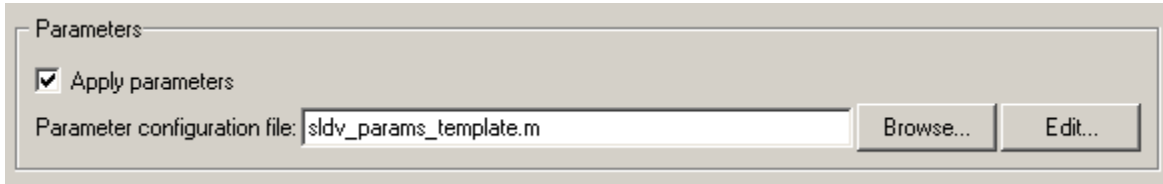
Value: any valid path and file name

Default: '\$modelName\$_replacement'

See Also

Working with Block Replacements

Design Verifier Pane: Parameters



In this section...

“Parameters Pane Overview” on page 13-16

“Apply parameters” on page 13-16

“Parameter configuration file” on page 13-16

Parameters Pane Overview

Specify options that control how the Simulink Design Verifier software uses parameter configurations when analyzing models.

Apply parameters

Specify whether the Simulink Design Verifier software uses parameter configurations when analyzing a model.

Settings

Default: On



On

The Simulink Design Verifier software uses parameter configurations when analyzing a model.



Off

The Simulink Design Verifier software does not use parameter configurations when analyzing a model.

Dependency

This parameter enables **Parameter configuration file**.

Command-Line Information

Parameter: DVParameters

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Specifying Parameter Configurations

Parameter configuration file

Specify an M-file function that defines parameter configurations for a model.

Settings

Default: sldv_params_template.m

- The default file, `sldv_params_template.m`, is a template that you can edit and save. The comments in the template explain the syntax you use to specify parameter configurations.
- Click the **Browse** button to select an existing M-file function using a file chooser dialog box.
- Click the **Edit** button to open the specified M-file function in an editor.

Dependency

This parameter is enabled by **Apply parameters**.

Command-Line Information

Parameter: DVParametersConfigFileName

Type: string

Value: any valid M-file function

Default: 'sldv_params_template.m'

See Also

Specifying Parameter Configurations

Design Verifier Pane: Test Generation

The image shows a configuration window titled "Test generation" with five rows of settings:

- Model coverage objectives: MCDC
- Test conditions: Use local settings
- Test objectives: Use local settings
- Maximum test case steps: 500
- Test suite optimization: CombinedObjectives

In this section...

“Test Generation Pane Overview” on page 13-19

“Model coverage objectives” on page 13-20

“Test conditions” on page 13-21

“Test objectives” on page 13-22

“Maximum test case steps” on page 13-23

“Test suite optimization” on page 13-24

Test Generation Pane Overview

Specify options that control how the Simulink Design Verifier software generates tests for the models it analyzes.

See Also

Generating Test Cases

Model coverage objectives

Specify the type of model coverage that the Simulink Design Verifier software attempts to achieve.

Settings

Default: MCDC

None

Generates test cases that achieve only the custom objectives that you specified in your model using, for example, Test Objective blocks.

Decision

Generates test cases that achieve decision coverage.

Condition Decision

Generates test cases that achieve condition and decision coverage.

MCDC

Generates test cases that achieve modified condition/decision coverage (MCDC).

When you set **Model coverage objectives** to MCDC, the Simulink Design Verifier software automatically enables every coverage objective for decision coverage and condition coverage as well. Similarly, enabling coverage for condition coverage causes every decision and condition coverage outcome to be enabled.

Command-Line Information

Parameter: DVModelCoverageObjectives

Type: string

Value: 'None' | 'Decision' | 'ConditionDecision' | 'MCDC'

Default: 'MCDC'

See Also

Generating Test Cases

Test conditions

Specify whether Test Condition blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Test Condition blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Test Condition blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Test Condition blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVTestConditions

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Test Condition
- Generating Test Cases

Test objectives

Specify whether Test Objective blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Test Objective blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Test Objective blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Test Objective blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVTestObjectives

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Test Objective
- Generating Test Cases

Maximum test case steps

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

Settings

Default: 500

You can specify a value that represents the maximum number of simulation steps the Simulink Design Verifier software takes when attempting to satisfy a test objective.

Command-Line Information

Parameter: DVMaxTestCaseSteps

Type: int32

Value: any valid value

Default: 500

See Also

Generating Test Cases

Test suite optimization

Specify the optimization strategy to use when generating test cases.

Settings

Default: Combined objectives

Combined objectives

Minimizes the number of test cases in a suite by generating cases that address more than one test objective. Each test case tends to be long, i.e., it includes many time steps.

Individual objectives

Maximizes the number of test cases in a suite by generating cases that each address only one test objective. Each test case tends to be short, i.e., it includes only a few time steps.

Large model

Minimizes the number of test cases in a suite by generating cases that address more than one test objective. This strategy is tailored for large, complex models; consequently, it tends to use all the time that the **Maximum analysis time** option allots.

Long test cases

Combines test cases to create a smaller number of test cases. This strategy generates fewer, but longer, test cases that each satisfy multiple test objectives and creates a more efficient analysis and easier-to-review results.

Tip

If an analysis using the **Combined objectives** strategy returns objectives without an outcome, set this option to **Individual objectives** and reanalyze the model. The **Individual objectives** strategy analyzes each objective independently and is better at identifying unsatisfiable objectives.

However, set this option to **Large model** if the model has both of the following characteristics:

- Nonlinearities, such as those that result from multiplying or dividing the model's input signals

- Numerous test objectives, such as those that result when using blocks that receive model coverage

The `Large model` strategy performs an analysis that is tailored to large, complex models; but, this strategy tends to use all the time that the **Maximum analysis time** option allots.

If you have a large number of test objectives, select `Long test cases` for a more efficient analysis and an easy-to-review report.

Command-Line Information

Parameter: `DVTestSuiteOptimization`

Type: `string`

Value: `'CombinedObjectives' | 'IndividualObjectives' | 'LargeModel' | 'LongTestCases'`

Default: `'CombinedObjectives'`

See Also

Generating Test Cases

Design Verifier Pane: Property Proving

Property proving

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

In this section...

“Property Proving Pane Overview” on page 13-27

“Assertion blocks” on page 13-28

“Proof assumptions” on page 13-29

“Strategy” on page 13-30

“Maximum violation steps” on page 13-31

Property Proving Pane Overview

Specify options that control how the Simulink Design Verifier software proves properties for the models it analyzes.

See Also

[Proving Properties of a Model](#)

Assertion blocks

Specify whether Assertion blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Assertion blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Assertion blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Assertion blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVAssertions

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Assertion
- Proving Properties of a Model

Proof assumptions

Specify whether Proof Assumption blocks in your model are enabled or disabled.

Settings

Default: Use local settings

Use local settings

Enables or disables Proof Assumption blocks based on the value of the **Enable** parameter of each block. If a block's **Enable** parameter is selected, the block is enabled; otherwise, the block is disabled.

Enable all

Enables all Proof Assumption blocks in the model regardless of the settings of their **Enable** parameters.

Disable all

Disables all Proof Assumption blocks in the model regardless of the settings of their **Enable** parameters.

Command-Line Information

Parameter: DVProofAssumptions

Type: string

Value: 'UseLocalSettings' | 'EnableAll' | 'DisableAll'

Default: 'UseLocalSettings'

See Also

- Proof Assumption
- Proving Properties of a Model

Strategy

Specify the strategy that the Simulink Design Verifier software uses when proving properties.

Settings

Default: Prove

Prove

Performs property proofs.

Find violation

Searches for property violations within the number of simulation steps specified by the **Maximum violation steps** option.

Prove with violation detection

Searches for property violations within the number of simulation steps specified by the **Maximum violation steps** option; then it attempts to prove properties for which it failed to detect a violation.

Dependency

Selecting Find violation or Prove with violation detection enables the **Maximum violation steps** parameter.

Command-Line Information

Parameter: DVProvingStrategy

Type: string

Value: 'Prove' | 'FindViolation' | 'ProveWithViolationDetection'

Default: 'Prove'

See Also

Proving Properties of a Model

Maximum violation steps

Specify the maximum number of simulation steps over which the Simulink Design Verifier software searches for property violations.

Settings

Default: 20

The Simulink Design Verifier software does not search beyond the maximum number of simulation steps that you specify. Therefore, it cannot identify violations that might occur later in a simulation.

Dependency

This parameter is enabled by **Strategy**.

Command-Line Information

Parameter: DVMaxViolationSteps

Type: int32

Value: any valid value

Default: 20

See Also

Proving Properties of a Model

Design Verifier Pane: Results

Data file options

Save test data to file

Data file name:

Include expected output values

Randomize data that do not affect the outcome

Harness model options

Save test harness as model

Harness model file name:

Reference input model in generated harness

SystemTest options

Save test harness as SystemTest TEST-file (will reference saved data file)

SystemTest file name:

In this section...

- “Results Pane Overview” on page 13-34
- “Save test data to file” on page 13-35
- “Data file name” on page 13-36
- “Include expected output values” on page 13-37
- “Randomize data that does not affect outcome” on page 13-39
- “Save test harness as model” on page 13-41
- “Harness model file name” on page 13-42
- “Reference input model in generated harness” on page 13-43

In this section...

“Save test harness as SystemTest TEST-file (will reference saved data file)”
on page 13-44

“SystemTest file name” on page 13-45

Results Pane Overview

Specify options that control how the Simulink Design Verifier software handles the results that it generates.

See Also

Reviewing the Results

Save test data to file

Save the test data that the Simulink Design Verifier software generates to a MAT-file.

Settings

Default: On



On

Saves the test data that the Simulink Design Verifier software generates to a MAT-file.



Off

Does not save the test data that the Simulink Design Verifier software generates.

Dependency

This parameter enables **Data file name**.

Command-Line Information

Parameter: DVSaveDataFile

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Data file name

Specify a folder and file name for the MAT-file that contains the data generated during the analysis, stored in an `sldvData` structure.

Settings

Default: `$ModelName$_sldvdata`

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name for the MAT-file.
- `$ModelName$` is a token that represents the model name.

Dependency

This parameter is enabled by **Save test data to file**.

Command-Line Information

Parameter: `DVDataFileName`

Type: `string`

Value: any valid path and file name

Default: `'$ModelName$_sldvdata'`

See Also

Reviewing the Results

Include expected output values

Simulate the model using test case signals and include the output values in the Simulink Design Verifier data file.

Settings

Default: Off



On

Simulates the model using the test case signals that the Simulink Design Verifier software produces. For each test case, the software collects the simulation output values associated with Outport blocks in the top-level system and includes those values in the MAT-file that it generates.



Off

Does not simulate the model and collect output values for inclusion in the MAT-file that the Simulink Design Verifier software generates.

Tips

- The `TestCases.expectedOutput` subfield of the MAT-file contains the output values. For more information, see “Overview of the `sldvData` Structure” on page 9-2.
- When **Include expected output values** is enabled, the Simulink Design Verifier software successively simulates the model using each test case that it generates. Enabling this option requires more time for the Simulink Design Verifier software to complete its analysis.

Dependency

This parameter is enabled by **Save test data to file**.

Command-Line Information

Parameter: DVSaveExpectedOutput

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

Randomize data that does not affect outcome

Use random values instead of zeros for input signals that have no impact on test or proof objectives.

Settings

Default: Off



On

Assigns random values to test case or counterexample signals that do not affect the outcome of test or proof objectives in a model. This option can enhance traceability and improve your regression tests.

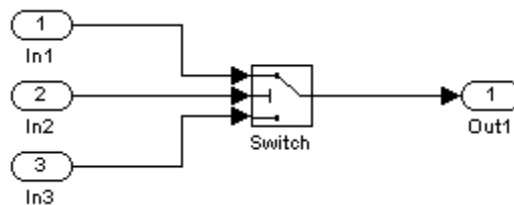


Off

Assigns zeros to test case or counterexample signals that do not affect the outcome of test or proof objectives in a model.

Tips

- This option assigns random values to test case or counterexample signals that otherwise would be zero. In the Simulink Design Verifier report, the Generated Input Data table always displays a dash (–) for such signals.
- Enable this option to enhance traceability when simulating test cases or counterexamples. For instance, consider the following model:



Only the signal entering the Switch block's control port impacts its decision coverage. If the **Randomize data that does not affect outcome** parameter is off, the Simulink Design Verifier software uses zeros to represent the signals from In1 and In3. When inspecting the results from test case or counterexample simulations, it is unclear which of these signals passes through the Switch block because they have the same value. But if the **Randomize data that does not affect outcome** parameter is on, the

software uses unique values to represent each of those signals. In this case, it is easier to determine which signal passes through the Switch block.

Dependency

This parameter is enabled by **Save test data to file**.

Command-Line Information

Parameter: DVRandomizeNoEffectData

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

Save test harness as model

Save the test harness that the Simulink Design Verifier software generates as a model file.

Settings

Default: On



On

Saves the test harness that the Simulink Design Verifier software generates as a model file.



Off

Does not save the test harness that the Simulink Design Verifier software generates.

Dependency

This parameter enables **Harness model file name**.

Command-Line Information

Parameter: DVSaveHarnessModel

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Harness model file name

Specify a folder and file name for the test harness model.

Settings

Default: \$ModelName\$_harness

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name for the test harness model.
- \$ModelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Save test harness as model**.

Command-Line Information

Parameter: DVHarnessModelFileName

Type: string

Value: any valid path and file name

Default: '\$ModelName\$_harness'

See Also

Reviewing the Results

Reference input model in generated harness

Use model reference to run the model in the test harness.

Settings

Default: Off



On

Uses model reference to run the model in the test harness.



Off

Uses a copy of the model in the test harness.

Command-Line Information

Parameter: DVModelReferenceHarness

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

Save test harness as SystemTest TEST-file (will reference saved data file)

Save the test harness as a SystemTest TEST-file so you can run test cases using the SystemTest capabilities.

Note The option to create a SystemTest TEST-file is only available in test-generation mode; you cannot create this file when running a property-proving analysis.

Settings

Default: Off



On

Saves the test harness as a SystemTest TEST-file.



Off

Does not save the test harness as a SystemTest TEST-file.

Dependency

This parameter enables **SystemTest file name**.

Command-Line Information

Parameter: DVSaveSystemTestHarness

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

SystemTest file name

Specify a folder and file name for the SystemTest TEST-file.

Settings

Default: \$ModelName\$_harness

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name for the SystemTest TEST-file.
- \$ModelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Save test harness as SystemTest TEST-file (will reference saved data file)**.

Command-Line Information

Parameter: DVMSystemTestFileName

Type: string

Value: any valid path and file name

Default: '\$ModelName\$_harness'

See Also

Reviewing the Results

Design Verifier Pane: Report

Report

Generate report of the results

Report file name:

Include screen shots of properties and test objectives

Display report

In this section...

“Report Pane Overview” on page 13-47

“Generate report of the results” on page 13-48

“Report file name” on page 13-49

“Include screen shots of properties and text objectives” on page 13-50

“Display report” on page 13-51

Report Pane Overview

Specify options that control how the Simulink Design Verifier software reports its results.

See Also

Reviewing the Results

Generate report of the results

Generate and save a Simulink Design Verifier report.

Settings

Default: on



On

Saves the HTML report that the Simulink Design Verifier software generates.



Off

Does not generate a Simulink Design Verifier report.

Dependencies

When this parameter is enabled, you must enable **Save test harness as model**.

This parameter enables the following parameters:

- **Report file name**
- **Include screen shots of properties and test objectives**
- **Display report**

Command-Line Information

Parameter: DVSaveReport

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Report file name

Specify a folder and file name for the report that Simulink Design Verifier software generates.

Settings

Default: \$ModelName\$_report

- Optionally, enter a path that is either absolute or relative to the path specified in **Output directory**.
- Enter a file name for the report Simulink Design Verifier software generates.
- \$ModelName\$ is a token that represents the model name.

Dependency

This parameter is enabled by **Generate report of the results**.

Command-Line Information

Parameter: DVReportFileName

Type: string

Value: any valid path and file name

Default: '\$ModelName\$_report'

See Also

Reviewing the Results

Include screen shots of properties and text objectives

Includes screen shots of properties in the Simulink Design Verifier report. Only valid in property-proving mode.

Settings

Default: Off



On

Includes screen shots of properties in the Simulink Design Verifier report. Only valid in property-proving mode.



Off

Does not include screen shots of properties in the Simulink Design Verifier report.

Dependency

This parameter is enabled by **Generate report of the results**.

Command-Line Information

Parameter: DVReportIncludeGraphics

Type: string

Value: 'on' | 'off'

Default: 'off'

See Also

Reviewing the Results

Display report

Display the report that the Simulink Design Verifier software generates after completing its analysis.

Settings

Default: On



On

Displays the report that the Simulink Design Verifier software generates after completing its analysis.



Off

Does not display the report that the Simulink Design Verifier software generates after completing its analysis.

Dependency

This parameter is enabled by **Generate report of the results**.

Command-Line Information

Parameter: DVDisplayReport

Type: string

Value: 'on' | 'off'

Default: 'on'

See Also

Reviewing the Results

Parameter Command-Line Information Summary

The following table lists parameters that you can use to configure the behavior of the Simulink Design Verifier software. Use the `get_param` and `set_param` functions to retrieve and specify values for these parameters programmatically.

For each parameter listed in the table, the **Description** column indicates where you can set its value on the Configuration Parameters dialog box. The **Values** column shows the type of value required, the possible values (separated with a vertical line), and the default value (enclosed in braces).

Parameter	Description	Values
DVAssertions	Set by the Assertion blocks option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
DVBlockReplacement	Set by the Apply block replacements option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	'on' {'off'}
DVBlockReplacementModel-FileName	Set by the File path of the output model option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	string {'\$modelName\$_replacement'}
DVBlockReplacementRules-List	Set by the List of block replacement rules option on the Design Verifier > Block Replacements pane of the Configuration Parameters dialog box.	string {'<FactoryDefaultRules>'}

Parameter	Description	Values
DVDataFileName	Set by the Data file name option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	string { '\$ModelName\$_sldvdata' }
DVDisplayUnsatisfiableObjectives	Set by the Display unsatisfiable test objectives option on the Design Verifier pane of the Configuration Parameters dialog box.	{ 'on' } 'off'
DVHarnessModelFileName	Set by the Harness model file name option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	string { '\$ModelName\$_harness' }
DVMakeOutputFilesUnique	Set by the Make output file names unique by adding a suffix check box on the Design Verifier pane of the Configuration Parameters dialog box.	{ 'on' } 'off'
DVMaxProcessTime	Set by the Maximum analysis time option on the Design Verifier pane of the Configuration Parameters dialog box.	double { '600' }
DVMaxTestCaseSteps	Set by the Maximum test case steps option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	int32 { '500' }

Parameter	Description	Values
DVMaxViolationSteps	Set by the Maximum violation steps option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	int32 {'20'}
DVMode	Set by the Mode option on the Design Verifier pane of the Configuration Parameters dialog box.	{'TestGeneration'} 'PropertyProving'
DVModelCoverageObjectives	Set by the Model coverage objectives option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'None' 'Decision' 'ConditionDecision' {'MCDC'}
DVModelReferenceHarness	Set by the Reference input model in generated harness option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}
DVOutputDir	Set by the Output directory option on the Design Verifier pane of the Configuration Parameters dialog box.	string {'sldv_output/\$ModelName\$'}
DVOutputDir	Set by the Output directory option on the Design Verifier pane of the Configuration Parameters dialog box.	string {'sldv_output/\$ModelName\$'}

Parameter	Description	Values
DVParameters	Set by the Apply parameters option on the Design Verifier > Parameters pane of the Configuration Parameters dialog box.	{'on'} {'off'}
DVParametersConfigFileName	Set by the Parameter configuration file option on the Design Verifier > Parameters pane of the Configuration Parameters dialog box.	string {'sldv_params_template.m'}
DVProofAssumptions	Set by the Proof assumptions option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' {'UseLocalSettings'}
DVProvingStrategy	Set by the Strategy option on the Design Verifier > Property Proving pane of the Configuration Parameters dialog box.	'FindViolation' {'Prove'} 'ProveWithViolationDetection'
DVRandomizeNoEffectData	Set by the Randomize data that does not affect outcome option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}
DVReportFileName	Set by the Report file name option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	string {'\$ModelName\$_report'}

Parameter	Description	Values
DVReportIncludeGraphics	Set by the Include screen shots of properties and test objectives option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	'on' {'off'}
DVSaveDataFile	Set by the Save test data to file option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	{'on'} 'off'
DVSaveExpectedOutput	Set by the Include expected output values option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}
DVSaveHarnessModel	Set by the Save test harness as model option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	{'on'} 'off'
DVSaveReport	Set by the Generate report of the results option on the Design Verifier > Report pane of the Configuration Parameters dialog box.	{'on'} 'off'
DVSaveSystemTestHarness	Set by the Save text harness as SystemTest TEST-file (will reference saved data file) option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	'on' {'off'}

Parameter	Description	Values
DVSystemTestFileName	Set by the SystemTest file name option on the Design Verifier > Results pane of the Configuration Parameters dialog box.	string { '\$modelName\$_harness' }
DVTestConditions	Set by the Test conditions option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' { 'UseLocalSettings' }
DVTestObjectives	Set by the Test objectives option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	'EnableAll' 'DisableAll' { 'UseLocalSettings' }
DVTestSuiteOptimization	Set by the Test suite optimization option on the Design Verifier > Test Generation pane of the Configuration Parameters dialog box.	{ 'CombinedObjectives' } 'IndividualObjectives' 'LargeModel' 'LongTestCases'

Simulink Block Support

- “Overview of Simulink Block Support” on page 14-2
- “Additional Math and Discrete Library” on page 14-3
- “Commonly Used Blocks Library” on page 14-4
- “Continuous Library” on page 14-5
- “Discontinuities Library” on page 14-6
- “Discrete Library” on page 14-7
- “Logic and Bit Operations Library” on page 14-8
- “Lookup Tables Library” on page 14-9
- “Math Operations Library” on page 14-10
- “Model Verification Library” on page 14-12
- “Model-Wide Utilities Library” on page 14-13
- “Ports & Subsystems Library” on page 14-14
- “Signal Attributes Library” on page 14-15
- “Signal Routing Library” on page 14-16
- “Sinks Library” on page 14-17
- “Sources Library” on page 14-18
- “User-Defined Functions Library” on page 14-19

Overview of Simulink Block Support

The following tables summarize the Simulink Design Verifier software's support for Simulink blocks. Each table lists all the blocks in that Simulink library and describes support information for that particular block. A dash (—) indicates that the software supports that block under all conditions.

If the software does not support a given block, you can turn on automatic stubbing, which considers the interface of the unsupported blocks, but not their behavior. However, if any of the unsupported blocks affect the simulation outcome, the analysis may achieve only partial results.

For details about automatic stubbing, see “Handling Incompatibilities with Automatic Stubbing” on page 2-6.

Additional Math and Discrete Library

The Simulink Design Verifier software supports all blocks in the Additional Math and Discrete library.

Commonly Used Blocks Library

The Commonly Used Blocks library includes blocks from other libraries. Those blocks are listed under their respective libraries.

Continuous Library

Block	Support Notes
Derivative	Not supported
Integrator	Not supported
PID Controller	Not supported
PID Controller (2 DOF)	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 Library

Block	Support Notes
Backlash	Not supported
Coulomb & Viscous Friction	—
Dead Zone	Not supported
Dead Zone Dynamic	—
Hit Crossing	—
Quantizer	—
Rate Limiter	—
Rate Limiter Dynamic	—
Relay	—
Saturation	—
Saturation Dynamic	—
Wrap To Zero	—

Discrete Library

Block	Support Notes
Difference	—
Discrete Derivative	Not supported
Discrete Filter	—
Discrete FIR Filter	—
Discrete PID Controller	—
Discrete PID Controller (2 DOF)	—
Discrete State-Space	Not supported
Discrete Transfer Fcn	—
Discrete Zero-Pole	Not supported
Discrete-Time Integrator	—
First-Order Hold	—
Integer Delay	—
Memory	—
Tapped Delay	—
Transfer Fcn First Order	—
Transfer Fcn Lead or Lag	—
Transfer Fcn Real Zero	—
Unit Delay	—
Zero-Order Hold	—

Logic and Bit Operations Library

Block	Support Notes
Bit Clear	—
Bit Set	—
Bitwise Operator	—
Combinatorial Logic	—
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 when the Number of bits to shift right parameter specifies a vector and the block's input or output signal has a data type other than <code>single</code> or <code>double</code> .

Lookup Tables Library

Block	Support Notes
Cosine	Not supported
Direct Lookup Table (n-D)	—
Interpolation Using Prelookup	Not supported
Lookup Table	Supported unless the Lookup method block parameter specifies Interpolation-Extrapolation and the block's input and output signals do not have the same floating-point data type.
Lookup Table (2-D)	Supported unless the Lookup method block parameter specifies Interpolation-Extrapolation and the block's input and output signals do not have the same floating-point data type.
Lookup Table (n-D)	<p>Input and output must have the same data type—either single or double.</p> <p>Not supported when either the Interpolation method or the Extrapolation method parameter specifies Cubic Spline.</p> <p>Supports only Number of table dimensions that specify either 1 or 2.</p>
Lookup Table Dynamic	Not supported
Prelookup	—
Sine	Not supported

Math Operations Library

Block	Support Notes																
Abs	—																
Add	—																
Algebraic Constraint	—																
Assignment	—																
Bias	—																
Complex to Magnitude-Angle	—																
Complex to Real-Imag	—																
Divide	—																
Dot Product	—																
Gain	—																
Magnitude-Angle to Complex	Not supported																
Math Function	<p>All signal types support the following Function parameter settings.</p> <table border="1"> <tbody> <tr> <td>conj</td> <td>hermitian</td> <td>magnitude^2</td> <td>mod</td> </tr> <tr> <td>rem</td> <td>reciprocal</td> <td>square</td> <td>transpose</td> </tr> </tbody> </table> <p>Integer and fixed-point signal types support the following Function parameter settings.</p> <table border="1"> <tbody> <tr> <td>sqrt</td> <td>1/sqrt</td> </tr> </tbody> </table> <p>The Simulink Design Verifier software does not support the following Function parameter settings.</p> <table border="1"> <tbody> <tr> <td>10^u</td> <td>exp</td> <td>hypot</td> </tr> <tr> <td>log</td> <td>log10</td> <td>pow</td> </tr> </tbody> </table>	conj	hermitian	magnitude^2	mod	rem	reciprocal	square	transpose	sqrt	1/sqrt	10^u	exp	hypot	log	log10	pow
conj	hermitian	magnitude^2	mod														
rem	reciprocal	square	transpose														
sqrt	1/sqrt																
10^u	exp	hypot															
log	log10	pow															
Matrix Concatenate	—																
MinMax	—																

Block	Support Notes
MinMax Running Resettable	—
Permute Dimensions	—
Polynomial	—
Product	—
Product of Elements	—
Real-Imag to Complex	Not supported
Reshape	—
Rounding Function	—
Sign	—
Sine Wave Function	Not supported
Slider Gain	—
Squeeze	—
Subtract	—
Sum	—
Sum of Elements	—
Trigonometric Function	Not supported
Unary Minus	—
Vector Concatenate	—
Weighted Sample Time Math	Not supported

Model Verification Library

The Simulink Design Verifier software supports all blocks in the Model Verification library.

Model-Wide Utilities Library

Block	Support Notes
Block Support Table	—
DocBlock	—
Model Info	—
Time-Based Linearization	Not supported
Trigger-Based Linearization	Not supported

Ports & Subsystems Library

Block	Support Notes
Atomic Subsystem	—
Code Reuse Subsystem	—
Configurable Subsystem	—
Enabled Subsystem	—
Enabled and Triggered Subsystem	Not supported when the trigger control signal specifies a fixed-point data type.
For Iterator Subsystem	—
Function-Call Generator	—
Function-Call Subsystem	—
If	Parameter configurations are not supported for the If and Fcn blocks. The Simulink Design Verifier software ignores any parameter configurations that you specify for these blocks.
If Action Subsystem	—
Model	Supported except for limitations described in “Limitations of Support for Model Reference” on page 3-11.
Subsystem	—
Switch Case	—
Switch Case Action Subsystem	—
Triggered Subsystem	Not supported when the trigger control signal specifies a fixed-point data type.
While Iterator Subsystem	—

Signal Attributes Library

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

Signal Routing Library

The Simulink Design Verifier software supports all blocks in the Signal Routing library.

Sinks Library

Block	Support Notes
Display	—
Floating Scope	—
Outport (Out1)	—
Scope	—
Stop Simulation	Not supported
Terminator	—
To File	—
To Workspace	—
XY Graph	—

Sources Library

Block	Support Notes
Band-Limited White Noise	Not supported
Chirp Signal	Not supported
Clock	—
Constant	—
Counter Free-Running	—
Counter Limited	—
Digital Clock	—
Enumerated Constant	—
From File	Not supported
From Workspace	Not supported
Ground	—
Inport (In1)	—
Pulse Generator	Supports only Sample based for the Pulse type parameter; also, must specify a discrete sample time.
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

User-Defined Functions Library

Block	Support Notes
Embedded MATLAB Function	For limitations, see “Support Limitations for the Embedded MATLAB Subset” on page 3-15 for more information.
Fcn	<p>Supports all operators except ^, and supports only the mathematical functions <code>abs</code>, <code>ceil</code>, <code>fabs</code>, <code>floor</code>, <code>rem</code>, and <code>sgn</code>.</p> <p>Parameter configurations are not supported for the If and Fcn blocks. The Simulink Design Verifier software ignores any parameter configurations that you specify for these blocks.</p>
Level-2 M-file S-Function	Not supported
MATLAB Fcn	Not supported
S-Function	Not supported
S-Function Builder	Not supported

Embedded MATLAB Subset Support

This table lists only the Embedded MATLAB library functions for which the Simulink Design Verifier software provides no support or limited support. See “Embedded MATLAB Function Library Reference” for the complete listing of available functions.

Function	Support Notes
Arithmetic Operator Functions	
mldivide (\)	Supports only scalar arguments.
mpower (^)	Supports only integer exponents.
mrdivide (/)	Supports only scalar arguments.
power (.^)	Supports only integer exponents.
Casting Functions	
char	Not supported.
typecast	Not supported.
Complex Number Functions	
complex	Not supported.
imag	Not supported.
Error-Handling Functions	
assert	Supported, but does not behave like a Proof Objective block.
Exponential Functions	
exp	Not supported.
expm	Not supported.
expm1	Not supported.
log	Not supported.
log2	Not supported.
log10	Not supported.
log1p	Not supported.
nextpow2	Not supported.

Function	Support Notes
nthroot	Not supported.
reallog	Not supported.
realpow	Not supported.
realsqrt	Not supported.
sqrt	Not supported.
Filtering and Convolution Functions	
detrend	Not supported.
Fixed-Point Toolbox™ Functions	
complex	Not supported.
Interpolation and Computational Geometry	
cart2pol	Not supported.
cart2sph	Not supported.
pol2cart	Not supported.
sph2cart	Not supported.
Matrix and Array Functions	
angle	Not supported.
cond	Not supported.
det	Not supported.
eig	Not supported.
inv	Not supported.
invhilb	Not supported.
logspace	Not supported.
lu	Not supported.
norm	Supported only when invoked using the syntax $\text{norm}(A,p)$ where p is either 1 or inf .

Function	Support Notes
normest	Not supported.
pinv	Not supported.
planerot	Not supported.
qr	Not supported.
rank	Not supported.
rcond	Not supported.
subspace	Not supported.
Polynomial Functions	
poly	Not supported.
polyfit	Not supported.
Signal Processing Functions	
chol	Not supported.
fft	Not supported.
fftshift	Not supported.
ifft	Not supported.
ifftshift	Not supported.
sosfilt	Not supported.
svd	Not supported.
Special Values	
rand	Not supported.
randn	Not supported.
Specialized Math	
beta	Not supported.
betainc	Not supported.
betaln	Not supported.
ellipke	Not supported.

Function	Support Notes
erf	Not supported.
erfc	Not supported.
erfcinv	Not supported.
erfcx	Not supported.
erfinv	Not supported.
expint	Not supported.
gamma	Not supported.
gammainc	Not supported.
gamma1n	Not supported.
Statistical Functions	
std	Not supported.
String Functions	
char	Not supported.
ischar	Not supported.
Trigonometric Functions	
acos	Not supported.
acosd	Not supported.
acosh	Not supported.
acot	Not supported.
acotd	Not supported.
acoth	Not supported.
acsc	Not supported.
acscd	Not supported.
acsch	Not supported.
asec	Not supported.
asecd	Not supported.

Function	Support Notes
asech	Not supported.
asin	Not supported.
asinh	Not supported.
atan	Not supported.
atan2	Not supported.
atand	Not supported.
atanh	Not supported.
cos	Not supported.
cosd	Not supported.
cosh	Not supported.
cot	Not supported.
cotd	Not supported.
coth	Not supported.
csc	Not supported.
cscd	Not supported.
csch	Not supported.
hypot	Not supported.
sec	Not supported.
secd	Not supported.
sech	Not supported.
sin	Not supported.
sind	Not supported.
sinh	Not supported.
tan	Not supported.
tand	Not supported.
tanh	Not supported.

abstraction

The process of ignoring certain aspects of model behavior that do not affect the test objective or property under investigation.

analysis model

The target model for a Simulink Design Verifier analysis. If you select an atomic subsystem for analysis, the analysis model is generated by extracting the subsystem to a new model.

assumption

A property that is assumed to be true during a property proof. The proof result holds only when the assumption is true.

block replacement rule

A rule that is registered with the Simulink Design Verifier software and defines how instances of specific blocks are replaced by an alternate implementation. The software uses M-code to define when and how to apply a block replacement rule (see Chapter 4, “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. Enabling condition coverage causes every decision and condition coverage outcome to be enabled. See “Types of 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. Decision coverage is a subset of modified decision/condition coverage. See “Types of Model Coverage” in the *Simulink Verification and Validation User’s Guide*.

floating-point approximation

The process of approximating floating-point numbers using rational numbers (i.e., fractions whose numerator and denominator are small integers). The Simulink Design Verifier software performs floating-point approximations during its analysis. It can generate invalid test cases that result from numerical differences. For example, given a sufficiently large floating-point number x , the expression $x == (x+1)$ is true; however, this expression never holds 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. When you set the coverage objective to MCDC, Simulink Design Verifier automatically enables every coverage objective for decision coverage and condition coverage as well. See “Types of 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.

Automatic Stubbing

“Analyzing a Model Using Automatic Stubbing” on page 2-6

Working with Block Replacements

“Example: Replacing Multiport Switch Blocks” on page 4-8

“Configuring Block Replacements” on page 4-16

Specifying Parameter Configurations

“Constructing the Example Model” on page 5-8

“Parameterizing the Constant Block” on page 5-10

“Specifying a Parameter Configuration” on page 5-11

“Analyzing the Example Model” on page 5-13

“Simulating the Test Cases” on page 5-15

Generating Test Cases

“Constructing the Example Model” on page 7-6

“Checking Compatibility of the Example Model” on page 7-7

“Configuring Test Generation Options” on page 7-11

“Analyzing the Example Model” on page 7-14

“Customizing Test Generation” on page 7-22

“Reanalyzing the Example Model” on page 7-26

Proving Properties of a Model

“Constructing the Example Model” on page 8-6

“Instrumenting the Example Model” on page 8-11

“Configuring Property-Proving Options” on page 8-14

“Analyzing the Example Model” on page 8-16
“Customizing the Example Proof” on page 8-22
“Reanalyzing the Example Model” on page 8-25
“Property-Proving Examples” on page 8-29

Symbols and Numerics

2-D lookup tables
linearizing 2-15

A

AnalysisInformation field 9-4
analyzing large models
initial steps 10-4
analyzing models
overview 2-2
approximations
types 2-14
automatic stubbing
definition 2-6
enabling 2-10
workflow 2-6

B

block replacements
configuration 4-16
example 4-7
execution 4-17
factory defaults 4-3
introduction 4-2
template 4-6
block support
limitations 3-11
summary 14-1

C

combining
test cases 1-23
configuration parameters
block replacements 6-7
Block Replacements pane 13-11
Apply block replacements 13-12
File path of the output model 13-14

List of block replacement rules 13-13
Design Verifier 6-5
Design Verifier pane 13-3
Automatic stubbing of unsupported
blocks and functions 13-7
Display unsatisfiable test objectives 13-6
Make output file names unique by adding
a suffix 13-9
Maximum analysis time 13-5
Mode 13-3
Output directory 13-8
pane
Reference input model in generated
harness 13-43
Save test harness as SystemTest
TEST-file (will reference saved data
file) 13-44
SystemTest file name: 13-45
parameters 6-9
Parameters pane 13-16
Apply parameters 13-16
Parameter configuration file 13-16
property proving 6-12
Property Proving pane 13-27
Assertion blocks 13-28
Maximum violation steps 13-31
Proof assumptions 13-29
Strategy 13-30
report 6-17
Report pane 13-47
Display report 13-51
Generate report of the results 13-48
Include screen shots of properties and
test objectives 13-50
Report file name 13-49
results 6-14
Results pane 13-34
Data file name 13-36
Harness model file name 13-42
Include expected output values 13-37

- Randomize data that does not affect outcome 13-39
- Save test data to file 13-35
- Save test harness as model 13-41
- summary 13-52
- test generation 6-10
- Test Generation pane 13-19
 - Maximum test case steps 13-23
 - Model coverage objectives 13-20
 - Test conditions 13-21
 - Test objectives 13-22
 - Test suite optimization 13-24
- CounterExamples field 9-5

D

- discretization
 - constraining data 10-9

E

- Embedded MATLAB functions
 - for property proofs 8-2
 - for test cases 7-2
- Embedded MATLAB library functions
 - limitations 3-16
- Embedded MATLAB subset support
 - summary 15-2

F

- floating-point data
 - constraining for model analysis 10-9
 - converting to rational 2-14

G

- generating test cases 1-8

H

- harness. *See* test harness

L

- large model optimization 10-6
- large models
 - analyzing
 - first steps 10-4
 - complexity of 10-2
- linearizing
 - 2-D lookup tables 2-15
- lookup tables
 - linearizing 2-15

M

- model compatibility
 - checking 3-2
- ModelInformation field 9-3
- ModelObjects field 9-4
- models 10-2
 - analyzing, overview 2-2
 - complexity of 10-2
 - mathematical techniques for simplifying analysis 2-5
 - See also* large models

O

- Objectives field 9-5

P

- parameter configurations
 - example 5-7
 - introduction 5-2
 - syntax 5-4
 - template 5-3
- Proof Assumption block 12-3
- Proof Objective block 12-8

- property proofs
 - blocks 8-2
 - Embedded MATLAB functions 8-2
 - example 8-5
 - introduction 8-2
 - Stateflow actions 8-2
 - subsystems 8-28
 - workflow 8-4

R

- rational data
 - converting floating-point data to 2-14

S

- Simulink Design Verifier
 - model parameters 13-52
- Simulink Design Verifier data files
 - fields 9-3
 - overview 9-2
 - simulation 9-7
- Simulink Design Verifier options
 - saving 6-20
 - viewing 6-2
- Simulink Design Verifier reports
 - analysis information 9-20
 - approximations 9-24
 - block replacements summary 9-23
 - Constraints 9-22
 - model items 9-29
 - summary 9-19
 - table of contents 9-18
 - test cases/counterexamples 9-30
 - test/proof objectives 9-25
 - title 9-18
 - Unsupported Blocks 9-22
- Simulink® Design Verifier™ software
 - analyzing demo model 1-6
 - block library 1-4

- HTML report 1-15
 - starting 1-4
 - workflow 1-30
- sldv.assume function 11-2
- sldv.condition function 11-11
- sldv.prove function 11-33
- sldv.test function 11-48
- sldvblockreplacement function 11-7
- sldvcompat function 11-9
- sldvextract function 11-17
- sldvisactive function 11-23
- sldvoptions function 11-28
- sldvrun function 11-41
- sldvruntime function 11-44
- stubbing. *See* automatic stubbing
- subsystems
 - analyzing 1-26
 - generating test cases for 7-31
 - proving properties of 8-28
- system requirements 1-3

T

- test case generation
 - blocks 7-2
 - Embedded MATLAB functions 7-2
 - example 7-5
 - introduction 7-2
 - subsystems 7-31
 - test objectives 2-3
 - workflow 7-4
- test cases
 - combining 1-23
 - generating 1-8
- Test Condition block 12-14
- test harness
 - contents 1-10
- test harness models
 - anatomy 9-8
 - simulation 9-13

- Test Objective block 12-19
- test objectives
 - generating test cases 2-3
- test suite optimization
 - large model option 10-6
- TestCases field 9-5
- Trigonometric Function block
 - Simulink Design Verifier does not support 2-6

U

- unrolling
 - while loops 2-15

- unsupported features
 - Embedded MATLAB subset 3-15
 - Simulink 3-9
 - Stateflow 3-13

V

- Verification Subsystem block 12-24
- Version field 9-7

W

- while loops
 - unrolling 2-15