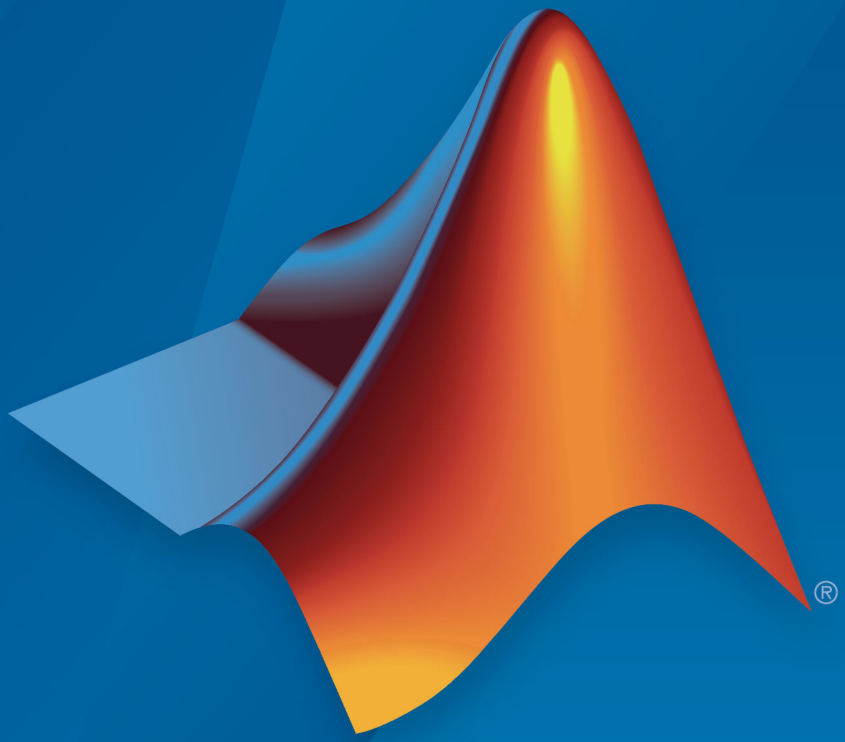
Simulink® Coverage™

User's Guide



MATLAB® & SIMULINK®

R2018a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

Simulink® Coverage ™ User's Guide

© COPYRIGHT 2017–2018 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

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

Revision History

September 2017	Online only	New for Version 4.0 (Release 2017b)
March 2018	Online only	Revised for Version 4.1 (Release 2018a)

Contents

1 Model Coverage Definition

Model Coverage **1-2**

Types of Model Coverage **1-3**
- Execution Coverage (EC) **1-3**
- Decision Coverage (DC) **1-3**
- Condition Coverage (CC) **1-3**
- Modified Condition/Decision Coverage (MCDC) **1-4**
- Cyclomatic Complexity **1-5**
- Lookup Table Coverage **1-5**
- Signal Range Coverage **1-6**
- Signal Size Coverage **1-6**
- Objectives and Constraints Coverage **1-7**
- Saturate on Integer Overflow Coverage **1-8**
- Relational Boundary Coverage **1-8**

Simulink Optimizations and Model Coverage **1-11**
- Inlined parameters **1-11**
- Block reduction **1-11**
- Conditional input branch execution **1-12**

2 Model Objects That Receive Model Coverage

Model Objects That Receive Coverage **2-2**
- Abs **2-8**
- Bias **2-9**
- Combinatorial Logic **2-9**
- Compare to Constant **2-9**
- Compare to Zero **2-10**

Data Type Conversion 2-10
Dead Zone 2-10
Direct Lookup Table (n-D) 2-11
Discrete Filter 2-12
Discrete FIR Filter 2-12
Discrete-Time Integrator 2-12
Discrete Transfer Fcn 2-13
Dot Product 2-13
Enabled Subsystem 2-13
Enabled and Triggered Subsystem 2-14
Fcn 2-15
For Iterator, For Iterator Subsystem 2-16
Gain 2-16
If, If Action Subsystem 2-16
Interpolation Using Prelookup 2-17
Library-Linked Objects 2-18
Logical Operator 2-18
1-D Lookup Table 2-18
2-D Lookup Table 2-19
n-D Lookup Table 2-20
Math Function 2-20
MATLAB Function 2-20
MATLAB System 2-21
MinMax 2-21
Model 2-21
Multiport Switch 2-22
PID Controller, PID Controller (2 DOF) 2-22
Product 2-23
Proof Assumption 2-23
Proof Objective 2-23
Rate Limiter 2-23
Relational Operator 2-24
Relay 2-25
C/C++ S-Function 2-25
Saturation 2-26
Saturation Dynamic 2-27
Simulink Design Verifier Functions in MATLAB Function Blocks 2-27
Sqrt, Signed Sqrt, Reciprocal Sqrt 2-28
Sum, Add, Subtract, Sum of Elements 2-28
Switch 2-28
SwitchCase, SwitchCase Action Subsystem 2-29
Test Condition 2-29
Test Objective 2-29

Triggered Models 2-30
Triggered Subsystem 2-31
Truth Table 2-31
Unary Minus 2-32
Weighted Sample Time Math 2-32
While Iterator, While Iterator Subsystem 2-32

Model Objects That Do Not Receive Coverage 2-33

3 Setting Coverage Options

Specify Coverage Options 3-2
Coverage Pane 3-2
Results Pane 3-7

Access, Manage, and Accumulate Coverage Results 3-10
Accessing Coverage Data from the Results Explorer 3-10
Managing Coverage Data from the Results Explorer 3-18
Accumulating Coverage Data from the Results Explorer 3-18

Cumulative Coverage Data 3-21

4 Code Coverage

Types of Code Coverage 4-2
Statement Coverage for Code Coverage 4-2
Condition Coverage for Code Coverage 4-3
Decision Coverage for Code Coverage 4-3
Modified Condition/Decision Coverage (MCDC) for Code Coverage 4-4
Cyclomatic Complexity for Code Coverage 4-5
Relational Boundary for Code Coverage 4-5
Function Coverage 4-5
Function Call Coverage 4-6

Code Coverage for Models in Software-in-the-Loop (SIL) Mode and Processor-in-the-Loop (PIL) Mode **4-7**
Enable SIL or PIL Code Coverage for a Model **4-7**
Simulink Coverage Code Coverage Measurement Workflows **4-8**
Review the Coverage Results for Models in SIL or PIL Mode **4-9**
Limitations **4-10**

Specify Code Coverage Options **4-11**
Models with S-Function Blocks **4-11**
Models with Software-in-the-Loop and Processor-in-the-Loop Mode Blocks **4-11**
Models with MATLAB Function Blocks **4-12**

Coverage for Models with Code Blocks and Simulink Blocks **4-13**
Set Up the Model to Record Coverage **4-13**
Record Coverage **4-14**
Review Results by Generating a Coverage Report **4-14**
Justify Missing Coverage **4-15**

5 Coverage Collection During Simulation

Model Coverage Collection Workflow **5-2**

Create and Run Test Cases **5-3**

Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage **5-4**
Differences between Masking MCDC and Unique-Cause MCDC in Simulink Coverage Coverage Analysis **5-4**
Certification Considerations for MCDC Coverage **5-6**
Setting the (MCDC) Definition Used for Simulink Coverage Coverage Analysis **5-6**
Modified Condition and Decision Coverage in Simulink Design Verifier **5-7**

Modified Condition and Decision Coverage in Simulink Design Verifier **5-8**
MCDC Definitions for Simulink Coverage and Simulink Design Verifier **5-8**

View Coverage Results in a Model **5-12**
Overview of Model Coverage Highlighting **5-12**
Enable Coverage Highlighting **5-13**
View Results in Coverage Display Window **5-16**

Model Coverage for Multiple Instances of a Referenced Model **5-19**
About Coverage for Model Blocks **5-19**
Record Coverage for Multiple Instances of a Referenced Model **5-19**

Model Coverage for MATLAB Functions **5-29**
About Model Coverage for MATLAB Functions **5-29**
Types of Model Coverage for MATLAB Functions **5-29**
How to Collect Coverage for MATLAB Functions **5-31**
Examples: Model Coverage for MATLAB Functions **5-32**

Coverage for C and C++ S-Functions **5-46**
Make S-Function Compatible with Model Coverage **5-46**
Generate Coverage Report for S-Function **5-47**

View Coverage Results for C/C++ Code in S-Function Blocks **5-49**

Model Coverage for Stateflow Charts **5-54**
How Model Coverage Reports Work for Stateflow Charts . . . **5-54**
Specify Coverage Report Settings for Stateflow Charts **5-55**
Cyclomatic Complexity for Stateflow Charts **5-55**
Decision Coverage for Stateflow Charts **5-56**
Condition Coverage for Stateflow Charts **5-59**
MCDC Coverage for Stateflow Charts **5-60**
Relational Boundary Coverage for Stateflow Charts **5-60**
Simulink Design Verifier Coverage for Stateflow Charts **5-60**
Model Coverage Reports for Stateflow Charts **5-62**
Model Coverage for Stateflow State Transition Tables **5-71**
Model Coverage for Stateflow Atomic Subcharts **5-72**
Model Coverage for Stateflow Truth Tables **5-75**
Colored Stateflow Chart Coverage Display **5-80**

6 Results Review

Types of Coverage Reports **6-2**
Model Summary Report **6-3**
Model Reference Coverage Report **6-4**
External MATLAB File Coverage Report **6-5**
Subsystem Coverage Report **6-9**
Code Coverage Report **6-11**

Top-Level Model Coverage Report **6-12**
Coverage Summary **6-12**
Details **6-14**
Cyclomatic Complexity **6-23**
Decisions Analyzed **6-25**
Conditions Analyzed **6-27**
MCDC Analysis **6-27**
Cumulative Coverage **6-29**
N-Dimensional Lookup Table **6-31**
Block Reduction **6-37**
Relational Boundary **6-38**
Saturate on Integer Overflow Analysis **6-42**
Signal Range Analysis **6-43**
Signal Size Coverage for Variable-Dimension Signals **6-45**
Simulink Design Verifier Coverage **6-46**

Export Model Coverage Web View **6-48**

7 Excluding Model Objects from Coverage

Coverage Filtering **7-2**
When to Use Coverage Filtering **7-2**
What Is Coverage Filtering? **7-2**

Coverage Filter Rules and Files **7-4**
What Is a Coverage Filter Rule? **7-4**
What Is a Coverage Filter File? **7-4**

Model Objects to Filter from Coverage **7-6**

Create, Edit, and View Coverage Filter Rules **7-7**
Create and Edit Coverage Filter Rules **7-7**
Save Coverage Filter to File **7-10**
Load Coverage Filter File **7-11**
Update the Report with the Current Filter Settings **7-11**
View Coverage Filter Rules in Your Model **7-11**
View Coverage Filter Rules in Your Model **7-12**

Coverage Filter Viewer **7-13**

8 Automating Model Coverage Tasks

Commands for Automating Model Coverage Tasks **8-2**

Create Tests with cvtest **8-3**

Run Tests with cvsim **8-5**

Retrieve Coverage Details from Results **8-7**

. . . . . **8-7**

Create HTML Reports with cvhtml **8-8**

Save Test Runs to File with cvsave **8-9**

Load Stored Coverage Test Results with cvload **8-10**
cvload Special Considerations **8-10**

Use Coverage Commands in a Script **8-11**

9 Component Verification

Component Verification **9-2**
Simulink Coverage Tools for Component Verification **9-2**
Workflow for Component Verification **9-3**
Verify a Component Independently of the Container Model . . . **9-4**
Verify a Model Block in the Context of the Container Model . . **9-5**

Verify Generated Code for a Component **9-7**

10 Verification and Validation

Test Model Against Requirements and Report Results **10-2**
Requirements Overview **10-2**
Test a Cruise Control Safety Requirement **10-2**

Analyze a Model for Standards Compliance and Design Errors **10-6**
Standards and Analysis Overview **10-6**
Check Model for Style Guideline Violations and Design Errors **10-6**

Perform Functional Testing and Analyze Test Coverage **10-9**
Functional Testing and Coverage Analysis Overview **10-9**
Incrementally Increase Test Coverage Using Test Case Generation **10-9**

Analyze Code and Test Software-in-the-Loop **10-13**
Code Analysis and Testing Software-in-the-Loop Overview . **10-13**
Analyze Code for Defects, Metrics, and MISRA C:2012 . . . . **10-13**

Module Verification and Testing Processor-in-the-Loop . . . **10-22**
Module Verification and Testing Processor-in-the-Loop Overview **10-22**

Test a Model in Real Time **10-23**
Real-Time Testing and Testing Production Models
Overview **10-23**

Model Coverage Definition

- “Model Coverage” on page 1-2
- “Types of Model Coverage” on page 1-3
- “Simulink Optimizations and Model Coverage” on page 1-11

Model Coverage

Model coverage helps you validate your model tests by measuring how thoroughly the model objects are tested. Model coverage calculates how much a model test case exercises simulation pathways through a model. It is a measure of how thoroughly a test case tests a model and the percentage of pathways that a test case exercises.

Model coverage analyzes the execution of the following types of model objects that directly or indirectly determine simulation pathways through your model:

- Simulink® blocks
- Models referenced in Model blocks
- The states and transitions of Stateflow® charts

During a simulation run, the tool records the behavior of the covered objects, states, and transitions. At the end of the simulation, the tool reports the extent to which the run exercised potential simulation pathways through each covered object in the model.

The Simulink Coverage™ software can only collect model coverage for a model if its simulation mode is set to `Normal`, `SIL`, or `PIL`. If the simulation mode is set to any other mode, model coverage is not measured during simulation.

For the types of coverage that model coverage performs, see “Types of Model Coverage” on page 1-3. For an example of a model coverage report, see “Top-Level Model Coverage Report” on page 6-12.

If you have an Embedded Coder® license, you can also measure code coverage for code generated from models in software-in-the-loop (SIL) mode or processor-in-the-loop (PIL) mode. For the types of coverage that code coverage performs, see “Types of Code Coverage” on page 4-2. For an example of how to enable code coverage, see “Code Coverage for Models in Software-in-the-Loop (SIL) Mode and Processor-in-the-Loop (PIL) Mode” on page 4-7

Types of Model Coverage

Simulink Coverage can perform several types of coverage analysis.

In this section...
“Execution Coverage (EC)” on page 1-3
“Decision Coverage (DC)” on page 1-3
“Condition Coverage (CC)” on page 1-3
“Modified Condition/Decision Coverage (MCDC)” on page 1-4
“Cyclomatic Complexity” on page 1-5
“Lookup Table Coverage” on page 1-5
“Signal Range Coverage” on page 1-6
“Signal Size Coverage” on page 1-6
“Objectives and Constraints Coverage” on page 1-7
“Saturate on Integer Overflow Coverage” on page 1-8
“Relational Boundary Coverage” on page 1-8

Execution Coverage (EC)

Execution coverage is the most basic form of coverage. For each item, execution coverage determines whether the item is executed during simulation.

Decision Coverage (DC)

Decision coverage analyzes elements that represent decision points in a model, such as a Switch block or Stateflow states. For each item, decision coverage determines the percentage of the total number of simulation paths through the item that the simulation traversed.

For an example of decision coverage data in a model coverage report, see “Decisions Analyzed” on page 6-25.

Condition Coverage (CC)

Condition coverage analyzes blocks that output the logical combination of their inputs (for example, the Logical Operator block) and Stateflow transitions. A test case achieves

full coverage when it causes each input to each instance of a logic block in the model and each condition on a transition to be true at least once during the simulation, and false at least once during the simulation. Condition coverage analysis reports whether the test case fully covered the block for each block in the model.

When you collect coverage for a model, you may not be able to achieve 100% condition coverage. For example, if you specify to short-circuit logic blocks, by selecting **Treat Simulink Logic blocks as short-circuited** in the **Coverage** pane of the Configuration Parameters, you might not be able to achieve 100% condition coverage for that block. See “MCDC Analysis” on page 6-27 for more information.

For an example of condition coverage data in a model coverage report, see “Conditions Analyzed” on page 6-27.

Modified Condition/Decision Coverage (MCDC)

Modified condition/decision coverage analysis by the Simulink Coverage software extends the decision and condition coverage capabilities. It analyzes blocks that output the logical combination of their inputs and Stateflow transitions to determine the extent to which the test case tests the independence of logical block inputs and transition conditions.

- A test case achieves full coverage for a block when a change in one input, independent of any other inputs, causes a change in the block output.
- A test case achieves full coverage for a Stateflow transition when there is at least one time when a change in the condition triggers the transition for each condition.

If your model contains blocks that define expressions that have different types of logical operators and more than 12 conditions, the software cannot record MCDC coverage.

Because the Simulink Coverage MCDC coverage may not achieve full decision or condition coverage, you can achieve 100% MCDC coverage *without* achieving 100% decision coverage.

Some Simulink objects support MCDC coverage, some objects support only condition coverage, and some objects support only decision coverage. The table in “Model Objects That Receive Coverage” on page 2-2 lists which objects receive which types of model coverage. For example, the Combinatorial Logic block can receive decision coverage and condition coverage, but not MCDC coverage.

To achieve 100% MCDC coverage for your model, as defined by the DO-178C/DO-331 standard, in the **Coverage** pane of the Configuration Parameters, select "Modified Condition/Decision Coverage (MCDC)" on page 1-4 as the **Structural coverage level**.

When you collect coverage for a model, you may not be able to achieve 100% MCDC coverage. For example, if you specify to short-circuit logic blocks, you may not be able to achieve 100% MCDC coverage for that block.

If you run the test cases independently and accumulate all the coverage results, you can determine if your model adheres to the modified condition and decision coverage standard. For more information about the DO-178C/DO-331 standard, see "DO-178C/DO-331 Checks" (Simulink Check).

For an example of MCDC coverage data in a model coverage report, see "MCDC Analysis" on page 6-27. For an example of accumulated coverage results, see "Cumulative Coverage" on page 6-29.

Cyclomatic Complexity

Cyclomatic complexity is a measure of the structural complexity of a model. It approximates the McCabe complexity measure for code generated from the model. The McCabe complexity measure is slightly higher on the generated code due to error checks that the model coverage analysis does not consider.

To compute the cyclomatic complexity of an object (such as a block, chart, or state), model coverage uses the following formula:

$$c = \sum_{1}^{N} (o_n - 1)$$

N is the number of decision points that the object represents and o_n is the number of outcomes for the nth decision point. The tool adds 1 to the complexity number for atomic subsystems and Stateflow charts.

For an example of cyclomatic complexity data in a model coverage report, see "Cyclomatic Complexity" on page 6-23.

Lookup Table Coverage

Lookup table coverage (LUT) examines blocks, such as the 1-D Lookup Table block, that output information from inputs in a table of inputs and outputs, interpolating between or

extrapolating from table entries. Lookup table coverage records the frequency that table lookups use each interpolation interval. A test case achieves full coverage when it executes each interpolation and extrapolation interval at least once. For each lookup table block in the model, the coverage report displays a colored map of the lookup table, indicating each interpolation. If the total number of breakpoints of an n-D Lookup Table block exceeds 1,500,000, the software cannot record coverage for that block.

For an example of lookup table coverage data in a model coverage report, see “N-Dimensional Lookup Table” on page 6-31.

Note Configure lookup table coverage only at the start of a simulation. If you tune a parameter that affects lookup table coverage at run time, the coverage settings for the affected block are not updated.

Signal Range Coverage

Signal range coverage records the minimum and maximum signal values at each block in the model, as measured during simulation. Only blocks with output signals receive signal range coverage.

The software does not record signal range coverage for control signals, signals used by one block to initiate execution of another block. See “Control Signals” (Simulink).

If the total number of signals in your model exceeds 65535, or your model contains a signal whose width exceeds 65535, the software cannot record signal range coverage.

For an example of signal range coverage data in a model coverage report, see “Signal Range Analysis” on page 6-43.

Note When you create cumulative coverage for reusable subsystems or Stateflow constructs with single range coverage, the cumulative coverage has the largest possible range of signal values. For more information, see “” on page 8-7.

Signal Size Coverage

Signal size coverage records the minimum, maximum, and allocated size for all variable-size signals in a model. Only blocks with variable-size output signals are included in the report.

If the total number of signals in your model exceeds 65535, or your model contains a signal whose width exceeds 65535, the software cannot record signal size coverage.

For an example of signal size coverage data in a model coverage report, see “Signal Size Coverage for Variable-Dimension Signals” on page 6-45.

For more information about variable-size signals, see “Variable-Size Signal Basics” (Simulink).

Objectives and Constraints Coverage

The Simulink Coverage software collects model coverage data for the following Simulink Design Verifier™ blocks and MATLAB® for code generation functions:

Simulink Design Verifier blocks	MATLAB for code generation functions
Test Condition	`sldv.condition`
Test Objective	`sldv.test`
Proof Assumption	`sldv.assume`
Proof Objective	`sldv.prove`

If you do not have a Simulink Design Verifier license, you can collect model coverage for a model that contains these blocks or functions, but you cannot analyze the model using the Simulink Design Verifier software.

By adding one or more Simulink Design Verifier blocks or functions into your model, you can:

- Check the results of a Simulink Design Verifier analysis, run generated test cases, and use the blocks to observe the results.
- Define model requirements using the Test Objective block and verify the results with model coverage data that the software collected during simulation.
- Analyze the model, create a test harness, and simulate the harness with the Test Objective block to collect model coverage data.
- Analyze the model and use the Proof Assumption block to verify any counterexamples that the Simulink Design Verifier identifies.

If you specify to collect Simulink Design Verifier coverage:

- The software collects coverage for the Simulink Design Verifier blocks and functions.
- The software checks the data type of the signal that links to each Simulink Design Verifier block. If the signal data type is fixed point, the block parameter must also be fixed point. If the signal data type is not fixed point, the software tries to convert the block parameter data type. If the software cannot convert the block parameter data type, the software reports an error and you must explicitly assign the block parameter data type to match the signal.
- If your model contains a Verification Subsystem block, the software only records coverage for Simulink Design Verifier blocks in the Verification Subsystem block; it does not record coverage for any other blocks in the Verification Subsystem.

If you do not specify to collect Simulink Design Verifier coverage, the software does not check the data types for any Simulink Design Verifier blocks and functions in your model and does not collect coverage.

For an example of coverage data for Simulink Design Verifier blocks or functions in a model coverage report, see “Simulink Design Verifier Coverage” on page 6-46.

Saturate on Integer Overflow Coverage

Saturate on integer overflow coverage examines blocks, such as the Abs block, with the **Saturate on integer overflow** parameter selected. Only blocks with this parameter selected receive saturate on integer overflow coverage.

Saturate on integer overflow coverage records the number of times the block saturates on integer overflow.

A test case achieves full coverage when the blocks saturate on integer overflow at least once and does not saturate at least once.

For an example of saturate on integer overflow coverage data in a model coverage report, see “Saturate on Integer Overflow Analysis” on page 6-42.

Relational Boundary Coverage

Relational boundary coverage examines blocks, Stateflow charts, and MATLAB function blocks that have an explicit or implicit relational operation.

- Blocks such as Relational Operator and If have an explicit relational operation.

- Blocks such as Abs and Saturation have an implicit relational operation.

For these model objects, the metric records whether a simulation tests the relational operation with:

- Equal operand values.

  This part of relational boundary coverage applies only if both operands are integers or fixed-point numbers.

- Operand values that differ by a certain tolerance.

  This part of relational boundary coverage applies to all operands. For integer and fixed-point operands, the tolerance is fixed. For floating-point operands, you can either use a predefined tolerance or you can specify your own tolerance.

The tolerance value depends on the data type of both the operands. If both operands have the same type, the tolerance follows the following rules:

Data Type of Operand	**Tolerance**
Floating point, such as `single` or `double`	`max(absTol, relTol* max(|lhs|,|rhs|))` • `absTol` is an absolute tolerance value you specify. Default is `1e-05`. • `relTol` is a relative tolerance value you specify. Default is `0.01`. • `lhs` is the left operand and `rhs` the right operand. • `max(x,y)` returns `x` or `y`, whichever is greater.
Fixed point	Value corresponding to least significant bit. For more information, see “Precision” (Fixed-Point Designer). To find the precision value, use the `lsb` function.
Integer	1
Boolean	N/A
Enum	N/A

If the two operands have different types, the tolerance follows the rules for the stricter type. If one of the operands is boolean, the tolerance follows the rules for the other operand. The strictness decreases in this order:

1. Floating point
2. Fixed point
3. Integer

If both operands are fixed point but have different precision, the smaller value of precision is used as tolerance.

You specify the value of absolute and relative tolerances for relational boundary coverage of floating point inputs when you select this metric in the **Coverage metrics** section in the“Coverage Pane” on page 3-2 of the Configuration Parameters dialog box.

For more information on:

- How this coverage metric appears in reports, see “Relational Boundary” on page 6-38.
- Which model objects receive this coverage, see the table in “Model Objects That Receive Coverage” on page 2-2.
- How to obtain coverage results from the MATLAB command-line, see “Collect Relational Boundary Coverage for Supported Block in Model”.

Simulink Optimizations and Model Coverage

In the Configuration Parameters dialog box, there are three Simulink optimization parameters that can affect your model coverage data:

In this section...
“Inlined parameters” on page 1-11
“Block reduction” on page 1-11
“Conditional input branch execution” on page 1-12

Inlined parameters

To transform tunable model parameters into constant values for code generation, in the Configuration Parameters dialog box, on the **Math and Data Types** pane, set **Default parameter behavior** to `Inlined`.

When the parameters are transformed into constants, Simulink may eliminate certain decisions in your model. You cannot achieve coverage for eliminated decision, so the coverage report displays `0/0` for those decisions.

Block reduction

To achieve faster execution during model simulation and in generated code, in the Configuration Parameters dialog box, select the **Block reduction** parameter. The Simulink software collapses certain groups of blocks into a single, more efficient block, or removes them entirely.

One of the model coverage options, **Force block reduction off**, allows you to ignore the **Block reduction** parameter when collecting model coverage.

If you do not select the **Block reduction** parameter, or if you select **Force block reduction off**, the Simulink Coverage software provides coverage data for every block in the model that collects coverage.

If you select the **Block reduction** parameter and do not set **Force block reduction off**, the coverage report lists the reduced blocks that would have collected coverage.

Conditional input branch execution

To improve model execution when the model contains Switch and Multiport Switch blocks, in the Configuration Parameters dialog box, select **Conditional input branch execution**. If you select this parameter, the simulation executes only blocks that are required to compute the control input and the data input selected by the control input.

When Conditional input branch execution is enabled, instead of executing all blocks driving the Switch block input ports at each time step, only the blocks required to compute the control input and the data input selected by the control input execute.

Several considerations affect or limit Switch block optimization:

- Only blocks with `-1` (inherited) or `inf` (Constant) sample time can be optimized.
- Blocks with outputs flagged as test points cannot be optimized.
- Multirate blocks cannot be optimized.
- Blocks with states cannot be optimized.
- Only S-functions with the `SS_OPTION_CAN_BE_CALLED_CONDITIONALLY` option enabled can be optimized.

For example, if your model has a Switch block with output flagged as a test point, such as when a Scope block is attached, that Switch block is not executed, and the model coverage data is incomplete. If you have a model with Switch blocks and you want to verify that the model coverage data is complete, clear **Conditional input branch execution**.

Conditional input branch execution does not apply to Stateflow charts.

2

Model Objects That Receive Model Coverage

Model Objects That Receive Coverage

Certain Simulink objects can receive any type of model coverage. Other Simulink objects can receive only certain types of coverage, as the following table shows. Click a link in the first column to get more detailed information about coverage for specific model objects.

All Simulink objects can receive Execution coverage, except blocks that are not instrumented in model coverage:

- Merge Blocks
- Scope Blocks
- Outport Blocks
- Inport Blocks
- Width Blocks
- Display Blocks

For Stateflow states, events, and state temporal logic decisions, model coverage provides decision coverage. For Stateflow transitions, model coverage provides decision, condition, and MCDC coverage. Model coverage provides condition and MCDC coverage for logical expressions in assignment statements in states and transitions. For more information, see “Model Coverage for Stateflow Charts” on page 5-54.

Model Object	Decision	Condition	MCDC	Lookup Table	Simulink Design Verifier	Saturate on Integer Overflow	Relational Boundary
“Abs” on page 2-8	●					●	●
“Bias” on page 2-9						●	
“Combinatorial Logic” on page 2-9	●	●					
“Compare to Constant” on page 2-9		●					●

Model Object	Decision	Condition	MCDC	Lookup Table	Simulink Design Verifier	Saturate on Integer Overflow	Relational Boundary
“Compare to Zero” on page 2-10		●					●
“Data Type Conversion” on page 2-10						●	
“Dead Zone” on page 2-10	●					●	●
“Direct Lookup Table (n-D)” on page 2-11				●			
“Discrete Filter” on page 2-12						●	
“Discrete FIR Filter” on page 2-12						●	
“Discrete-Time Integrator” on page 2-12 (when saturation limits are enabled or reset)	●					●	
“Discrete Transfer Fcn” on page 2-13						●	
“Dot Product” on page 2-13						●	
“Enabled Subsystem” on page 2-13	●	●	●				

Model Object	Decision	Condition	MCDC	Lookup Table	Simulink Design Verifier	Saturate on Integer Overflow	Relational Boundary
“Enabled and Triggered Subsystem” on page 2-14	●	●	●				
“Fcn” on page 2-15		●	●				●
“For Iterator, For Iterator Subsystem” on page 2-16	●						
“Gain” on page 2-16						●	
“If, If Action Subsystem” on page 2-16	●	●	●				●
“Interpolation Using Prelookup” on page 2-17				●		●	
“Library-Linked Objects” on page 2-18	●	●	●	●	●		
“Logical Operator” on page 2-18		●	●				
“1-D Lookup Table” on page 2-18				●		●	
“2-D Lookup Table” on page 2-19				●		●	

Model Object	Decision	Condition	MCDC	Lookup Table	Simulink Design Verifier	Saturate on Integer Overflow	Relational Boundary
"n-D Lookup Table" on page 2-20				●		●	
"Math Function" on page 2-20						●	
"MATLAB Function" on page 2-20	●	●	●				●
"MATLAB System" on page 2-21	●	●	●				
"MinMax" on page 2-21	●					●	
"Model" on page 2-21 See also "Triggered Models" on page 2-30.	●	●	●	●	●	●	●
"Multiport Switch" on page 2-22	●					●	
"PID Controller, PID Controller (2 DOF)" on page 2-22						●	
"Product" on page 2-23						●	
"Proof Assumption" on page 2-23					●		

Model Object	Decision	Condition	MCDC	Lookup Table	Simulink Design Verifier	Saturate on Integer Overflow	Relational Boundary
"Proof Objective" on page 2-23					●		
"Rate Limiter" on page 2-23	● (Relative to slew rates)						●
"Relational Operator" on page 2-24		●					●
"Relay" on page 2-25	●						●
"C/C++ S-Function" on page 2-25	●	●	●				
"Saturation" on page 2-26	●						●
"Saturation Dynamic" on page 2-27						●	
"Simulink Design Verifier Functions in MATLAB Function Blocks" on page 2-27					●		
Stateflow charts on page 5-54	●	●	●				●
Stateflow state transition tables on page 5-71	●	●	●				●

Model Object	Decision	Condition	MCDC	Lookup Table	Simulink Design Verifier	Saturate on Integer Overflow	Relational Boundary
“Sqrt, Signed Sqrt, Reciprocal Sqrt” on page 2-28						●	
“Sum, Add, Subtract, Sum of Elements” on page 2-28						●	
“Switch” on page 2-28	●					●	●
“SwitchCase, SwitchCase Action Subsystem” on page 2-29	●						
“Test Condition” on page 2-29					●		
“Test Objective” on page 2-29					●		
“Triggered Models” on page 2-30	●	●	●				
“Triggered Subsystem” on page 2-31	●	●	●				
“Truth Table” on page 2-31	●	●	●				
“Unary Minus” on page 2-32						●	

Model Object	Decision	Condition	MCDC	Lookup Table	Simulink Design Verifier	Saturate on Integer Overflow	Relational Boundary
"Weighted Sample Time Math" on page 2-32						•	
"While Iterator, While Iterator Subsystem" on page 2-32	•						

Abs

The Abs block receives decision coverage. Decision coverage is based on:

- Input to the block being less than zero.
- Data type of the input signal.

For input to the block being less than zero, the decision coverage measures:

- The number of time steps that the block input is less than zero, indicating a true decision.
- The number of time steps the block input is not less than zero, indicating a false decision.

If you select the **Saturate on integer overflow** coverage metric, the Abs block receives saturate on integer overflow coverage. For more information, see "Saturate on Integer Overflow Coverage" on page 1-8.

If the input data type to the Abs block is `uint8`, `uint16`, or `uint32`, the Simulink Coverage software reports no coverage for the block. The software sets the block output equal to the block input without making a decision. If the input data type to the Abs block is Boolean, an error occurs.

The Abs block contains an implicit comparison of the input with zero. Therefore, if you select the **Relational Boundary** coverage metric, the Abs block receives relational boundary coverage. For more information, see "Relational Boundary Coverage" on page 1-8.

Bias

If you select the **Saturate on integer overflow** coverage metric, the Bias block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

Combinatorial Logic

The Combinatorial Logic block receives decision and condition coverage. Decision coverage is based on achieving each output row of the truth table. The decision coverage measures the number of time steps that each output row of the truth table is set to the block output.

The condition coverage measures the number of time steps that each input is false (equal to zero) and the number of times each input is true (not equal to zero). If the Combinatorial Logic block has a single input element, the Simulink Coverage software reports only decision coverage, because decision and condition coverage are equivalent.

If all truth table values are set to the block output for at least one time step, decision coverage is 100%. Otherwise, the software reports the coverage as the number of truth table values output during at least one time step, divided by the total number of truth table values. Because this block always has at least one value in the truth table as output, the minimum coverage reported is one divided by the total number of truth table values.

If all block inputs are false for at least one time step and true for at least one time step, condition coverage is 100%. Otherwise, the software reports the coverage as achieving a false value at each input for at least one time step, plus achieving a true value for at least one time step, divided by two raised to the power of the total number of inputs (i.e., 2^*number_of_inputs*). The minimum coverage reported is the total number of inputs divided by two raised to the power of the total number of inputs.

Compare to Constant

The Compare to Constant block receives condition coverage.

Condition coverage measures:

- the number of times that the comparison between the input and the specified constant was true.

- the number of times that the comparison between the input and the specified constant was false.

The Compare to Constant block contains a comparison of the input with a constant. Therefore, if you select the **Relational Boundary** coverage metric, the Compare to Constant block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Compare to Zero

The Compare to Zero block receives condition coverage.

Condition coverage measures:

- the number of times that the comparison between the input and zero was true.
- the number of times that the comparison between the input and zero was false.

The Compare to Zero block contains a comparison of the input with zero. Therefore, if you select the **Relational Boundary** coverage metric, the Compare to Zero block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Data Type Conversion

If you select the **Saturate on integer overflow** coverage metric, the Data Type Conversion block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

Dead Zone

The Dead Zone block receives decision coverage. The Simulink Coverage software reports decision coverage for these parameters:

- **Start of dead zone**
- **End of dead zone**

The **Start of dead zone** parameter specifies the lower limit of the dead zone. For the **Start of dead zone** parameter, decision coverage measures:

- The number of time steps that the block input is greater than or equal to the lower limit, indicating a true decision.
- The number of time steps that the block input is less than the lower limit, indicating a false decision.

The **End of dead zone** parameter specifies the upper limit of the dead zone. For the **End of dead zone**, decision coverage measures:

- The number of time steps that the block input is greater than the upper limit, indicating a true decision.
- The number of time steps that the block input is less than or equal to the upper limit, indicating a false decision.

When the upper limit is true, the software does not measure **Start of dead zone** coverage for that time step. Therefore, the total number of **Start of dead zone** decisions equals the number of time steps that the **End of dead zone** is false.

If you select the **Saturate on integer overflow** coverage metric, the Dead Zone block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

The Dead Zone block contains an implicit comparison of the input with an upper and lower limit value. Therefore, if you select the **Relational Boundary** coverage metric, the Dead Zone block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Direct Lookup Table (n-D)

The Direct Lookup Table (n-D) block receives lookup table coverage. For an *n*-dimensional lookup table, the number of output break points is the product of all the number of break points for each table dimension.

Lookup table coverage measures:

- The number of times during simulation that each combination of dimension input values is between each of the break points.
- The number of times during simulation that each combination of dimension input values is below the lowest break point and above the highest break point for each table dimension.

The total number of coverage points for an *n*-dimensional lookup table is the product of the number of break points in each table dimension plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of at least one time step during simulation between a break point or beyond the end points.

Discrete Filter

If you select the **Saturate on integer overflow** coverage metric, the Discrete Filter block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

Discrete FIR Filter

If you select the **Saturate on integer overflow** coverage metric, the Discrete FIR Filter block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

Discrete-Time Integrator

The Discrete-Time Integrator block receives decision coverage. The Simulink Coverage software reports decision coverage for these parameters:

- **External reset**
- **Limit output**

If you set **External reset** to `none`, the Simulink Coverage software does not report decision coverage for the reset decision. Otherwise, the decision coverage measures:

- The number of time steps that the block output is reset, indicating a true decision.
- The number of time steps that the block output is not reset, indicating a false decision.

If you do not select **Limit output**, the software does not report decision coverage for that decision. Otherwise, the software reports decision coverage for the **Lower saturation limit** and the **Upper saturation limit**.

For the **Upper saturation limit**, decision coverage measures:

- The number of time steps that the integration result is greater than or equal to the upper limit, indicating a true decision.
- The number of time steps that the integration result is less than the upper limit, indicating a false decision.

For the **Lower saturation limit**, decision coverage measures

- The number of time steps that the integration result is less than or equal to the lower limit, indicating a true decision.
- The number of time steps that the integration result is greater than the lower limit, indicating a false decision.

For a time step when the upper limit is true, the software does not measure **Lower saturation limit** coverage. Therefore, the total number of lower limit decisions equals the number of time steps that the upper limit is false.

If you select the **Saturate on integer overflow** coverage metric, the Discrete-Time Integrator block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

Discrete Transfer Fcn

If you select the **Saturate on integer overflow** coverage metric, the Discrete Transfer Fcn block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

Dot Product

If you select the **Saturate on integer overflow** coverage metric, the Dot Product block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

Enabled Subsystem

The Enabled Subsystem block receives decision, condition, and MCDC coverage.

Decision coverage measures:

- The number of time steps that the block is enabled, indicating a true decision.
- The number of time steps that the block is disabled, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

The Simulink Coverage software measures condition coverage for the enable input only if the enable input is a vector. For the enable input, condition coverage measures the number of time steps each element of the enable input is true and the number of time steps each element of the enable input is false. The software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

The software measures MCDC coverage for the enable input only if the enable input is a vector. Because the enable of the subsystem is an `OR` of the vector inputs, MCDC coverage is 100% if, during at least one time step, each vector enable input is exclusively true and if, during at least one time step, all vector enable inputs are false. For MCDC coverage measurement, the software treats each element of the vector as a separate condition.

Enabled and Triggered Subsystem

The Enabled and Triggered Subsystem block receives decision, condition, and MCDC coverage. Decision coverage measures:

- The number of time steps that a trigger edge occurs while the block is enabled, indicating a true decision.
- The number of time steps that a trigger edge does not occur while the block is enabled, or the block is disabled, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

The software measures condition coverage for the enable input and for the trigger input separately:

- For the enable input, condition coverage measures the number of time steps the enable input is true and the number of time steps the enable input is false.
- For the trigger input, condition coverage measures the number of time steps the trigger edge occurs, indicating true, and the number of time steps the trigger edge does not occur, indicating false.

The software reports condition coverage based on the total number of possible conditions and how many conditions are true for at least one time step and how many are false for at least one time step. The software treats each element of a vector as a separate condition coverage measurement.

The software measures MCDC coverage for the enable input and for the trigger input in combination. Because the enable input of the subsystem is an `AND` of these two inputs, MCDC coverage is 100% if all of the following occur:

- During at least one time step, both inputs are true.
- During at least one time step, the enable input is true and the trigger edge is false.
- During one time step, the enable input is false and the trigger edge is true.

The software treats each vector element as a separate MCDC coverage measurement. It measures each trigger edge element against each enable input element. However, if the number of elements in both the trigger and enable inputs exceeds 12, the software does not report MCDC coverage.

Fcn

The Fcn block receives condition and MCDC coverage. The Simulink Coverage software reports condition or MCDC coverage for Fcn blocks only if the top-level operator is Boolean (`&&`, `||`, or `!`).

Condition coverage is based on input values or arithmetic expressions that are inputs to Boolean operators in the block. The condition coverage measures:

- The number of time steps that each input to a Boolean operator is true (not equal to zero).
- The number of time steps that each input to a Boolean operator is false (equal to zero).

If all Boolean operator inputs are false for at least one time step and true for at least one time step, condition coverage is 100%. Otherwise, the software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

The software measures MCDC coverage for Boolean expressions within the Fcn block. If, during at least one time step, each condition independently sets the output of the expression to true and if, during at least one time step, each condition independently sets the output of the expression to false, MCDC coverage is 100%. Otherwise, the software

reports MCDC coverage based on the total number of possible conditions and how many times each condition independently sets the output to true during at least one time step and how many conditions independently set the output to false during at least one time step.

If the Fcn block contains a relational operation and you select the **Relational Boundary** coverage metric, the Fcn block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

For Iterator, For Iterator Subsystem

The For Iterator block and For Iterator Subsystem receive decision coverage. The Simulink Coverage software measures decision coverage for the loop condition value, which is determined by one of the following:

- The iteration value being at or below the iteration limit, indicated as true.
- The iteration value being above the iteration limit, indicated as false.

The software reports the total number of times that each loop condition evaluates to true and to false. If the loop condition evaluates to true at least once and false at least once, decision coverage is 100%. If no loop conditions are true, or if no loop conditions are false, decision coverage is 50%.

Gain

If you select the **Saturate on integer overflow** coverage metric, the Gain block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8.

If, If Action Subsystem

The If block that causes an If Action Subsystem to execute receives condition, decision, and MCDC coverage:

- The software measures decision coverage for the `if` condition and all `elseif` conditions defined in the If block.
- If the `if` condition or any of the `elseif` conditions contains a logical expression with multiple conditions, such as `u1 & u2 & u3`, the software also measures condition and MCDC coverage for each condition in the expression, `u1`, `u2`, and `u3` in the preceding example.

The software does not directly measure the `else` condition. When there are no `elseif` conditions, the `else` condition is the direct complement of the `if` condition, or the `else` condition is the direct complement of the last `elseif` condition.

The software reports the total number of time steps that each `if` and `elseif` condition evaluates to true and to false. If the `if` or `elseif` condition evaluates to true at least once, and evaluates to false at least once, decision coverage is 100%. If no `if` or `elseif` conditions are true, or if no `if` or `elseif` conditions are false, decision coverage is 50%. If the previous `if` or `elseif` condition never evaluates as false, an `elseif` condition can have 0% decision coverage.

The If block contains a comparison between its inputs. Therefore, if you select the **Relational Boundary** coverage metric, the If block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Interpolation Using Prelookup

The Interpolation Using Prelookup block receives lookup table coverage. For an *n*-D lookup table, the number of output break points equals the product of all the number of break points for each table dimension. The lookup table coverage measures:

- The number of times during simulation that each combination of dimension input values is between each of the break points.
- The number of times during simulation that each combination of dimension input values is below the lowest break point and above the highest break point for each table dimension.

The total number of coverage points for an *n*-dimensional lookup table is the product of the number of break points in each table dimension plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of at least one time step during simulation between a break point or beyond the end points.

If you select the **Saturate on integer overflow**, the Interpolation Using Prelookup block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Library-Linked Objects

Simulink blocks and Stateflow charts that are linked to library objects receive the same coverage that they would receive if they were not linked to library objects. The Simulink Coverage software records coverage individually for each library object in the model. If your model contains multiple instances of the same library object, each instance receives its own coverage data.

Logical Operator

The Logical Operator block receives condition and MCDC coverage. The Simulink Coverage software measures condition coverage for each input to the block. The condition coverage measures:

- The number of time steps that each input is true (not equal to zero).
- The number of time steps that each input is false (equal to zero).

If all block inputs are false for at least one time step and true for at least one time step, the software condition coverage is 100%. Otherwise, the software reports the condition coverage based on the total number of possible conditions and how many are true at least one time step and how many are false at least one time step.

The software measures MCDC coverage for all inputs to the block. If, during at least one time step, each condition independently sets the output of the block to true and if, during at least one time step, each condition independently sets the output of the block to false, MCDC coverage is 100%. Otherwise, the software reports the MCDC coverage based on the total number of possible conditions and how many times each one of them independently set the output to true for at least one time step and how many independently set the output to false for at least one time step.

1-D Lookup Table

The 1-D Lookup Table block receives lookup table coverage; for a one-dimensional lookup table, the number of input and output break points is equal. Lookup table coverage measures:

- The number of times during simulation that the input and output values are between each of the break points.
- The number of times during simulation that the input and output values are below the lowest break point and above the highest break point.

The total number of coverage points for a one-dimensional lookup table is the number of break points in the table plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of at least one time step during simulation between a break point or beyond the end points.

If you select the **Saturate on integer overflow** coverage metric, the 1-D Lookup Table block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

2-D Lookup Table

The 2-D Lookup Table block receives lookup table coverage. For a two-dimensional lookup table, the number of output break points equals the number of row break points multiplied by the number of column inputs. Lookup table coverage measures:

- The number of times during simulation that each combination of row input and column input values is between each of the break points.
- The number of times during simulation that each combination of row input and column input values is below the lowest break point and above the highest break point for each row and column.

The total number of coverage points for a two-dimensional lookup table is the number of row break points in the table plus one, multiplied by the number of column break points in the table plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

If you select the **Saturate on integer overflow** coverage metric, the 2-D Lookup Table block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

n-D Lookup Table

The n-D Lookup Table block receives lookup table coverage. For an *n*-dimensional lookup table, the number of output break points equals the product of all the number of break points for each table dimension. Lookup table coverage measures:

- The number of times during simulation that each combination of dimension input values is between each of the break points.
- The number of times during simulation that each combination of dimension output values is below the lowest break point and above the highest break point for each table dimension.

The total number of coverage points for an *n*-dimensional lookup table is the product of the number of break points in each table dimension plus one. In the coverage report, an increasing white-to-green color scale, with six evenly spaced data ranges starting with zero, indicates the number of time steps that the software measures each interpolation or extrapolation point.

The software determines a percentage of total coverage by measuring the total interpolation and extrapolation points that achieve a measurement of at least one time step during simulation between a break point or beyond the end points.

If you select the **Saturate on integer overflow** coverage metric, the n-D Lookup Table block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Math Function

If you select the **Saturate on integer overflow** coverage metric, the Math Function block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

MATLAB Function

For information about the type of coverage that the Simulink Coverage software reports for the MATLAB Function block, see “Model Coverage for MATLAB Functions” on page 5-29.

MATLAB System

Simulink Coverage records only Decision, Condition, and MCDC coverage for MATLAB System blocks.

MinMax

The MinMax block receives decision coverage based on passing each input to the output of the block.

For decision coverage based on passing each input to the output of the block, the coverage measures the number of time steps that the simulation passes each input to the block output. The number of decision points is based on the number of inputs to the block and whether they are scalar, vector, or matrix.

If all inputs are passed to the block output for at least one time step, the Simulink Coverage software reports the decision coverage as 100%. Otherwise, the software reports the coverage as the number of inputs passed to the output during at least one time step, divided by the total number of inputs.

If you select the **Saturate on integer overflow** coverage metric, the MinMax block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Model

The Model block does not receive coverage directly; the model that the block references receives coverage. If the simulation mode for the referenced model is set to `Normal`, the Simulink Coverage software reports coverage for all objects within the referenced model that receive coverage. . If the simulation mode for the referenced model is set to `SIL` or `PIL` and you have Embedded Coder installed, the Simulink Coverage software reports coverage for the code generated from your model .If the simulation mode is set to a value other than `Normal`, `SIL`, or `PIL`, the software cannot measure coverage for the referenced model.

In the **Coverage** pane of the Configuration Parameters dialog box, select the referenced models for which you want to report coverage. The software generates a coverage report for each referenced model you select.

If your model contains multiple instances of the same referenced model, the software records coverage for all instances of that model where the simulation mode of the Model block is set to `Normal`. The coverage report for that referenced model combines the coverage data for all Normal mode instances of that model.

The coverage reports for referenced models are linked from a summary report for the parent model.

Note For details on how to select referenced models to report coverage, see “Referenced Models” on page 3-4.

Multiport Switch

The Multiport Switch block receives decision coverage based on passing each input, excluding the first control input, to the output of the block.

For decision coverage based on passing each input, excluding the first control input, to the output of the block, the coverage measures the number of time steps that each input is passed to the block output. The number of decision points is based on the number of inputs to the block and whether the control input is scalar or vector.

If all inputs, excluding the first control input, are passed to the block output for at least one time step, decision coverage is 100%. Otherwise, the Simulink Coverage software reports coverage as the number of inputs passed to the output during at least one time step, divided by the total number of inputs minus one.

If you select the **Saturate on integer overflow** coverage metric, the Multiport Switch block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

PID Controller, PID Controller (2 DOF)

If you select the **Saturate on integer overflow** coverage metric, the PID Controller and PID Controller (2 DOF) blocks receive saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Product

If you select the **Saturate on integer overflow** coverage metric, the Product block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Proof Assumption

The Proof Assumption block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box. Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Proof Assumption block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Coverage software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

Proof Objective

The Proof Objective block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box. Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Proof Objective block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Coverage software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

Rate Limiter

The Rate Limiter block receives decision coverage. The Simulink Coverage software reports decision coverage for the **Rising slew rate** and **Falling slew rate** parameters.

For the **Rising slew rate**, decision coverage measures:

- The number of time steps that the block input changes more than or equal to the rising rate, indicating a true decision.
- The number of time steps that the block input changes less than the rising rate, indicating a false decision.

For the **Falling slew rate**, decision coverage measures:

- The number of time steps that the block input changes less than or equal to the falling rate, indicating a true decision.
- The number of time steps that the block input changes more than the falling rate, indicating a false decision.

The software does not measure **Falling slew rate** coverage for a time step when the **Rising slew rate** is true. Therefore, the total number of **Falling slew rate** decisions equals the number of time steps that the **Rising slew rate** is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the two individual decisions for the block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

The Rate Limiter block implicitly compares the derivative of the input signal with an upper and lower limit value. Therefore, if you select the **Relational Boundary** coverage metric, the Rate Limiter block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Relational Operator

The Relational Operator block receives condition coverage.

Condition coverage measures:

- the number of times that the specified relational operation was true.
- the number of times that the specified relational operation was false.

The Relational Operator block contains a comparison between its inputs. Therefore, if you select the **Relational Boundary** coverage metric, the Relational Operator block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Relay

The Relay block receives decision coverage. The Simulink Coverage software reports decision coverage for the **Switch on point** and the **Switch off point** parameters.

For the **Switch on point**, decision coverage measures:

- The number of consecutive time steps that the block input is greater than or equal to the **Switch on point**, indicating a true decision.
- The number of consecutive time steps that the block input is less than the **Switch on point**, indicating a false decision.

For the **Switch off point**, decision coverage measures:

- The number of consecutive time steps that the block input is less than or equal to the **Switch off point**, indicating a true decision.
- The number of consecutive time steps that the block input is greater than the **Switch off point**, indicating a false decision.

The software does not measure **Switch off point** coverage for a time step when the switch on threshold is true. Therefore, the total number of **Switch off point** decisions equals the number of time steps that the **Switch on point** is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the two individual decisions for the block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

The Relay block contains an implicit comparison of its second input with a threshold value. Therefore, if you select the **Relational Boundary** coverage metric, the Relay block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

C/C++ S-Function

Model coverage is supported for C/C++ S-Functions. The coverage report for the model contains results for each instance of an S-Function block in the model. The results for an S-Function block link to a separate coverage report for the C/C++ code in the block.

To generate coverage report for S-Functions:

1. When creating the S-Functions, enable support for coverage. For more information, see “Make S-Function Compatible with Model Coverage” on page 5-46.
2. When generating the coverage report, enable support for S-Functions. For more information, see “Generate Coverage Report for S-Function” on page 5-47.

The following coverage types are reported for S-Functions:

- “Cyclomatic Complexity for Code Coverage” on page 4-5
- “Condition Coverage for Code Coverage” on page 4-3
- “Decision Coverage for Code Coverage” on page 4-3
- “Modified Condition/Decision Coverage (MCDC) for Code Coverage” on page 4-4
- “Relational Boundary for Code Coverage” on page 4-5
- Percentage of statements covered

The coverage data for S-Function blocks is obtained in the following way:

- The coverage result for a block is a weighted average of the result over all files in the block.

  For instance, an S-Function block has two files, `file1.c` and `file2.c`. The decision coverage for `file1.c` is 75% (3/4 outcomes covered) and that for `file2.c` is 50% (10/20 outcomes covered). The decision coverage for the block is 13/24 ≈ 54 %.
- For each file, the coverage result is a weighted average of the result over all functions in the file.
- For each function, the coverage result is a weighted average of the result over all statements in the function that receive that coverage.

Note Model coverage for S-Functions have the following restrictions:

- Only Level-2 C/C++ S-Functions are supported for coverage. For an example of a level-2 C S-Function, see “Basic C MEX S-Function” (Simulink).
- C++ class templates are not instrumented for coverage.

Saturation

The Saturation block receives decision coverage. The Simulink Coverage software reports decision coverage for the **Lower limit** and **Upper limit** parameters.

For the **Upper limit**, decision coverage measures:

- The number of time steps that the block input is greater than or equal to the upper limit, indicating a true decision.
- The number of time steps that the block input is less than the upper limit, indicating a false decision.

For the **Lower limit**, decision coverage measures:

- The number of time steps that the block input is greater than the lower limit, indicating a true decision.
- The number of time steps that the block input is less than or equal to the lower limit, indicating a false decision.

The software does not measure **Lower limit** coverage for a time step when the upper limit is true. Therefore, the total number of **Lower limit** decisions equals the number of time steps that the **Upper limit** is false.

If at least one time step is true and at least one time step is false, decision coverage for each of the two individual decisions for the Saturation block is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%. The software treats each element of a vector or matrix as a separate coverage measurement.

The Saturation block contains an implicit comparison of the input with an upper and lower limit value. Therefore, if you select the **Relational Boundary** coverage metric, the Saturation block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Saturation Dynamic

If you select the **Saturate on integer overflow** coverage metric, the Saturation Dynamic block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Simulink Design Verifier Functions in MATLAB Function Blocks

The following functions in MATLAB Function blocks receive Simulink Design Verifier coverage:

- `sldv.condition`
- `sldv.test`
- `sldv.assume`
- `sldv.prove`

Each of these functions evaluates an expression *`expr`*, for example, `sldv.test(`*`expr`*`)`, where *`expr`* is any valid Boolean MATLAB expression. Simulink Design Verifier coverage measures the number of time steps that the expression *`expr`* evaluates to true.

If *`expr`* is true for at least one time step, Simulink Design Verifier coverage for that function is 100%. Otherwise, the Simulink Coverage software reports coverage for that function as 0%.

Sqrt, Signed Sqrt, Reciprocal Sqrt

If you select the **Saturate on integer overflow** coverage metric, the Sqrt, Signed Sqrt, and Reciprocal Sqrt blocks receive saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Sum, Add, Subtract, Sum of Elements

If you select the **Saturate on integer overflow** coverage metric, the Sum, Add, Subtract, and Sum of Elements blocks receive saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Switch

The Switch block receives decision coverage based on the control input to the block. Decision coverage measures:

- The number of time steps that the control input evaluates to true.
- The number of time steps the control input evaluates to false.

The number of decision points is based on whether the control input is scalar or vector.

If you select the **Saturate on integer overflow** coverage metric, the Switch block receives saturate on integer overflow coverage. For more information, see “Saturate on

Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

The Switch block contains an implicit comparison of its second input with a threshold value. Therefore, if you select the **Relational Boundary** coverage metric, the Switch block receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

SwitchCase, SwitchCase Action Subsystem

The SwitchCase block and SwitchCase Action Subsystem receive decision coverage. The Simulink Coverage software measures decision coverage individually for each switch case defined in the block and also for the default case. The number of decision outcomes is equal to the number of case conditions plus one for the `default` case, if one is defined.

The software reports the total number of time steps that each case evaluates to true. If each case, including the default case, evaluates to true at least once, decision coverage is 100%. The software determines the decision coverage by the number of cases that evaluate true for at least one time step divided by the total number of cases.

If the SwitchCase block does not contain a `default` case, the software measures decision coverage for the number of time steps in which none of the cases evaluated to true. In the coverage report, this coverage is reported as **implicit-default**.

Test Condition

The Test Condition block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box. Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Test Condition block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Coverage software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

Test Objective

The Test Objective block receives Simulink Design Verifier coverage. Simulink Design Verifier coverage is based on the points and intervals defined in the block dialog box.

Simulink Design Verifier coverage measures the number of time steps that each point or interval defined in the block is satisfied. The total number of objective outcomes is based on the number of points or intervals defined in the Test Objective block.

If all points and intervals defined in the block are satisfied for at least one time step, Simulink Design Verifier coverage is 100%. Otherwise, the Simulink Coverage software reports coverage as the number of points and intervals satisfied during at least one time step, divided by the total number of points and intervals defined for the block.

Triggered Models

A Model block can reference a model that contains edge-based trigger ports at the root level of the model. Triggered models receive decision, condition, and MCDC coverage.

Decision coverage measures:

- The number of time steps that the referenced model is triggered, indicating a true decision.
- The number of time steps that the referenced model is not triggered, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage for the Model block that references the triggered model is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

Only if the trigger input is a vector, the Simulink Coverage software measures condition coverage for the trigger port in the referenced model. For the trigger port, condition coverage measures:

- The number of time steps that each element of the trigger port is true.
- The number of time steps that each element of the trigger port is false.

The software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

If the trigger port is a vector, the software measures MCDC coverage for the trigger port only. Because the trigger port of the referenced model is an `OR` of the vector inputs, if, during at least one time step, each vector trigger port is exclusively true and if, during at least one time step, all vector trigger port inputs are false, MCDC coverage is 100%. The

software treats each element of the vector as a separate condition for MCDC coverage measurement.

Triggered Subsystem

The Triggered Subsystem block receives decision, condition, and MCDC coverage.

Decision coverage measures:

- The number of time steps that the block is triggered, indicating a true decision.
- The number of time steps that the block is not triggered, indicating a false decision.

If at least one time step is true and at least one time step is false, decision coverage is 100%. If no time steps are true, or if no time steps are false, decision coverage is 50%.

The Simulink Coverage software measures condition coverage for the trigger input only if the trigger input is a vector. For the trigger input, condition coverage measures:

- The number of time steps that each element of the trigger edge is true.
- The number of time steps that each element of the trigger edge is false.

The software reports condition coverage based on the total number of possible conditions and how many are true for at least one time step and how many are false for at least one time step.

If the trigger input is a vector, the software measures MCDC coverage for the trigger input only. Because the trigger edge of the subsystem is an `OR` of the vector inputs, if, during at least one time step, each vector trigger edge input is exclusively true and if, during at least one time step, all vector trigger edge inputs are false, MCDC coverage is 100%. The software treats each element of the vector as a separate condition for MCDC coverage measurement.

Truth Table

The Truth Table block is a Stateflow block that enables you to use truth table logic directly in a Simulink model. The Truth Table block receives condition, decision, and MCDC coverage. For more information on model coverage with Stateflow truth tables, see “Model Coverage for Stateflow Truth Tables” on page 5-75.

Unary Minus

If you select the **Saturate on integer overflow** coverage metric, the Unary Minus block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

Weighted Sample Time Math

If you select the **Saturate on integer overflow** coverage metric, the Weighted Sample Time Math block receives saturate on integer overflow coverage. For more information, see “Saturate on Integer Overflow Coverage” on page 1-8. The software treats each element of a vector or matrix as a separate coverage measurement.

While Iterator, While Iterator Subsystem

The While Iterator block and While Iterator Subsystem receive decision coverage. Decision coverage is measured for the `while` condition value, which is determined by the `while` condition being satisfied (true), or the `while` condition not being satisfied (false). Simulink Coverage software reports the total number of times that each `while` condition evaluates to true and to false. If the `while` condition evaluates to true at least once, and false at least once, decision coverage for the `while` condition is 100%. If no `while` conditions are true, or if no `while` conditions are false, decision coverage is 50%.

If the iteration limit is exceeded (true) or is not exceeded (false), the software measures decision coverage independently. If the iteration limit evaluates to true at least once, and false at least once, decision coverage for the iteration limit is 100%. If no iteration limits are true, or if no iteration limits are false, decision coverage is 50%. If you set **Maximum number of iterations** to `-1` (no limit), the decision coverage for the iteration limit is true for all iterations and false for zero iterations, and decision coverage is 50%.

Model Objects That Do Not Receive Coverage

The Simulink Coverage software does not record Decision, Condition, or MCDC coverage for blocks that are not listed in “Model Objects That Receive Coverage” on page 2-2.

Note The software only records model coverage when the **Simulation mode** parameter is set to `Normal`. If you have Embedded Coder installed, the software can measure the coverage of code generated from models in SIL or PIL mode. For more information, see“Code Coverage for Models in Software-in-the-Loop (SIL) Mode and Processor-in-the-Loop (PIL) Mode” on page 4-7.

The following table identifies specific model objects that do not receive coverage in certain conditions.

Model object	Does not receive coverage...
Logical Operator block	When the **Operator** parameter specifies `XOR` or `NXOR` and there are more than twelve scalar inputs or more than twelve elements in a vector input.
Model block	When the **Simulation mode** parameter specifies `Accelerator`. Coverage for Model blocks is the sum of the coverage data for the contents of the referenced model.
Subsystem block	When the **Read/Write Permissions** parameter is set to `NoReadOrWrite`.
Stateflow chart MATLAB Function block	When debugging/animation is not enabled for the model or object.
Virtual Blocks	doc Virtual blocks do not receive model coverage. For more information, see “Nonvirtual and Virtual Blocks” (Simulink).

Setting Coverage Options

- “Specify Coverage Options” on page 3-2
- “Access, Manage, and Accumulate Coverage Results” on page 3-10
- “Cumulative Coverage Data” on page 3-21

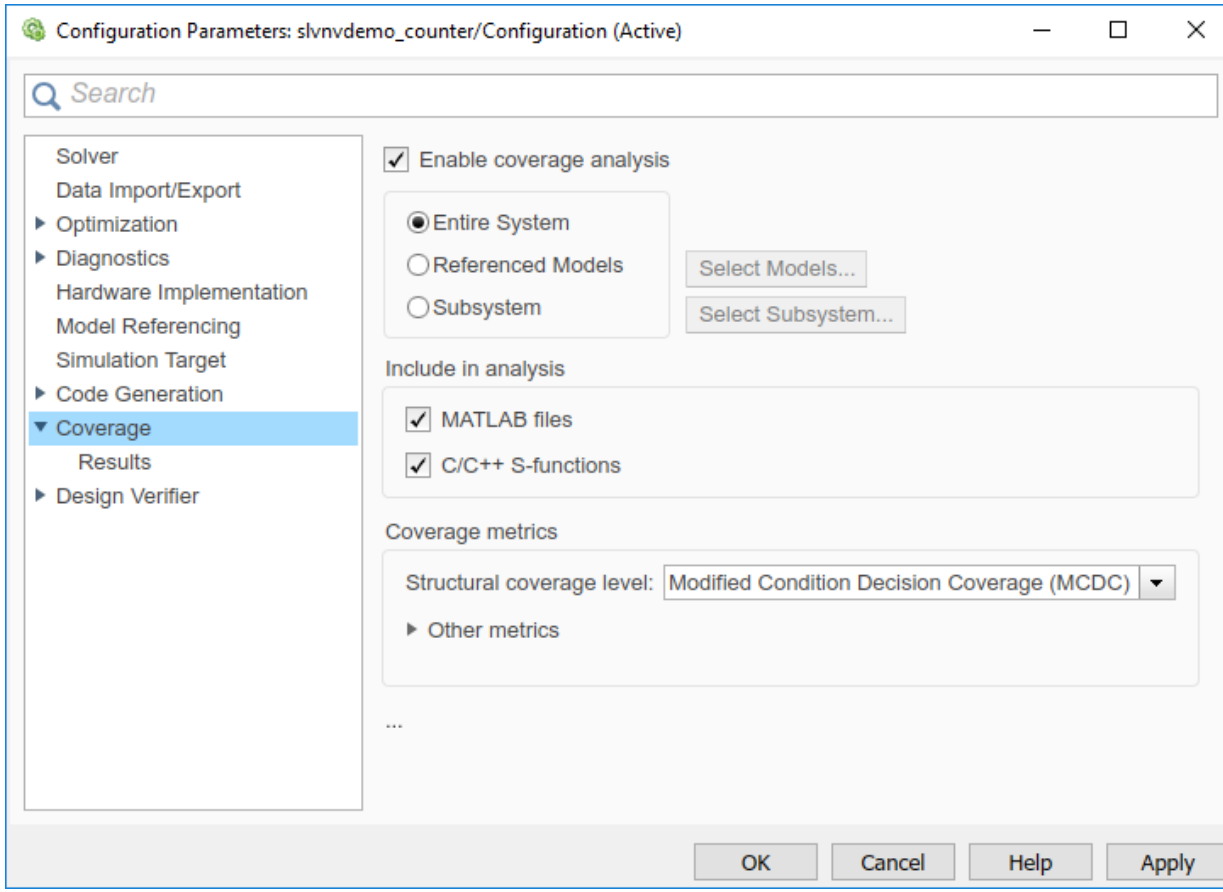
Specify Coverage Options

Before starting a coverage analysis, you specify several coverage recording options. In the Simulink Editor, select **Analysis > Coverage > Settings**.

In this section...
“Coverage Pane” on page 3-2
“Results Pane” on page 3-7

Coverage Pane

On the **Coverage** pane in the Configuration Parameters dialog box, set the options for the coverage calculated during simulation.



Enable coverage analysis

Gather specified coverage results during simulation and report the coverage. When you select **Enable coverage analysis**, these sections become available:

- “Scope of analysis” on page 3-4
- “Include in analysis” on page 3-6
- “Coverage metrics” on page 3-7

Scope of analysis

Specifies the systems for which the software gathers and reports coverage data. The options are:

- “Entire System” on page 3-4
- “Referenced Models” on page 3-4
- “Subsystem” on page 3-5

You must select **Enable coverage analysis** to specify the scope of analysis.

Entire System

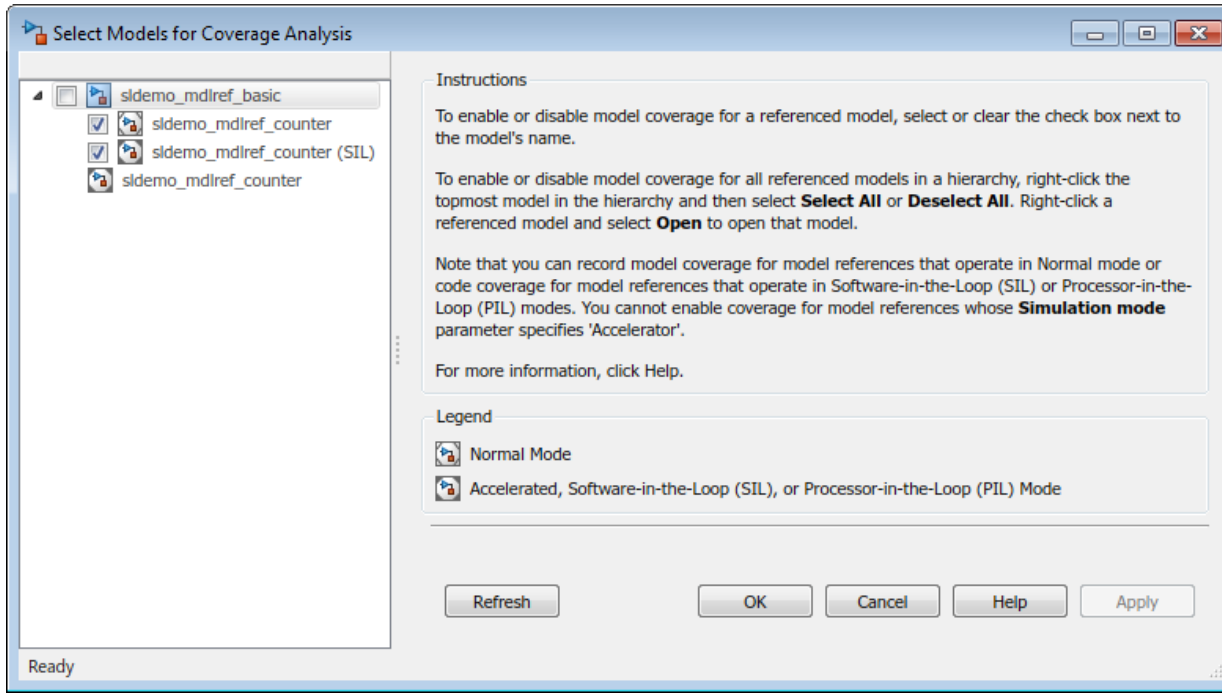
By default, generates coverage data for the entire system. The coverage results include the top-level and all supported subsystems and model references.

Referenced Models

Coverage analysis records the coverages for the referenced models that you select. By default, generates coverage data for all referenced models where the simulation mode of the Model block is set to `Normal`, `Software-in-the-loop (SIL)`, or `Processor-in-the-loop (PIL)`.

To specify the referenced models for which the Simulink Coverage software records coverage data:

1. In the Configuration Parameters dialog box, on the **Coverage** pane, select **Enable coverage analysis**.
2. Click **Select Models**.



3 In the Select Models for Coverage Analysis dialog box, select the referenced models for which you want to record coverage. You can also select the top-level model.

   The icon next to the model name indicates the simulation mode for that referenced model. You can select only referenced models whose simulation mode is set to `Normal`, `SIL`, or `PIL`.

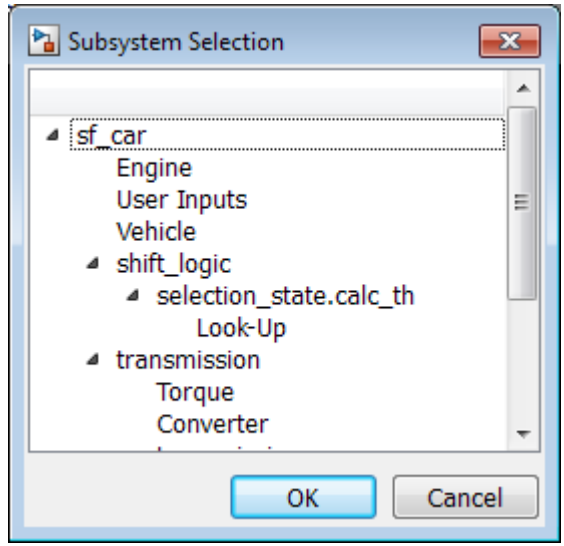
   If you have multiple Model blocks that reference the same model and whose simulation modes are the same, selecting the check box for that model selects the check boxes for all instances of that model with the same simulation mode.

4 To close the Select Models for Coverage Analysis dialog box and return to the Configuration Parameters dialog box, click **OK**.

Subsystem

Coverage analysis records coverage during simulation for the subsystem that you select. By default, generates coverage data for the entire model. To restrict coverage reporting to a particular subsystem:

1. In the Configuration Parameters dialog box, on the **Coverage** pane, select **Enable coverage analysis**.
2. Click **Select Subsystem**.



3. In the Subsystem Selection dialog box, select the subsystem for which you want to enable coverage reporting and click **OK**.

Include in analysis

The **Include in analysis** section contains two options:

- **MATLAB files** enables coverage for any external functions called by MATLAB functions in your model. You can define MATLAB functions in MATLAB Function blocks or in Stateflow charts.

  To select the **Coverage for MATLAB files** option, you must select **Enable coverage analysis**.
- **C/C++ S-functions** enables coverage for C/C++ S-Function blocks in your model. Coverage metrics are reported for the S-Function blocks and the C/C++ code in those blocks. For more information, see “Generate Coverage Report for S-Function” on page 5-47.

  You must select **Enable coverage analysis** to select the **Coverage for S-Functions** option.

Coverage metrics

Select the structural coverage level and other types of test case coverage analysis that you want the tool to perform (see “Types of Model Coverage” on page 1-3). The Simulink Coverage software gathers and reports those types of coverage for the subsystems, models, and referenced models that you specify.

The structural coverage levels are listed in order of strictness of test case coverage analysis:

- `Block Execution` — Enables “Execution Coverage (EC)” on page 1-3
- `Decision` — Enables “Execution Coverage (EC)” on page 1-3 and “Decision Coverage (DC)” on page 1-3
- `Condition Decision` — Enables “Execution Coverage (EC)” on page 1-3, “Decision Coverage (DC)” on page 1-3, and “Condition Coverage (CC)” on page 1-3
- `Modified Condition Decision Coverage (MCDC)` — enables “Execution Coverage (EC)” on page 1-3, “Decision Coverage (DC)” on page 1-3, “Condition Coverage (CC)” on page 1-3, and “Modified Condition/Decision Coverage (MCDC)” on page 1-4

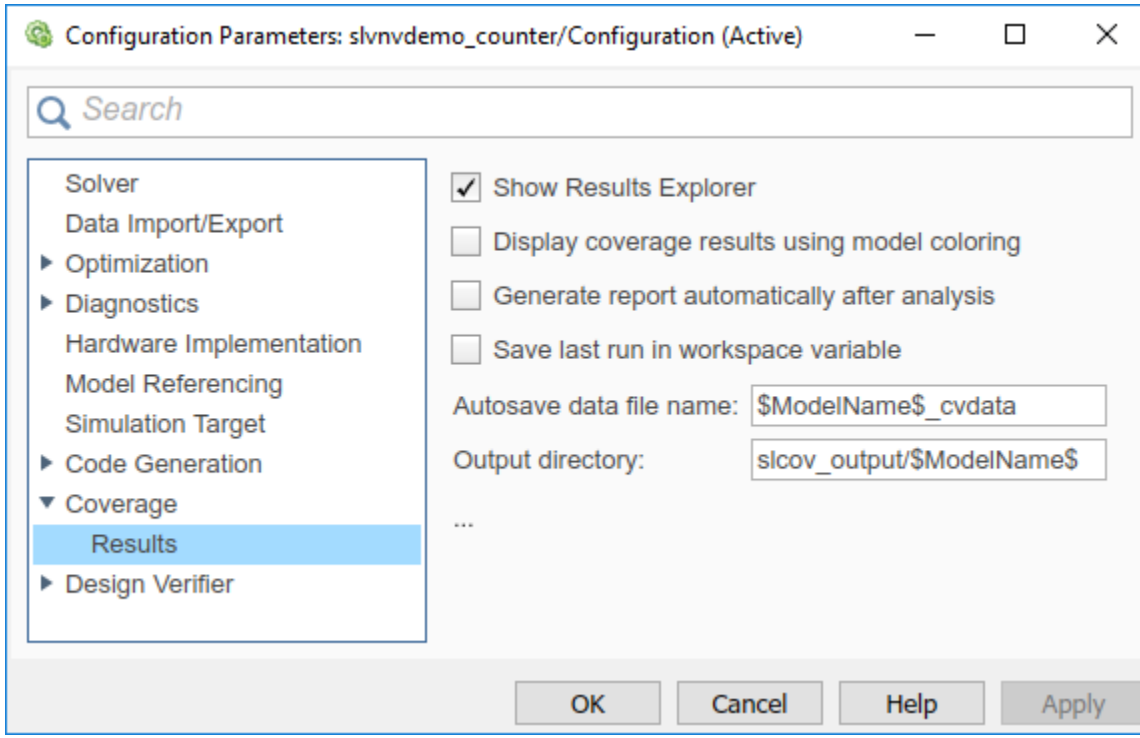
Coverage metrics also includes **Other metrics**:

- “Lookup Table Coverage” on page 1-5
- “Signal Range Coverage” on page 1-6
- “Signal Size Coverage” on page 1-6
- “Objectives and Constraints Coverage” on page 1-7
- “Saturate on Integer Overflow Coverage” on page 1-8
- “Relational Boundary Coverage” on page 1-8

You must select **Enable coverage analysis** to select the coverage metrics.

Results Pane

On the **Coverage > Results** pane in the Configuration Parameters dialog box, select the destination for coverage results. You must select **Enable coverage analysis** on the **Coverage** pane to set the **Coverage > Results** pane options.



Show Results Explorer

After simulation, shows the results explorer.

Display coverage results using model coloring

After simulation, colors model objects according to their level of coverage. Objects highlighted in light green receive full coverage during testing. Objects highlighted in light red receive incomplete coverage. See “View Coverage Results in a Model” on page 5-12.

Generate report automatically after analysis

Specifies whether to open a generated HTML coverage report in a MATLAB browser window at the end of model simulation.

Save last run in workspace variable

Saves the results of the last simulation run in a `cvdata` object in the workspace. Specify the workspace variable name in **cvdata object name**.

cvdata object name

Name of the workspace variable where the results of the last simulation run are saved. You must select **Save last run in workspace variable** to specify the `cvdata` object name.

Increment variable name with each simulation (var1, var2, ...)

Appends numerals to the workspace variable names for each new result so that earlier results are not overwritten. You must select **Save last run in workspace variable** to enable this option.

Autosave data file name

Name of file to which coverage data results are saved. The default name is `$ModelName$_cvdata`. `$ModelName$` is the name of the model.

Output directory

The folder where the coverage data is saved. The default location is `slcov_output/$ModelName$` in the current folder. `$ModelName$` is the name of the model.

See Also

Related Examples

- “Access, Manage, and Accumulate Coverage Results” on page 3-10

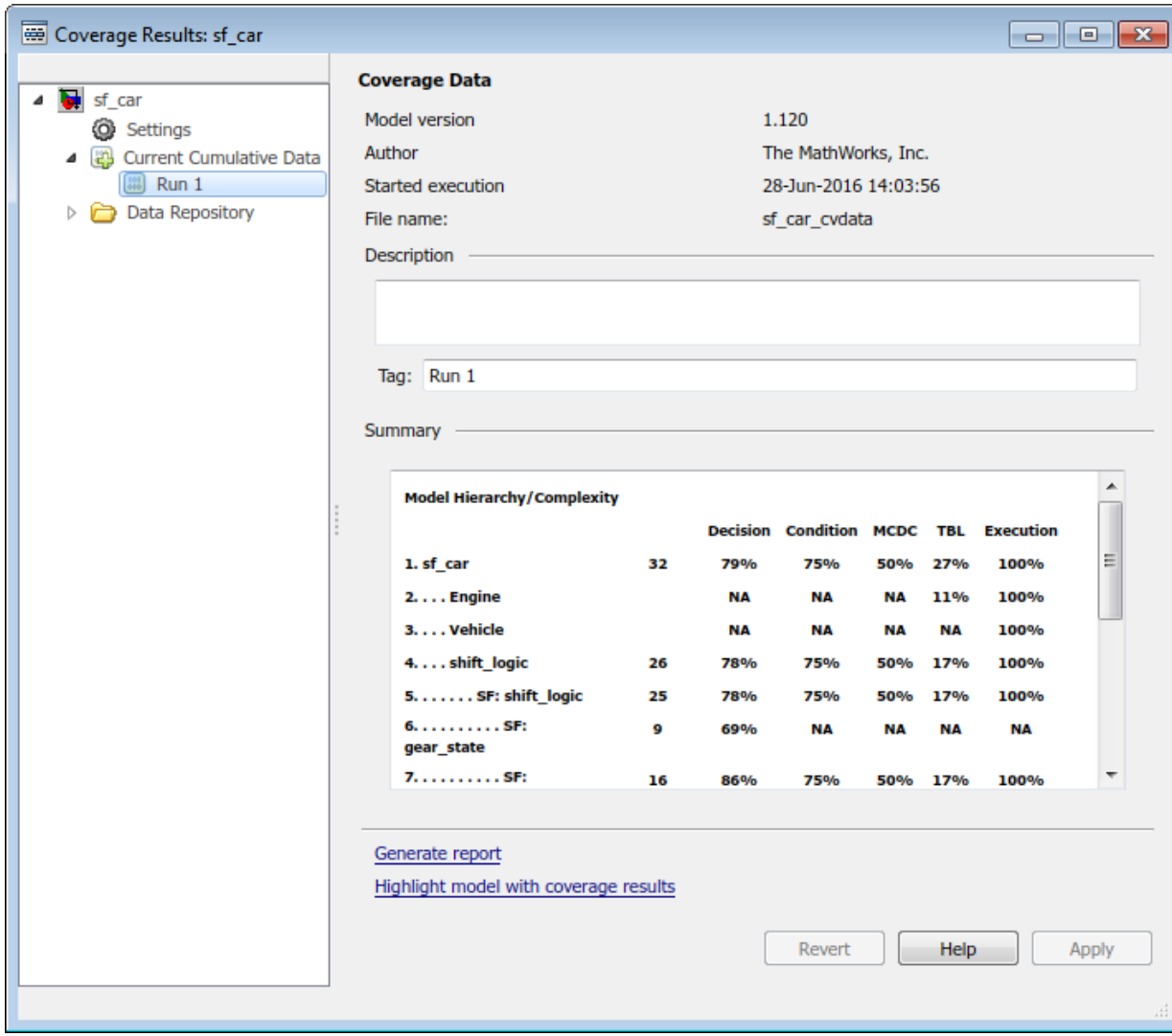
Access, Manage, and Accumulate Coverage Results

After you “Specify Coverage Options” on page 3-2 and record coverage results, you can use the Results Explorer to access, manage, and accumulate the coverage data that you record. After you accumulate the coverage results you need, you can then create a “Top-Level Model Coverage Report” on page 6-12 or “Export Model Coverage Web View” on page 6-48 using your accumulated coverage data.

In this section...
“Accessing Coverage Data from the Results Explorer” on page 3-10
“Managing Coverage Data from the Results Explorer” on page 3-18
“Accumulating Coverage Data from the Results Explorer” on page 3-18

Accessing Coverage Data from the Results Explorer

In the Configuration Parameters dialog box, on the **Coverage > “Results Pane” on page 3-7**, you can specify whether to show the Results Explorer after each simulation. You can also specify whether to generate an HTML report after each simulation. If you do not specify to show the Results Explorer or generate an HTML report, you can access the Results Explorer by selecting **Analysis > Coverage > Open Results Explorer** after you record coverage. The Coverage Results Explorer opens to show the most recent coverage run:

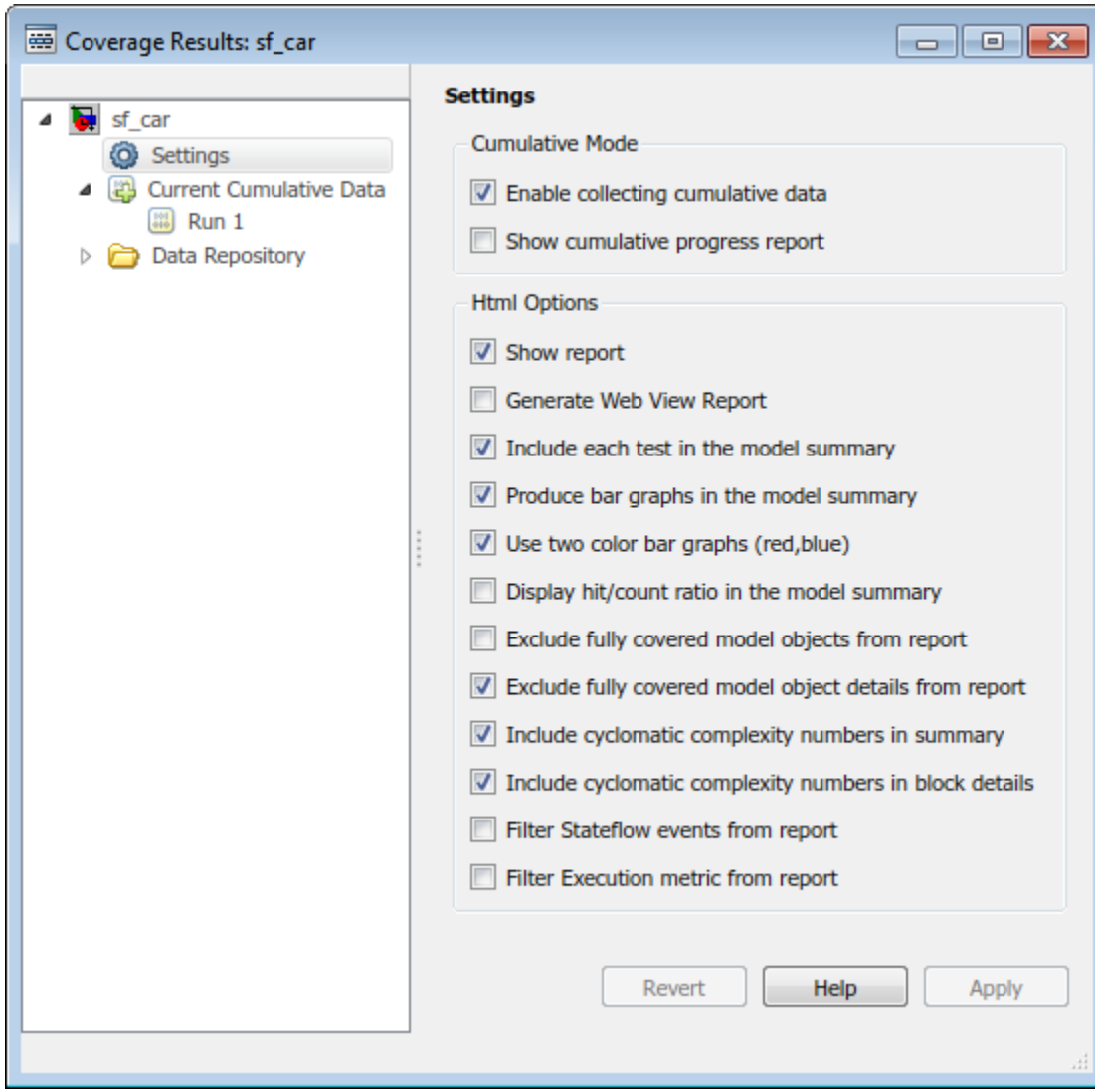


You can view the current data results summary from within the Results Explorer or click **Generate Report** to create a full coverage report. If you do not make any changes to your model after you record coverage, you do not need to resimulate the model to generate a new coverage report. For more information on coverage reports, see “Top-Level Model Coverage Report” on page 6-12.

Click **Highlight model with coverage results** to provide highlighted results in your model that allow you to quickly see coverage results for model objects. For more information, see “Overview of Model Coverage Highlighting” on page 5-12.

Settings

In the coverage Results Explorer, you can access the data and reporting settings for your coverage data. To access these settings, click **Settings**.


Coverage Results: sf_car
sf_car
Settings
Current Cumulative Data
Run 1
Data Repository
Settings
Cumulative Mode
Enable collecting cumulative data
Show cumulative progress report
Html Options
Show report
Generate Web View Report
Include each test in the model summary
Produce bar graphs in the model summary
Use two color bar graphs (red,blue)
Display hit/count ratio in the model summary
Exclude fully covered model objects from report
Exclude fully covered model object details from report
Include cyclomatic complexity numbers in summary
Include cyclomatic complexity numbers in block details
Filter Stateflow events from report
Filter Execution metric from report
Revert
Help
Apply

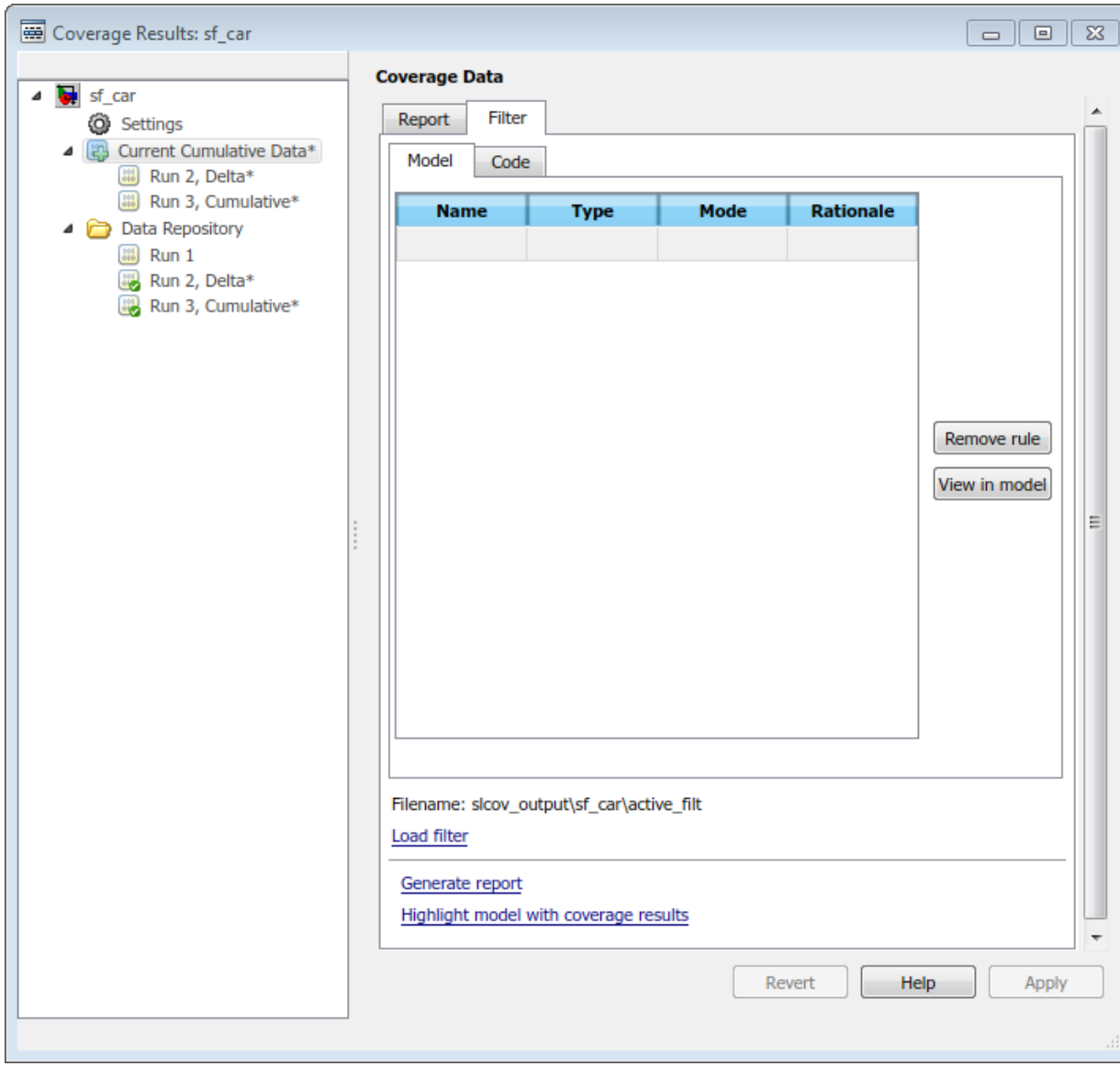
Option	Description
Enable collecting cumulative data	Accumulates coverage results from successive simulations, by default. You specify the name and output folder of the `.cvt` file in the Configuration Parameters dialog box, on the “Results Pane” on page 3-7. For more information, see .“Cumulative Coverage Data” on page 3-21
Show cumulative progress report	Shows the **Current Run** coverage results, the **Delta** of coverage compared to the previous cumulative data, and the total **Cumulative** data from all current cumulative data separately in the coverage reports. If you do not select this option, only the total **Cumulative** data from all current cumulative data are shown.
Show report	Opens a generated HTML coverage report in a MATLAB browser window at the end of model simulation. For more information, see “Top-Level Model Coverage Report” on page 6-12.
Generate Web View Report	Opens a generated Model Coverage Web View in a MATLAB browser window at the end of model simulation. For more information, see “Export Model Coverage Web View” on page 6-48.
Include each test in the model summary	At the top of the HTML report, the model hierarchy table includes columns listing the coverage metrics for each test. If you do not select this option, the model summary reports only the total coverage.
Produce bar graphs in the model summary	Causes the model summary to include a bar graph for each coverage result for a visual representation of the coverage.

Option	Description
Use two color bar graphs (red, blue)	Red and blue bar graphs are displayed in the report instead of black and white bar graphs.
Display hit/count ratio in the model summary	Reports coverage numbers as both a percentage and a ratio, for example, 67% (8/12).
Exclude fully covered model objects from report	The coverage report includes only model objects that the simulation does not cover fully, useful when developing tests, because it reduces the size of the generated reports.
Exclude fully covered model object details from report	If you choose to include fully covered model objects in the report, the report does not include the details of the fully covered model objects
Include cyclomatic complexity numbers in summary	Includes the cyclomatic complexity (see "Types of Model Coverage" on page 1-3) of the model and its top-level subsystems and charts in the report summary. A cyclomatic complexity number shown in boldface indicates that the analysis considered the subsystem itself to be an object when computing its complexity. Boldface text can occur for atomic and conditionally executed subsystems and Stateflow Chart blocks.
Include cyclomatic complexity numbers in block details	Includes the cyclomatic complexity metric in the block details section of the report.
Filter Stateflow events from report	Excludes coverage data on Stateflow events.
Filter Execution metric from report	Excludes coverage data on Execution metrics

Creating and Managing Filters

You can create, load, or edit filters for the current coverage data from within the Results Explorer.

1. Open the Results Explorer.
2. Click the **Current Cumulative Data**.
3. Click the **Filter** tab.



For more information on filtering model objects, see “Creating and Using Coverage Filters”.

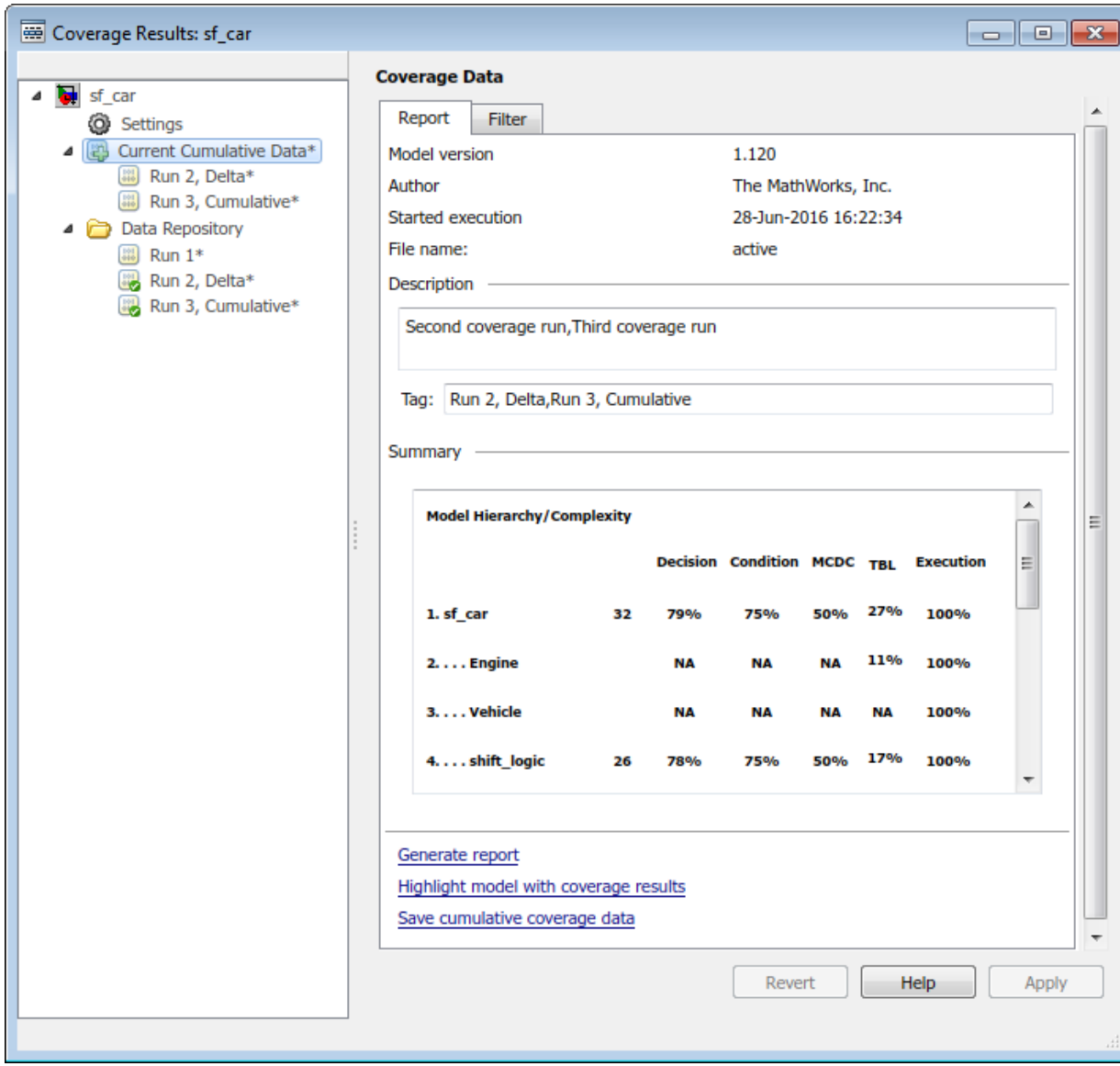
Managing Coverage Data from the Results Explorer

After you record coverage, you can manage the coverage data from the Results Explorer. To view coverage data details, under **Current Cumulative Data**, click the coverage data of interest. You can edit the description and tags for each run. Before you leave the coverage data details view, click **Apply** to apply your changes. Otherwise, the changes are reverted.

When you apply changes to coverage data, such as adding descriptions and tags, the data shows an asterisk next to its icon. To save these changes, right-click the data and click **Save modified coverage data**.

Accumulating Coverage Data from the Results Explorer

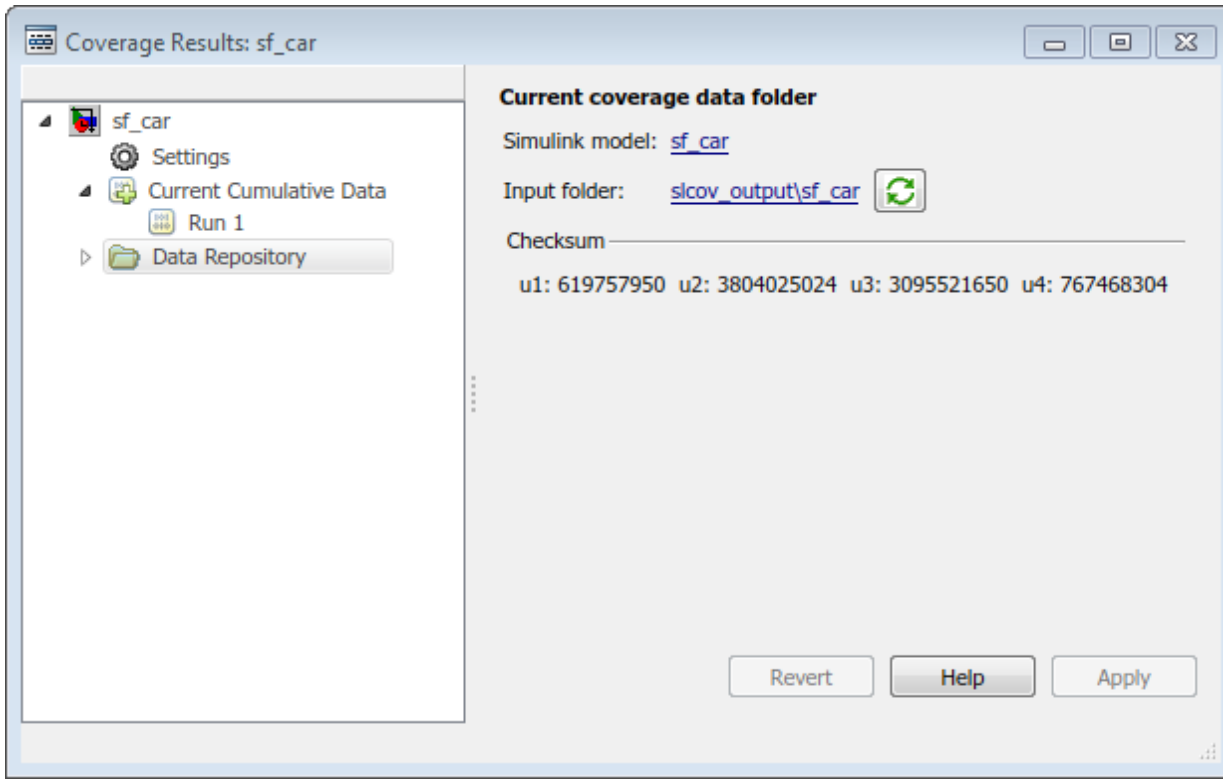
If you record multiple coverage runs, each run is listed separately in the Data Repository. You can drag and drop runs from the Data Repository to the Current Cumulative Data to manage which runs to include in the cumulative coverage data. Alternatively, right-click runs in the Data Repository or the Current Cumulative Data to include or exclude them in the cumulative coverage data.



To save the current cumulative data set to a `.cvt` file, click **Save cumulative coverage data**. Alternatively, you can right-click the **Current Cumulative Data** and select **Save cumulative coverage data**.

Load Existing Coverage Data

The Data Repository contains the coverage data, which is saved to the Input folder. You specify the Input folder on the **Configuration Parameters dialog box > Coverage > "Results Pane" on page 3-7**, in the **Output directory** field.



To synchronize the data in the input folder and the data in the Data Repository, click **Synchronize with the current coverage data folder**.

To load existing coverage data to the Data Repository:

1. Right-click the **Data Repository**.
2. Select **Load coverage data**.
3. Select existing coverage data for the current model and click **Open**.

Cumulative Coverage Data

On the **Coverage > Results** pane in the Configuration Parameters dialog box, if you select **Enable cumulative data collection** and **Save cumulative results in workspace variable**, a coverage running total is updated with new results at the end of each simulation. However, if you change model or block settings between simulations that are incompatible with settings from previous simulations and affect the type or number of coverage points, the cumulative coverage data resets.

When you restore a running total from saved data, the saved results are reflected in the next cumulative report. If a running total exists when you restore a saved value, the existing value is overwritten.

Whenever you report on more than one single simulation, the coverage displayed for truth tables and lookup-table maps is based on the total coverage of all the reported runs. For cumulative reports, this information includes all the simulations where cumulative results are stored. For more information about managing cumulative results, see “Access, Manage, and Accumulate Coverage Results” on page 3-10.

You can make cumulative coverage results persist between MATLAB sessions. The `cvload` parameter `RESTORETOTAL` must be `1` to restore cumulative results. At the end of the sessions, use `cvsave` to save results to a file. At the beginning of the next session, use `cvload` to load the results.

When you save the coverage results to a file using `cvsave` and a model name argument, the file also contains the cumulative running total. When you load that file into the coverage tool using `cvload`, you can select whether you want to restore the running total from the file.

You can also calculate cumulative coverage results at the command line, through the `+` operator:

```
covdata1 = cvsim(test1);
covdata2 = cvsim(test2);
cvhtml('cumulative_report', covdata1 + covdata2);
```

4

Code Coverage

Types of Code Coverage

If you have Embedded Coder, Simulink Coverage can perform several types of code coverage analysis for models in software-in-the-loop (SIL) mode, processor-in-the-loop (PIL) mode, and for the code within supported S-Function blocks.

In this section...
“Statement Coverage for Code Coverage” on page 4-2
“Condition Coverage for Code Coverage” on page 4-3
“Decision Coverage for Code Coverage” on page 4-3
“Modified Condition/Decision Coverage (MCDC) for Code Coverage” on page 4-4
“Cyclomatic Complexity for Code Coverage” on page 4-5
“Relational Boundary for Code Coverage” on page 4-5
“Function Coverage” on page 4-5
“Function Call Coverage” on page 4-6

Statement Coverage for Code Coverage

Statement coverage determines the number of source code statements that execute when the code runs. Use this type of coverage to determine whether every statement in the program has been invoked at least once.

Statement coverage = (Number of executed statements / Total number of statements) *100

Statement Coverage Example

This code snippet contains five statements. To achieve 100% statement coverage, you need at least three test cases. Specifically, tests with positive *x* values, negative *x* values, and *x* values of zero.

```
if (x > 0)
  printf( "x is positive" );
else if (x < 0)
  printf( "x is negative" );
else
  printf( "x is 0" );
```

Condition Coverage for Code Coverage

Condition coverage analyzes statements that include conditions in source code. Conditions are C/C++ Boolean expressions that contain relation operators (<, >, <=, or >=), equation operators (!= or ==), or logical negation operators (!), but that do not contain logical operators (&& or ||). This type of coverage determines whether every condition has been evaluated to all possible outcomes at least once.

Condition coverage = (Number of executed condition outcomes / Total number of condition outcomes) *100

Condition Coverage Example

In this expression:

```
y = x<=5 && x!=7;
```

there are these conditions:

```
x<=5
x!=7
```

Decision Coverage for Code Coverage

Decision coverage analyzes statements that represent decisions in source code. Decisions are Boolean expressions composed of conditions and one or more of the logical C/C++ operators && or ||. Conditions within branching constructs (if/else, while, do-while) are decisions. Decision coverage determines the percentage of the total number of decision outcomes the code exercises during execution. Use this type of coverage to determine whether all decisions, including branches, in your code are tested.

Note The decision coverage definition for DO-178C compliance differs from the Simulink Coverage definition. For decision coverage compliance with DO-178C, select the `Condition Decision` structural coverage level for Boolean expressions not containing && or || operators.

Decision coverage = (Number of executed decision outcomes / Total number of decision outcomes) *100

Decision Coverage Example

This code snippet contains three decisions:

```
y = x<=5 && x!=7;                // decision #1

if( x > 0 )                     // decision #2
  printf( "decision #2 is true" );
else if( x < 0 && y )                // decision #3
  printf( "decision #3 is true" );
else
  printf( "decisions #2 and #3 are false" );
```

Modified Condition/Decision Coverage (MCDC) for Code Coverage

Modified condition/decision coverage (MCDC) is the extent to which the conditions within decisions are independently exercised during code execution.

- All conditions within decisions have been evaluated to all possible outcomes at least once.
- Every condition within a decision independently affects the outcome of the decision.

MCDC coverage = (Number of conditions evaluated to all possible outcomes affecting the outcome of the decision / Total number of conditions within the decisions) *100

Modified Condition/Decision Coverage Example

For this decision:

```
X || ( Y && Z )
```

the following set of test cases delivers 100% MCDC coverage.

	X	Y	Z
Test case #1	0	0	1
Test case #2	0	1	0
Test case #3	0	1	1
Test case #4	1	0	1

Cyclomatic Complexity for Code Coverage

Cyclomatic complexity is a measure of the structural complexity of code that uses the McCabe complexity measure. To compute the cyclomatic complexity of code, code coverage uses this formula:

$$c = \sum_{1}^{N} (o_n - 1)$$

N is the number of decisions in the code. o_n is the number of outcomes for the n^{th} decision point. Code coverage adds 1 to the complexity number for each C/C++ function.

Coverage Example

For this code snippet:

```
void evalNum( int x ){

    if (x > 0)
      printf( "x is positive" );
    else if (x < 0)
      printf( "x is negative" );
    else
  printf( "x is 0" );
}
```

the cyclomatic complexity is 3.

Relational Boundary for Code Coverage

Relational boundary code coverage examines code that has relational operations. Relational boundary code coverage metrics align with those for model coverage, as described in “Relational Boundary Coverage” on page 1-8. Fixed-point values in your model are integers during code coverage.

Function Coverage

Function coverage determines whether all the functions of your code have been called during simulation. For instance, if there are ten unique functions in your code, function coverage checks if all ten functions have been executed at least once during simulation.

Function Call Coverage

Function call coverage determines whether all function call-sites in your code have been executed during simulation. For instance, if functions are called twenty times in your code, function call coverage checks if all twenty function calls have been executed during simulation.

Code Coverage for Models in Software-in-the-Loop (SIL) Mode and Processor-in-the-Loop (PIL) Mode

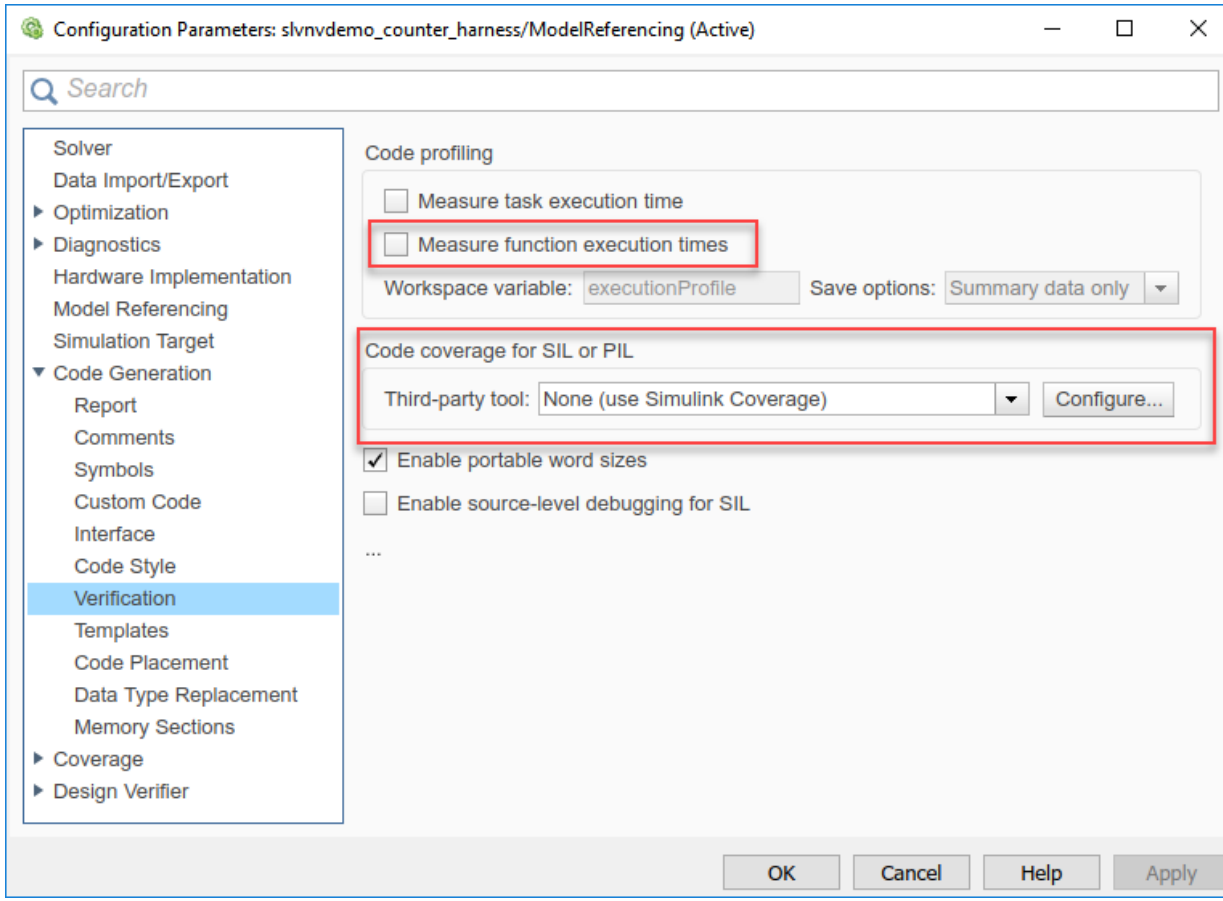
If you have Embedded Coder and Simulink Coverage, you can analyze coverage for generated code during a software-in-the-loop (SIL) or processor-in-the-loop (PIL) simulation.

In this section...
“Enable SIL or PIL Code Coverage for a Model” on page 4-7
“Simulink Coverage Code Coverage Measurement Workflows” on page 4-8
“Review the Coverage Results for Models in SIL or PIL Mode” on page 4-9
“Limitations” on page 4-10

Enable SIL or PIL Code Coverage for a Model

To record SIL or PIL code coverage for a model:

1. In the Configuration Parameters dialog box, on the left pane, click **Code Generation**. From the list, select **Verification**.
2. Under **Code profiling**, clear **Measure function execution times**.
3. Under **Code coverage for SIL or PIL**, for the **Third-party tool** select `None (use Simulink Coverage)`.



Simulink Coverage Code Coverage Measurement Workflows

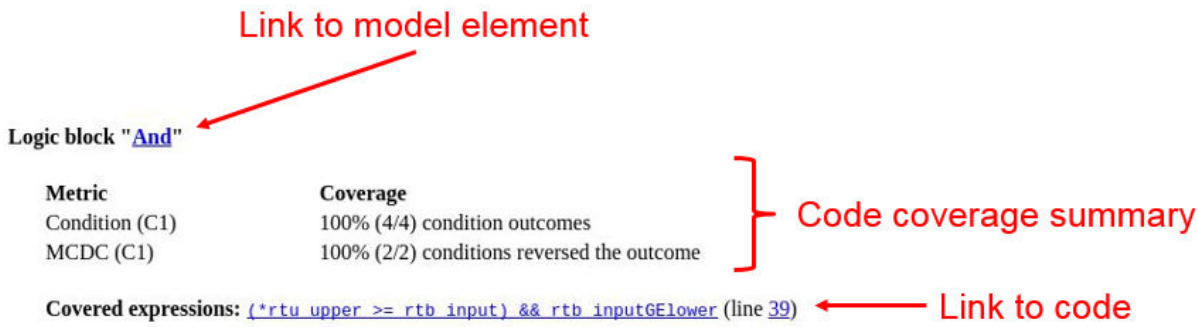
To measure code coverage, use either of these workflows:

- The top model is in SIL mode or PIL mode. Simulink Coverage measures code coverage for the top model, depending `RecordCoverage`. Simulink Coverage also measures code coverage for referenced models, depending on `CovModelRefEnable`.
- The top model is in Normal mode and contains at least one reference model in SIL or PIL mode. Simulink Coverage measures code coverage for the referenced model if

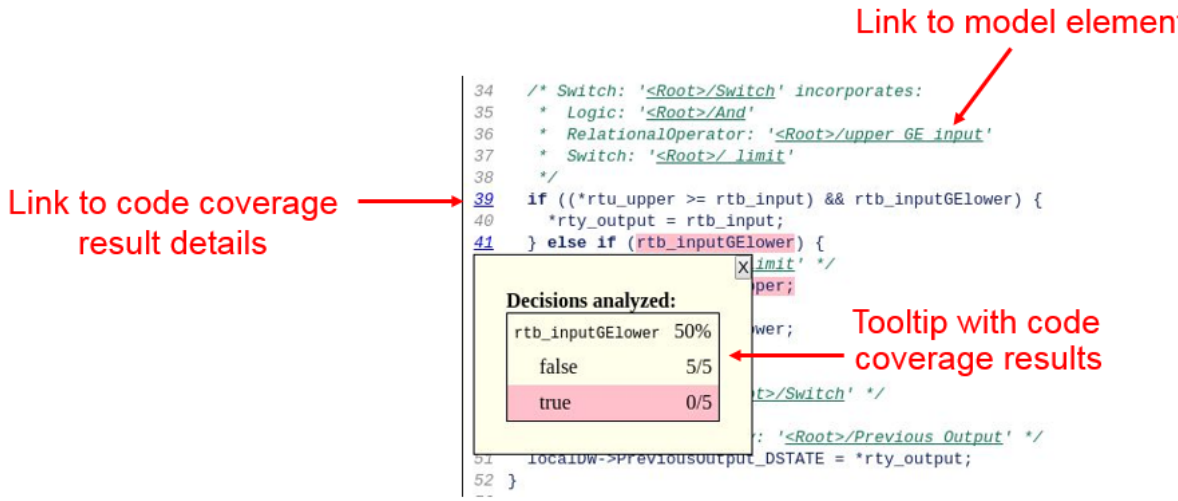
`CovModelRefEnable` is `'on'`, `'all'`, or `'filtered'` and `RecordCoverage` is `'off'`.

Review the Coverage Results for Models in SIL or PIL Mode

In the code coverage report, each hyperlink opens a report with more details on the coverage analysis for the model. The code coverage results in these reports are similar to the coverage results for C/C++ code in S-function blocks, as described in “View Coverage Results for C/C++ Code in S-Function Blocks” on page 5-49. You can navigate from code coverage results to the associated model blocks by using the links within the detailed code coverage reports.



Each detailed code coverage report also contains syntax highlighted code with coverage information.



Limitations

Coverage for models in SIL and PIL mode has these limitations:

- The model must meet the requirements listed in “Enable SIL or PIL Code Coverage for a Model” on page 4-7.
- Code coverage results must not include external C/C++ files in read-only folders.

See Also

Related Examples

- “Software-in-the-Loop Code Coverage”

Specify Code Coverage Options

Simulink Coverage provides three modes of code coverage analysis. For general coverage options, see “Specify Coverage Options” on page 3-2.

In this section...
“Models with S-Function Blocks” on page 4-11
“Models with Software-in-the-Loop and Processor-in-the-Loop Mode Blocks” on page 4-11
“Models with MATLAB Function Blocks” on page 4-12

Models with S-Function Blocks

Configure an S-Function block for coverage based on how you created it. For more information, see “Coverage for C and C++ S-Functions” on page 5-46.

Note If you have software-in-the-loop or processor-in-the-loop blocks in your model, set the options described in “Models with Software-in-the-Loop and Processor-in-the-Loop Mode Blocks” on page 4-11.

Models with Software-in-the-Loop and Processor-in-the-Loop Mode Blocks

1. In the Simulink Editor, select **Simulation** > **Model Configuration Parameters**.
2. Before setting code coverage options, on the **Code Generation** pane in the Configuration Parameters dialog box, set the **System target file** in the **Target selection** menu to `ert.tlc`.
3. In the Configuration Parameters dialog box, on the left pane, click **Code Generation**. From the list, select **Verification**.
4. Select the code coverage tool from the **Code coverage for SIL or PIL** tab.

You can measure code coverage using these tools:

- Simulink Coverage code coverage tool
- BullseyeCoverage
- LDRA TestBed

BullseyeCoverage and LDRA TestBed are third-party tools supported by Embedded Coder. For more information on third-party code coverage tool support, see “Code Coverage Tool Support” (Embedded Coder). To set code coverage options, click **Configure**. If you select `None (use Simulink Coverage)` as the code coverage tool, the software opens the **Coverage** pane when you click **Configure**.

Using Simulink Coverage for code coverage means that you can analyze coverage results, justify missing coverage, and generate more test cases from within the Simulink environment.

Models with MATLAB Function Blocks

When you record coverage for models containing MATLAB Function blocks, code coverage is recorded for the code within the MATLAB Function blocks. To include MATLAB Function blocks in your analysis:

1. In the Simulink Editor, select **Analysis > Coverage > Settings**.
2. In the Configuration Parameters dialog box, on the **Coverage** pane, under **Include in analysis**, select **MATLAB files**.

See Also

More About

- “Create and Run Test Cases” on page 5-3
- “Types of Coverage Reports” on page 6-2
- “View Coverage Results for C/C++ Code in S-Function Blocks” on page 5-49
- “Coverage Filtering” on page 7-2

Coverage for Models with Code Blocks and Simulink Blocks

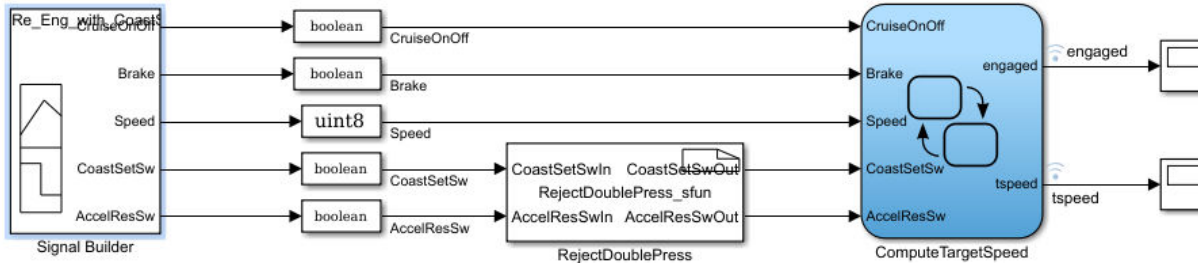
In this section...
“Set Up the Model to Record Coverage” on page 4-13 “Record Coverage” on page 4-14 “Review Results by Generating a Coverage Report” on page 4-14 “Justify Missing Coverage” on page 4-15

In this example, you record coverage for a model which contains a combination of code blocks and other Simulink blocks.

Set Up the Model to Record Coverage

1 Open the model.

```
addpath(fullfile(docroot, 'toolbox', 'slvnv', 'examples'));
open_system('ex_cc_cruise_control_doublepress_sfun');
```



The model is a cruise control system that consists of test cases and input signals from a Signal Builder block. The signals from the Signal Builder act as inputs to the Stateflow chart `ComputeTargetSpeed`, which engages or disengages the cruise control system and sets the target speed, `tspeed`.

Navigate to a writable folder on the MATLAB path. Copy the header file `RejectDoublePress.h` to the current folder.

```
copyfile(fullfile(docroot, 'toolbox', 'slvnv', ...
'examples', 'RejectDoublePress.h'),'./');
```

2. In the Simulink Editor, select **Simulation > Model Configuration Parameters**. Before setting code coverage options, on the **Code Generation** pane in the Configuration Parameters dialog box, set the **System target file** in the **Target selection** menu to `ert.tlc`. Navigate to the **Verification** tab of the **Code Generation** pane. From the **Code coverage for SIL or PIL** tab, select `None (use Simulink Coverage)` as the code coverage tool.
3. In the **Coverage** pane, set the options for coverage calculated during simulation.
    1. Select **Enable coverage analysis**.
    2. In the **Include in analysis** section, ensure that **C/C++ S-Functions** is selected.
    3. In the **Coverage metrics** section, select `Modified Condition Decision (MCDC)` as the **Structural coverage level**. Apply the changes by clicking **Apply**.
4. Open the `RejectDoublePress` S-Function Builder block. In the **Build options** of the **Build Info** tab, select **Enable support for coverage**. To build the S-Function, click **Build** .

Note To build the S-Function, you must have a compiler installed. For more information on supported compilers for various platforms, see Supported and Compatible Compilers.

Record Coverage

1. Open the Signal Builder block.

```
open_system('ex_cc_cruise_control_doublepress_sfun/Signal Builder');
```

2. The Signal Builder consists of eight signal groups with five signals each. In this example, we simulate all the signal groups and record coverage. Click **Run all and produce coverage** to start recording coverage. At the end of the simulation, the Coverage Results Explorer opens, showing the results for the latest coverage analysis. The blocks in the model are highlighted in different colors corresponding to the level of coverage achieved by each block.

Review Results by Generating a Coverage Report

The Coverage Results Explorer offers several options for displaying and reporting coverage results. Select the `Not_Engaged_with_Enable` group in the **Current Cumulative Data** tab of the left pane. Click the **Generate report** link at the bottom of the Coverage Results Explorer to generate an HTML coverage report in the built-in

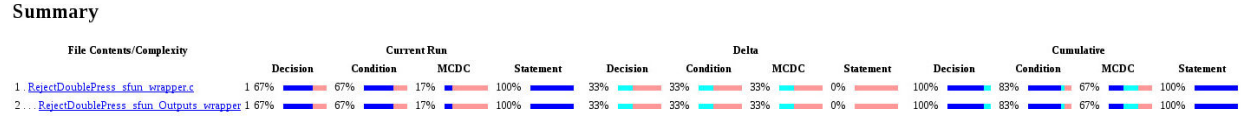
MATLAB web browser. The coverage report lists model coverage for Simulink model blocks and code coverage for code blocks.

Scroll down to view the coverage metrics for the S-Function block in the coverage report. Click the **Detailed Report** link to open the code coverage report for the S-Function block. For more details on the code coverage report for S-Function blocks, see“View Coverage Results for C/C++ Code in S-Function Blocks” on page 5-49.

Justify Missing Coverage

In this example, we justify coverage for one input signal group by creating a coverage filter. In the code coverage report for the S-Function block created in “Review Results by Generating a Coverage Report” on page 4-14, scroll down to Decision/Condition 2.1 `! (CoastSetSwIn[0] && AccelResSwIn[0])`. This condition is never `False` for the current test case. We can therefore justify this condition in our coverage analysis.

1. Click the **Justify or Exclude** link under the detailed results for this condition. The **Filter** tab of the Coverage Results Explorer opens, and the rule filtering this transition is added. Change the **Mode** for this rule to `Justified` and enter a description for the **Rationale**, such as “expression cannot be false”. Click **Apply** to apply the changes.
2. After you click **Apply**, the **Generate report** link becomes available. Click the link to generate the report with the updated coverage filter. The new code coverage report for the `RejectDoublePress` S-Function block lists the excluded condition under **Objects Filtered from Coverage Analysis**. The detailed results for the condition `! (CoastSetSwIn[0] && AccelResSwIn[0])` show that missing coverage for this condition has been justified. The justified objects are treated as satisfied when reporting coverage percentages and appear light blue in the “Coverage Summary” on page 6-12.



For more information on coverage filters, see “Coverage Filtering” on page 7-2.

See Also

“Types of Coverage Reports” on page 6-2 | “Creating and Using Coverage Filters” | “Coverage for C and C++ S-Functions” on page 5-46

Coverage Collection During Simulation

- “Model Coverage Collection Workflow” on page 5-2
- “Create and Run Test Cases” on page 5-3
- “Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage” on page 5-4
- “Modified Condition and Decision Coverage in Simulink Design Verifier” on page 5-8
- “View Coverage Results in a Model” on page 5-12
- “Model Coverage for Multiple Instances of a Referenced Model” on page 5-19
- “Model Coverage for MATLAB Functions” on page 5-29
- “Coverage for C and C++ S-Functions” on page 5-46
- “View Coverage Results for C/C++ Code in S-Function Blocks” on page 5-49
- “Model Coverage for Stateflow Charts” on page 5-54

Model Coverage Collection Workflow

To develop effective tests with model coverage:

1. Develop one or more test cases for your model. (See “Create and Run Test Cases” on page 5-3.)
2. Run the test cases to verify model behavior.
3. Analyze the coverage reports produced by the Simulink Coverage software.
4. Using the information in the coverage reports, modify the test cases to increase their coverage or add new test cases to cover areas not currently covered.
5. Repeat the preceding steps until you are satisfied with the coverage of your test suite.

Note The Simulink Coverage software comes with an example of model coverage to validate model tests. To step through the example, at the MATLAB command prompt, enter `simcovdemo`.

Create and Run Test Cases

To create and run test cases, model coverage provides the MATLAB commands `cvtest` and `cvsim`. The `cvtest` command creates test cases that the `cvsim` command runs. (See “Run Tests with cvsim” on page 8-5.)

You can also run the coverage tool interactively:

1. Open the `sldemo_fuelsys` model.
2. In the Simulink model window, select **Analysis > Coverage > Settings**.

   In the Configuration Parameters dialog box, on the “Coverage Pane” on page 3-2, select **Enable coverage analysis**, which enables the coverage settings.
3. Under **Coverage metrics**, select the types of coverage that you want to record in the coverage report. Click **OK**.
4. In the Simulink Editor, select **Simulation > Run** to start simulating the model.

   On the “Results Pane” on page 3-7 of the Configuration Parameters dialog box, if you specify to report model coverage, Simulink Coverage saves coverage data for the current run in the workspace object `covdata` and cumulative coverage data in `covCumulativeData`, by default. Simulink Coverage also saves these results to a `.cvt` file by default. At the end of the simulation, the data appears in an HTML report that opens in a browser window. For more information on coverage data settings, see “Specify Coverage Options” on page 3-2.

   You cannot run simulations if you select both the model coverage reporting and acceleration options. In the Simulink Editor, if you select **Simulation > Mode > Accelerator** , Simulink Coverage does not record coverage.

   When you perform coverage analysis, you cannot select both block reduction and conditional branch input optimization, because they interfere with coverage recording.

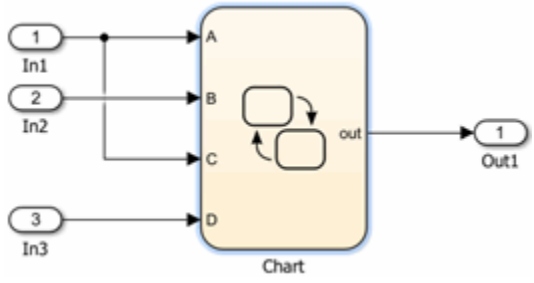
Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage

Simulink Coverage by default uses the masking modified condition and decision coverage (MCDC) definition for recording MCDC coverage results. Although you can change the MCDC definition that Simulink Coverage uses during analysis to the unique-cause MCDC definition, there are some differences in how Simulink Coverage records coverage for models depending on which definition you use.

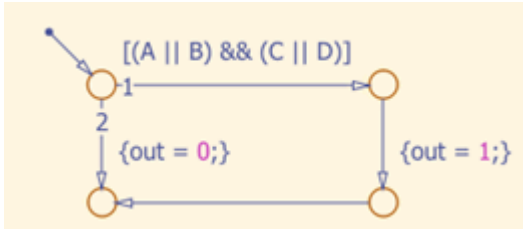
In this section...
“Differences between Masking MCDC and Unique-Cause MCDC in Simulink Coverage Coverage Analysis” on page 5-4
“Certification Considerations for MCDC Coverage” on page 5-6
“Setting the (MCDC) Definition Used for Simulink Coverage Coverage Analysis” on page 5-6
“Modified Condition and Decision Coverage in Simulink Design Verifier” on page 5-7

Differences between Masking MCDC and Unique-Cause MCDC in Simulink Coverage Coverage Analysis

Masking MCDC accounts for the masking of conditions in subexpressions, allowing for an increased number of satisfied MCDC objectives compared to the unique-cause definition of MCDC. As a result, some Simulink models that receive less than complete MCDC coverage using the unique-cause MCDC definition receive increased coverage when using the masking MCDC definition. Consider the following example, where two inputs to a Stateflow chart, condition A and condition C, cannot change independently:



This input dependence results in dependent conditions for the expression contained within the Stateflow chart:



For the expression (A||B)&&(C||D), changing the value of condition C also changes the value of condition A. Due to the interdependence of conditions A and C, unique-cause MCDC for condition C cannot be achieved:

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	True Out	False Out
(A || B) && (C || D)		
A	TxTx	FFxx
B	FTFT	FFxx
C	TxTx	(TxFF)
D	FTFT	FTFF

However, masking MCDC for condition C can be achieved, because masking MCDC allows the value of condition A to change in the independence pair for condition C, as long as the subexpression (A||B) remains true:

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	True Out	False Out
(A ‖ B) && (C ‖ D)		
A	TxTx	FFxx
B	FTFT	FFxx
C	TxTx	FTFF
D	FTFT	FTFF

Certification Considerations for MCDC Coverage

The Certification Authorities Software Team (CAST), in their CAST 6 position paper, states that masking MCDC is acceptable for meeting the MC/DC objective of DO-178B certification.

Setting the (MCDC) Definition Used for Simulink Coverage Coverage Analysis

By default, Simulink Coverage uses the masking MCDC definition during coverage analysis. There are two ways to change the MCDC definition used for Simulink Coverage coverage analysis:

Use the Model Configuration Parameters to Set the MCDC Definition Used

1. Open the **Configuration Parameters** dialog box.
2. Set the `CovMcdcMode` parameter to `Masking` or `Unique-Cause`.

Use the cvtest Object to Set the MCDC Definition Used

Create a `cvtest` object for your model to set the `mcdcMode` to `'Masking'` or `'UniqueCause'`:

```
cvt = cvtest(model)
cvt.options.mcdcMode = 'UniqueCause'
covdata = cvsim(cvt)
```

Modified Condition and Decision Coverage in Simulink Design Verifier

Setting `CovMcdcMode` to `'UniqueCause'` can result in differences between MCDC reporting in Simulink Coverage and test generation in Simulink Design Verifier. Simulink Design Verifier always uses the masking MCDC definition for test case generation. For more information, see “Modified Condition and Decision Coverage in Simulink Design Verifier” on page 5-8.

See Also

More About

- “MCDC” (Simulink Design Verifier)

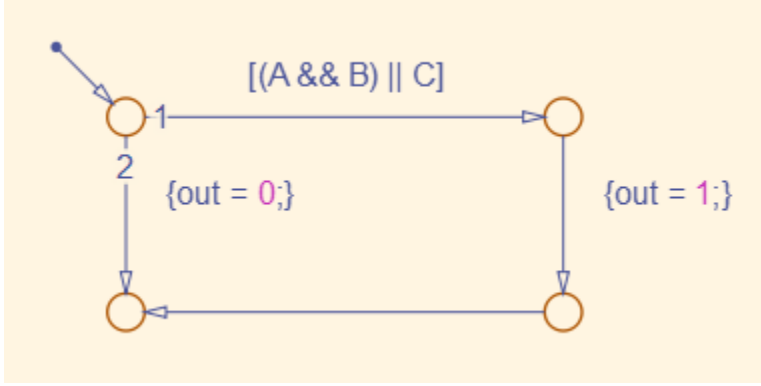
Modified Condition and Decision Coverage in Simulink Design Verifier

Depending on the settings you apply for Simulink Coverage coverage recording, there can be a difference between the definition of modified condition and decision (MCDC) coverage used for model coverage analysis in Simulink Coverage and that used for test case generation analysis in Simulink Design Verifier.

MCDC Definitions for Simulink Coverage and Simulink Design Verifier

Simulink Design Verifier always uses the masking MCDC definition for test case generation. By default, Simulink Coverage also uses the masking MCDC definition when recording coverage. However, if you set the `CovMcdcMode` model configuration parameter to `'UniqueCause'`, Simulink Coverage instead uses the unique-cause MCDC definition when recording coverage. For information on the differences between the masking MCDC definition and the unique-cause MCDC definition, see “Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage” on page 5-4.

Setting the `CovMcdcMode` model configuration parameter to `'UniqueCause'` can result in differences between MCDC reporting in Simulink Coverage and test generation in Simulink Design Verifier. An example of this difference can be seen in analysis results for logical expressions containing a mixture of `AND` and `OR` operators, as in this Stateflow transition.



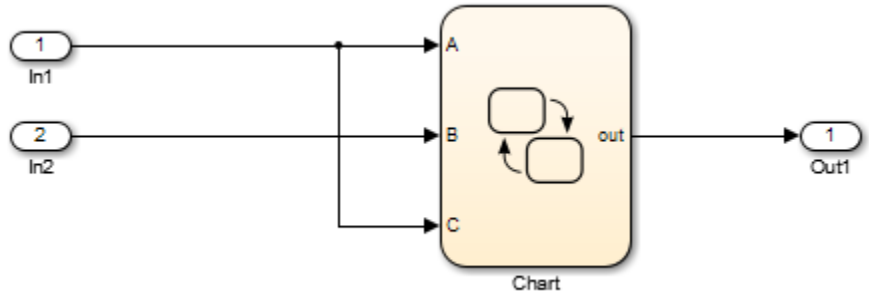
Given that `A`, `B`, and `C` are each separate inputs, there are five possible ways to evaluate the condition on the Stateflow transition, shown in the following table.

	A	B	C	(A && B) || C
1	F	x	F	F
2	F	x	T	T
3	T	F	F	F
4	T	F	T	T
5	T	T	x	T

Satisfying MCDC for a Boolean variable requires a pair of condition evaluations, showing that a change in that variable alone changes the evaluation of the entire expression. In this example, MCDC can be satisfied for `C` with either the pair 1, 2 or the pair 3, 4. In both of those cases, the value of the expression changed because the value of `C` changed, while all other variable values stayed the same.

Each pair has a different set of values for `A` and `B` which are held constant, but each pair contains one evaluation where `C` and `out` are true and one evaluation where `C` and `out` are false. To satisfy MCDC for `C`, Simulink Design Verifier test generation analysis accepts any pair containing one evaluation of true values and one evaluation of false values for `C` and `out`. In this example, Simulink Design Verifier test generation analysis accepts not only pair 1, 2 and pair 3, 4 but also pair 1, 4 and pair 2, 3. Simulink Coverage model coverage analysis using the unique-cause MCDC definition is satisfied only by pair 1, 2 or by pair 3, 4.

The preceding example assumes that `A`, `B`, and `C` are all separate inputs. When input `A` is constrained to be the same value as `C`, as in this model, only a subset of condition evaluations are possible.



This subset of condition evaluations for the Stateflow transition is shown in the following table.

	A	B	C	(A && B) || C
1	F	x	F	F
4	T	F	T	T
5	T	T	x	T

Evaluations 2 and 3 are no longer possible, so neither pair 1, 2 nor pair 3, 4 is possible. As a result, unique-cause MCDC for `C` can no longer be satisfied in Simulink Coverage model coverage analysis. Since pair 1, 4 is still possible, however, Simulink Design Verifier test generation analysis reports that MCDC for `C` is satisfiable.

The complexity of MCDC analysis for logical expressions with a mixture of `AND` and `OR` operators causes this difference between results from Simulink Coverage set to unique-cause MCDC analysis and Simulink Design Verifier. The default`CovMcdcMode` model configuration parameter value of `'Masking'` does not cause this discrepancy. However, if you require the use of unique-cause MCDC analysis in Simulink Coverage, you can minimize this effect by using the `IndividualObjectives` test suite optimization for test generation analysis in Simulink Design Verifier For more information, see the Tip section of “Test suite optimization” (Simulink Design Verifier).

See Also

More About

- “MCDC” (Simulink Design Verifier)

View Coverage Results in a Model

In this section...
“Overview of Model Coverage Highlighting” on page 5-12
“Enable Coverage Highlighting” on page 5-13
“View Results in Coverage Display Window” on page 5-16

Overview of Model Coverage Highlighting

When you simulate a Simulink model, you can configure your model to provide visual results that allow you to see at a glance which objects recorded 100% coverage. After the simulation:

- In the model window, model objects are highlighted in certain colors according to what coverage was recorded:
  - Light green indicates that an object received full coverage during testing.
  - Light red indicates that an object received incomplete coverage.
  - Gray indicates that an object was filtered from coverage.
  - Objects with no color highlighting received no coverage.
- When you click a colored object, the Coverage Display Window provides details about the coverage recorded for that block. For subsystems and Stateflow charts, the Coverage Display Window lists the summary coverage for all objects in that subsystem or chart. For other blocks, the Coverage Display Window lists specific details about the objects that did not receive 100% coverage.

The simulation highlights blocks that received the following types of model coverage:

- “Execution Coverage (EC)” on page 1-3
- “Decision Coverage (DC)” on page 1-3
- “Condition Coverage (CC)” on page 1-3
- “Modified Condition/Decision Coverage (MCDC)” on page 1-4
- “Relational Boundary Coverage” on page 1-8
- “Saturate on Integer Overflow Coverage” on page 1-8

- “Objectives and Constraints Coverage” on page 1-7

Enable Coverage Highlighting

To enable the model coverage colored diagram display:

1. In the Simulink Editor, select **Analysis > Coverage > Settings** to open the **Coverage** pane of the Configuration Parameters dialog box.
2. In the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis** and then select **Entire System**.
3. Select `MCDC` as the **Structural coverage level**.
4. Select **Objectives and Constraints** under **Other metrics**.
5. On the **Results** pane of the Configuration Parameters dialog box, select **Display coverage results using model coloring**. This is the default setting.

After you have enabled the coverage coloring, simulate your model. In the model, you can see at a glance which objects received full, partial, or no coverage.

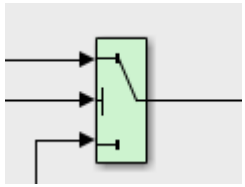
Highlighted Coverage Results

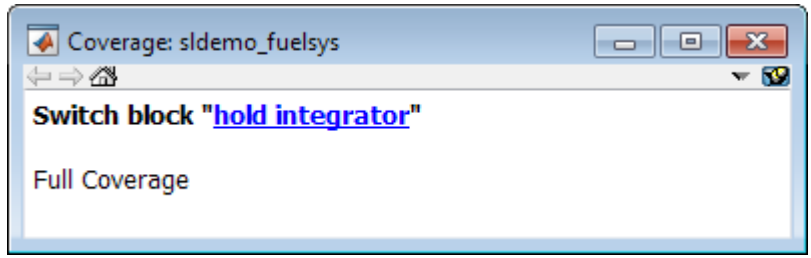
The following sections show examples of highlighted model objects in colors that correspond to the recorded coverage.

- “Green: Full Coverage” on page 5-13
- “Red: Partial Coverage” on page 5-14
- “Gray: Filtered Coverage” on page 5-16

Green: Full Coverage

In this example, the Switch block received 100% coverage, as indicated by the green highlighting and the information in the Coverage Display Window.

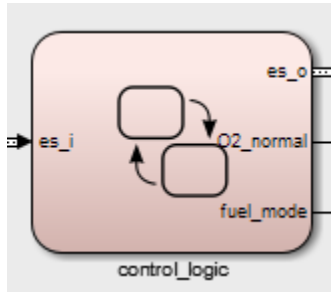


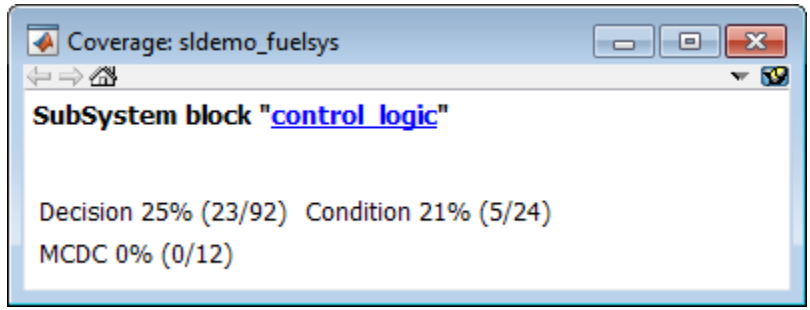


Red: Partial Coverage

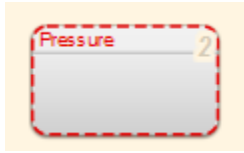
In this example, the control_logic Stateflow chart received the following coverage:

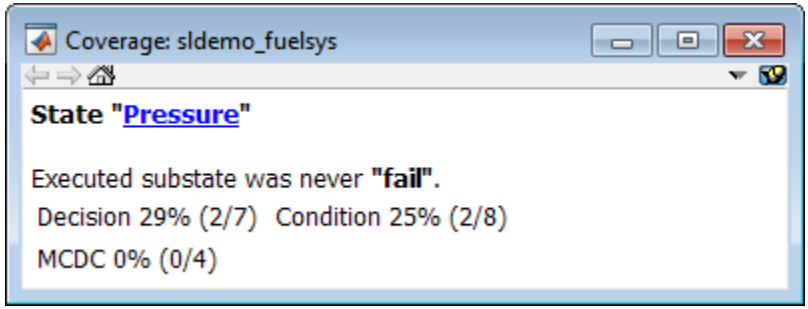
- Decision: 25%
- Condition: 21%
- MCDC: 0%



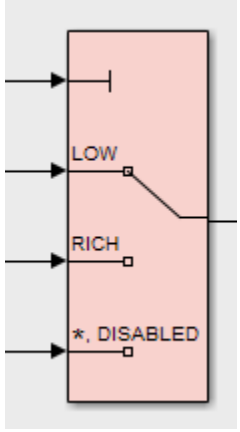


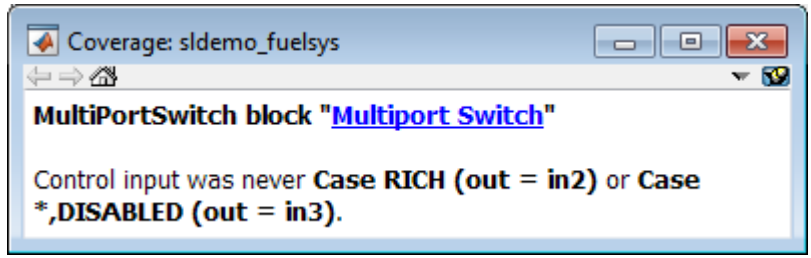
Inside the control_logic subsystem, the Pressure substate was never `fail`.





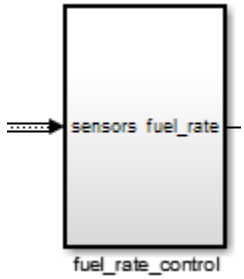
In the next example, in the Multiport Switch block, two of the data ports were never executed.





Gray: Filtered Coverage

In this example, the fuel_rate_control subsystem is highlighted in gray because it was filtered out of coverage recording.



View Results in Coverage Display Window

After simulating the model and recording coverage, by default, the Coverage Display Window is the top-most visible window. When you click an object that recorded coverage, the Coverage Display Window displays details of the coverage recorded during simulation.

In the Coverage Display Window, you can:

- Configure the window so it is not always the top-most visible window. Next to **Always on top**, click and removing the check.
- Configure the window to display coverage information when you click an object that recorded coverage. Click and select **Click**.
- Configure the window to display coverage information when you hover your cursor on an object that recorded coverage. Click and select **Focus**.

- Close the window. Press **Alt**+**F4**.
- Close the window and remove highlighting on the model. Select **Display** > **Remove Highlighting**.

Abbreviations in Coverage Results

The Coverage Display Window shows results with abbreviations. You can view expanded results in the “Top-Level Model Coverage Report” on page 6-12.

Abbreviation	Meaning
CND	condition
ENBL	enable
FCALL	function call
IN	input
LGC	logic
LL	lower limit
MX ITER	maximum iterations exceeded
NA	not applicable
OffThresh	[switch] off threshold
OnThresh	switch on threshold
OUT	output
SO	saturate on integer overflow
TBL	lookup table
THRESH	threshold
TI	test interval
TO	test objective
TP	test point
TRIG	trigger
U	input
UL	upper limit

Abbreviation	Meaning
X	• integration result (Discrete-Time Integrator block) • slew rate (Rate Limiter block)

Model Coverage for Multiple Instances of a Referenced Model

In this section...
“About Coverage for Model Blocks” on page 5-19 “Record Coverage for Multiple Instances of a Referenced Model” on page 5-19

About Coverage for Model Blocks

Model blocks do not receive coverage directly; if you set the simulation mode of the Model block to `Normal` , `SIL`, or `PIL`, the Simulink Coverage software records coverage for the model referenced from the Model block. If the simulation mode for the Model block is anything other than `Normal`, `SIL`, or `PIL`, the software does not record coverage for the referenced model.

Your Simulink model can contain multiple Model blocks with the same simulation mode that reference the same model. When the software records coverage, each instance of the referenced model can be exercised with different inputs or parameters, possibly resulting in additional coverage for the referenced model.

The Simulink Coverage software records coverage for all instances of the referenced model with the same simulation mode and combines the coverage data for that referenced model in the final results.

Record Coverage for Multiple Instances of a Referenced Model

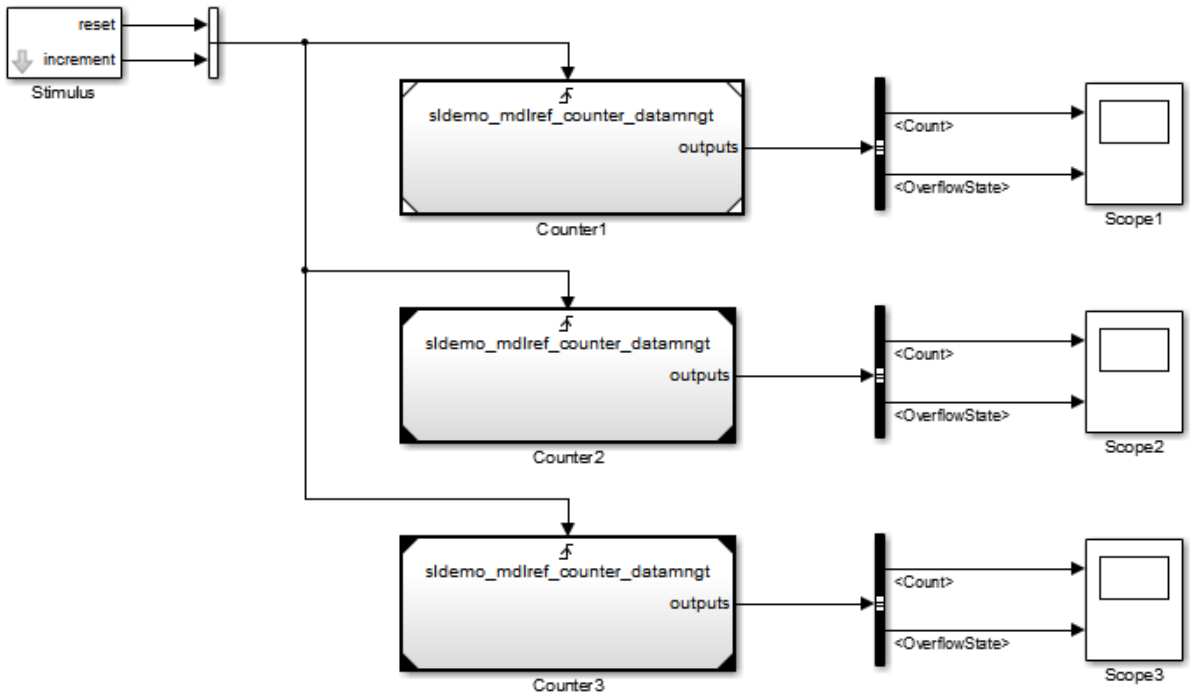
To see how this works, simulate a model twice. The first time, you record coverage for one Model block in `Normal` simulation mode. The second time, you record coverage for two Model blocks in `Normal` simulation mode. Both Model blocks reference the same model.

- “Record Coverage for the First Instance of the Referenced Model” on page 5-19
- “Record Coverage for the Second Instance of the Referenced Model” on page 5-25

Record Coverage for the First Instance of the Referenced Model

Record coverage for one Model block.

1 Open your top-level model. This example uses the following model:



This model contains three Model blocks that reference the `sldemo_mdlref_counter_datamngt` example model. The corners of each Model block indicate the value of their **Simulation mode** parameter:

- Counter1 — Simulation mode: `Normal`
- Counter2 — Simulation mode: `Accelerator`
- Counter3 — Simulation mode: `Accelerator`

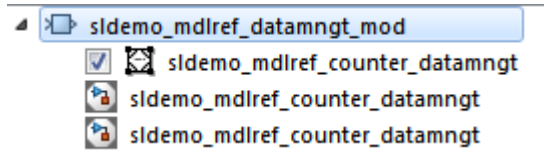
2 Configure your model to record coverage during simulation:

a In the model window, select **Analysis** > **Coverage** > **Settings**.

b On the **Coverage** pane of the Configuration Parameters dialog box, select:

- **Enable coverage analysis**
- **Referenced Models**

c Click **Select Models**. In the Select Models for Coverage Analysis dialog box, you can select only those referenced models whose simulation mode is `Normal`, `SIL`, or `PIL`. In this example, only the first Model block that references `sldemo_mdlref_counter_datamngt` is available for recording coverage.



d Click **OK** to exit the Select Models for Coverage Analysis dialog box.

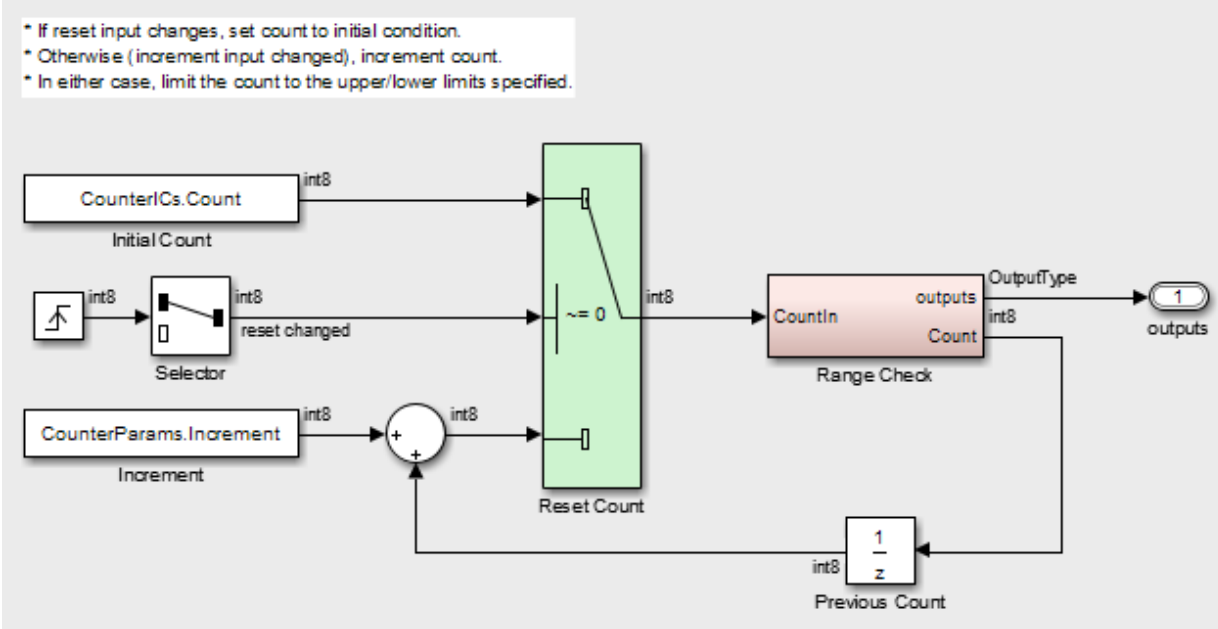
3 Click **OK** to save your coverage settings and exit the Configuration Parameters dialog box.

4 Simulate your model.

When the simulation is complete, the HTML coverage report opens. In this example, the coverage data for the referenced model, `sldemo_mdlref_counter_datamngt`, shows that the model achieved 69% coverage.

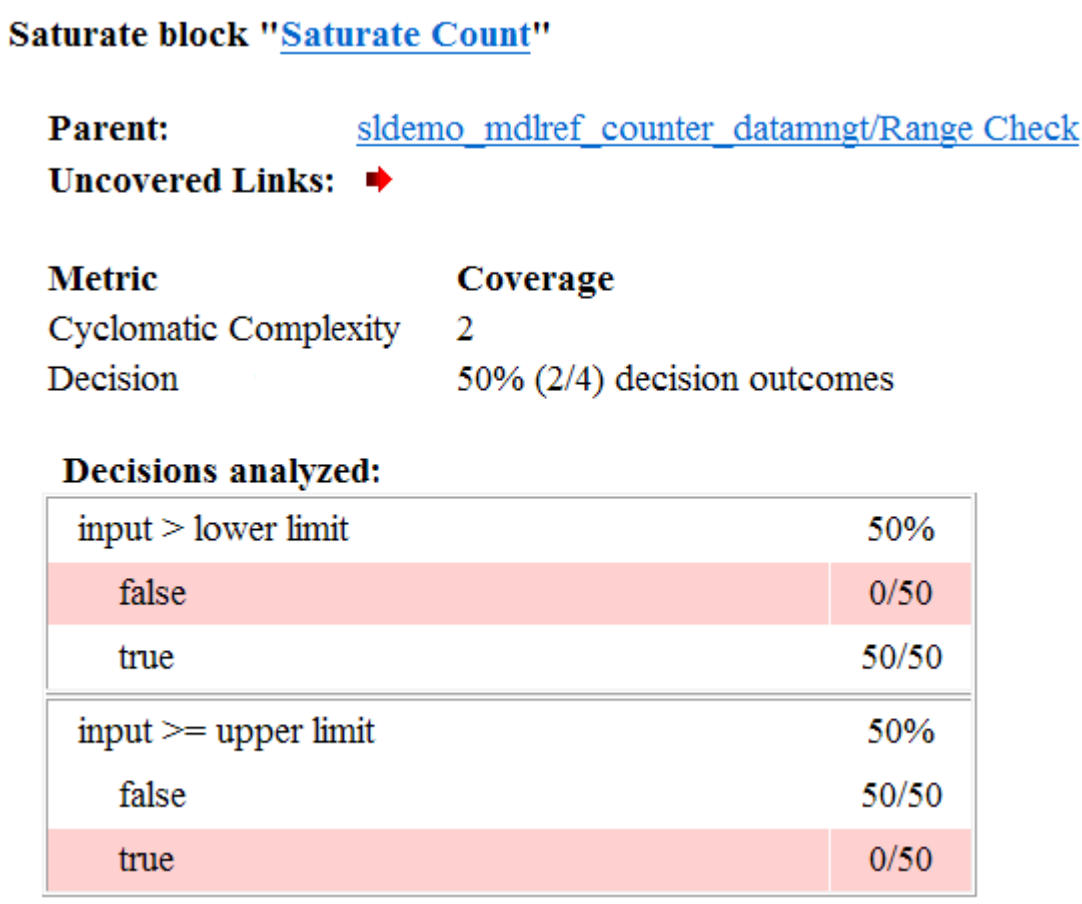
5 Click the hyperlink in the report for the referenced model.

The detailed coverage report for the referenced model opens, and the referenced model appears with highlighting to show coverage results.



Note the following about the coverage for the Range Check subsystem in this example:

- The Saturate Count block executed 100 times. This block has four Boolean decisions. Decision coverage was 50%, because two of the four decisions were never recorded:
  - The decision `input > lower limit` was never `false`.
  - The decision `input >= upper limit` was never `true`.



Saturate block "Saturate Count"

Parent: sldemo_mdlref_counter_datamngt/Range Check
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	2
Decision	50% (2/4) decision outcomes

Decisions analyzed:

input > lower limit	50%
false	0/50
true	50/50
input >= upper limit	50%
false	50/50
true	0/50

- The `DetectOverflow` function executed 50 times. This script has five decisions. The `DetectOverflow` script achieved 60% coverage because two of the five decisions were never recorded:
  - The expression `count >= CounterParams.UpperLimit` was never `true`.
  - The expression `count > CounterParams.LowerLimit` was never `false`.

MATLAB Function "DetectOverflow"

Parent: sldemo_mdlref_counter_datamngt/Range Check/Detect Overflow
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	3
Decision	60% (3/5) decision outcomes

```
 1  function result = DetectOverflow(count, CounterParams)
 2  % DETECTOVERFLOW  Check count
 3  %#codegen
 4
 5  if (count >= CounterParams.UpperLimit)
 6    result = SlDemoRangeCheck.UpperLimit;
 7  elseif (count > CounterParams.LowerLimit)
 8    result = SlDemoRangeCheck.InRange;
 9  else
10    result = SlDemoRangeCheck.LowerLimit;
11  end
12
```

#1: function result = DetectOverflow(count, CounterParams)

Decisions analyzed:

function result = DetectOverflow(count, CounterParams)	100%
executed	50/50

#5: if (count >= CounterParams.UpperLimit)

Decisions analyzed:

if (count >= CounterParams.UpperLimit)	50%
false	50/50
true	0/50

#7: elseif (count > CounterParams.LowerLimit)

Decisions analyzed:

elseif (count > CounterParams.LowerLimit)	50%
false	0/50
true	50/50

Record Coverage for the Second Instance of the Referenced Model

Record coverage for two Model blocks. Set the simulation mode of a second Model block to `Normal` and simulate the model. In this example, the Counter2 block adds to the coverage for the model referenced from both Model blocks.

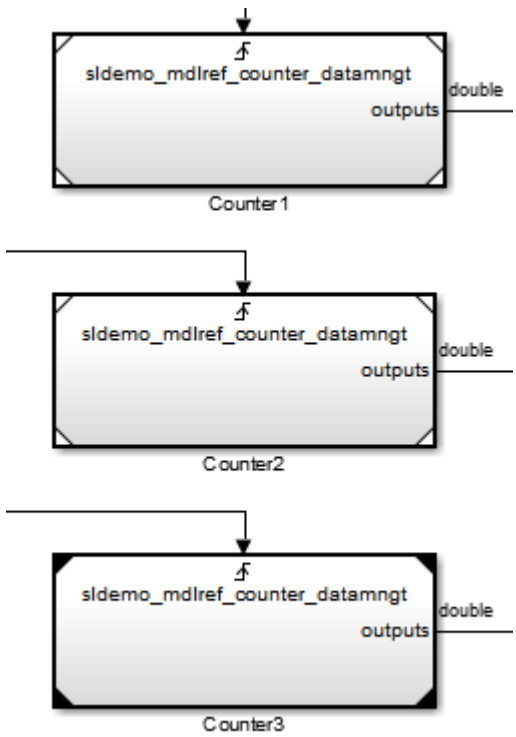
1. In the Simulink Editor for your top-level model, right-click a second Model block and select **Block Parameters (ModelReference)**.

   The Function Block Parameters dialog box opens.

2. Set the **Simulation mode** parameter to `Normal`.

3. Click **OK** to save your change and exit the Function Block Parameters dialog box.

   The corners of the Model block change to indicate that the simulation mode for this block is `Normal`, as in the example below.



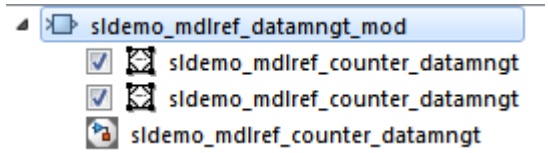
4. To make sure that the software records coverage for both instances of this model:

   a. Select **Analysis > Coverage > Settings**.

   b. On the **Coverage** pane, select **Enable coverage analysis**.

   c. Select **Referenced Models** and click **Select Models**.

      In the Select Models for Coverage Analysis dialog box, verify that both instances of the referenced model are selected. In this example, the list now looks like the following.

      If you have multiple instances of a referenced model in `Normal` mode, you can choose to record coverage for all of them or none of them.

   d. Click **OK** to close the Select Models for Coverage Analysis dialog box.

5. Simulate your model again.

6. When the simulation is complete, open the HTML coverage report.

   In this example, the referenced model achieved 85% coverage. Note the following about the coverage data for the Range Check subsystem:

   - The Saturate Count block executed 179 times. The simulation of the Counter2 block executed the Saturate Count block an additional 79 times, for a total of 179 executions.

     The decision `input >= upper limit` was `true` 21 times during this simulation, compared to 0 during the first simulation. The fourth decision `input > lower limit` was still never `false`. Three out of four decisions were recorded during simulation, so this block achieved 75% coverage.

Saturate block "Saturate Count"

Parent: sldemo_mdlref_counter_datamngt/Range Check
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	2
Decision	75% (3/4) decision outcomes

Decisions analyzed:

input > lower limit	50%
false	0/79
true	79/79
input >= upper limit	100%
false	79/100
true	21/100

- The `DetectOverflow` function executed 100 times. The simulation of the Counter2 block executed the `DetectOverflow` function an additional 50 times.

  The `DetectOverflow` function has five decisions. The expression `count >= CounterParams.UpperLimit` was `true` 21 times during this simulation, compared to 0 during the first simulation. The expression `count > CounterParams.LowerLimit` was never `false`. Four out of five decisions were recorded during simulation, so the `DetectOverflow` function achieved 80% coverage.

MATLAB Function "DetectOverflow"

Parent: sldemo_mdlref_counter_datamngt/Range Check/Detect Overflow
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	3
Decision	80% (4/5) decision outcomes

```
 1  function result = DetectOverflow(count, CounterParams)
 2  % DETECTOVERFLOW  Check count
 3  %#codegen
 4
 5  if (count >= CounterParams.UpperLimit)
 6    result = SlDemoRangeCheck.UpperLimit;
 7  elseif (count > CounterParams.LowerLimit)
 8    result = SlDemoRangeCheck.InRange;
 9  else
10    result = SlDemoRangeCheck.LowerLimit;
11  end
12
```

#1: function result = DetectOverflow(count, CounterParams)

Decisions analyzed:

function result = DetectOverflow(count, CounterParams)	100%
executed	100/100

#5: if (count >= CounterParams.UpperLimit)

Decisions analyzed:

if (count >= CounterParams.UpperLimit)	100%
false	79/100
true	21/100

#7: elseif (count > CounterParams.LowerLimit)

Decisions analyzed:

elseif (count > CounterParams.LowerLimit)	50%
false	0/79
true	79/79

Model Coverage for MATLAB Functions

In this section...
“About Model Coverage for MATLAB Functions” on page 5-29
“Types of Model Coverage for MATLAB Functions” on page 5-29
“How to Collect Coverage for MATLAB Functions” on page 5-31
“Examples: Model Coverage for MATLAB Functions” on page 5-32

About Model Coverage for MATLAB Functions

The Simulink Coverage software simulates a Simulink model and reports model coverage data for the decisions and conditions of code in MATLAB Function blocks. Model coverage only supports coverage for MATLAB functions configured for code generation.

For example, consider the following `if` statement:

```
if (x > 0 || y > 0)
    reset = 1;
```

The `if` statement contains a decision with two conditions (`x > 0` and `y > 0`). The Simulink Coverage software verifies that all decisions and conditions are taken during the simulation of the model.

Types of Model Coverage for MATLAB Functions

The types of model coverage that the Simulink Coverage software records for MATLAB functions configured for code generation are:

- “Decision Coverage” on page 5-29
- “Condition and MCDC Coverage” on page 5-30
- “Simulink Design Verifier Coverage” on page 5-30
- “Relational Boundary Coverage” on page 5-31

Decision Coverage

During simulation, the following MATLAB Function block statements are tested for decision coverage:

- Function header — Decision coverage is 100% if the function or local function is executed.
- `if` — Decision coverage is 100% if the `if` expression evaluates to `true` at least once, and `false` at least once.
- `switch` — Decision coverage is 100% if every `switch` case is taken, including the fall-through case.
- `for` — Decision coverage is 100% if the equivalent loop condition evaluates to `true` at least once, and `false` at least once.
- `while` — Decision coverage is 100% if the equivalent loop condition evaluates to `true` at least once, and evaluates to `false` at least once.

Condition and MCDC Coverage

During simulation, in the MATLAB Function block function, the following logical conditions are tested for condition and MCDC coverage:

- `if` statement conditions
- `while` statement conditions
- Logical expressions in assignment statements

Simulink Design Verifier Coverage

The following MATLAB functions are active in code generation and in Simulink Design Verifier:

- `sldv.condition`
- `sldv.test`
- `sldv.assume`
- `sldv.prove`

When you specify the **Objectives and Constraints** coverage metric in the **Coverage** pane of the Configuration Parameters dialog box, the Simulink Coverage software records coverage for these functions.

Each of these functions evaluates an expression *`expr`*, for example, `sldv.test(`*`expr`*`)`, where *`expr`* is a valid Boolean MATLAB expression. Simulink Design Verifier coverage measures the number of time steps that the expression *`expr`* evaluates to `true`.

If *expr* is `true` for at least one time step, Simulink Design Verifier coverage for that function is 100%. Otherwise, the Simulink Coverage software reports coverage for that function as 0%.

For an example of coverage data for Simulink Design Verifier functions in a coverage report, see “Simulink Design Verifier Coverage” on page 6-46.

Relational Boundary Coverage

If the MATLAB function block contains a relational operation, the relational boundary coverage metric applies to this block.

If the MATLAB function block calls functions containing relational operations multiple times, the relational boundary coverage reports a cumulative result over all instances where the function is called. If a relational operation in the function uses operands of different types in the different calls, relational boundary coverage uses tolerance rules for the stricter operand type. For instance, if a relational operation uses `int32` operands in one call, and `double` operands in another call, relational boundary coverage uses tolerance rules for `double` operands.

For information on the tolerance rules and the order of strictness of types, see “Relational Boundary Coverage” on page 1-8.

How to Collect Coverage for MATLAB Functions

When you simulate your model, the Simulink Coverage software can collect coverage data for MATLAB functions configured for code generation. To enable model coverage, select **Analysis** > **Coverage** > **Settings** and select **Enable coverage analysis**.

You collect model coverage for MATLAB functions as follows:

- Functions in a MATLAB Function block
- Functions in an external MATLAB file

  To collect coverage for an external MATLAB file, **Coverage** pane of the Configuration Parameters dialog box, select **Coverage for MATLAB files**.
- Simulink Design Verifier functions:
  - `sldv.condition`
  - `sldv.test`

- `sldv.assume`
- `sldv.prove`

To collect coverage for these functions, on the **Coverage** pane of the Configuration Parameters dialog box, select the **Objectives and Constraints** coverage metric.

The following section provides model coverage examples for each of these situations.

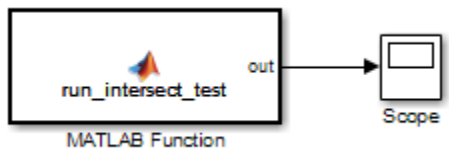
Examples: Model Coverage for MATLAB Functions

- "Model Coverage for MATLAB Function Blocks" on page 5-32
- "Model Coverage for MATLAB Functions in an External File" on page 5-42
- "Model Coverage for Simulink Design Verifier MATLAB Functions" on page 5-43

Model Coverage for MATLAB Function Blocks

Simulink Coverage software measures model coverage for functions in a MATLAB Function block.

The following model contains two MATLAB functions in its MATLAB Function block:



In the Configuration Parameters dialog box, on the **Solver** pane, under **Solver selection**, the simulation parameters are set as follows:

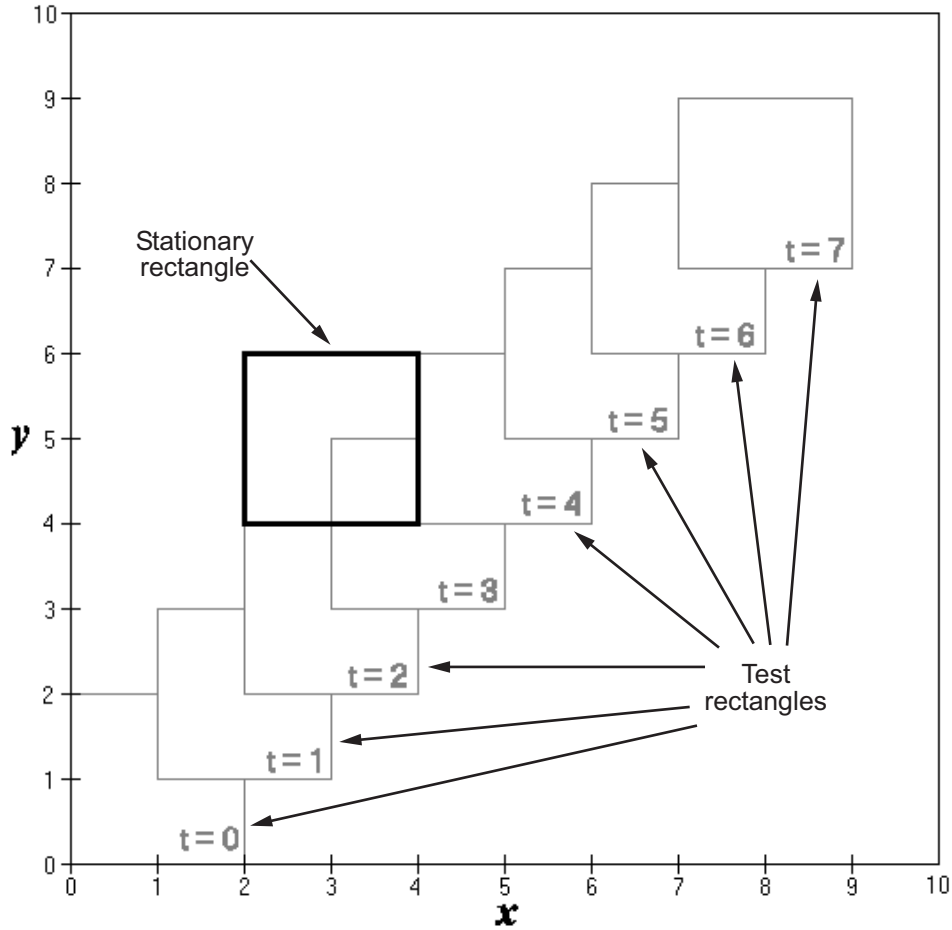
- **Type** — `Fixed-step`
- **Solver** — `discrete (no continuous states)`
- **Fixed-step size (fundamental sample time)** — 1

The MATLAB Function block contains two functions:

- The top-level function, `run_intersect_test`, sends the coordinates for two rectangles, one fixed and the other moving, as arguments to `rect_intersect`.

- The local function, `rect_intersect`, tests for intersection between the two rectangles. The origin of the moving rectangle increases by `1` in the `x` and `y` directions with each time step.

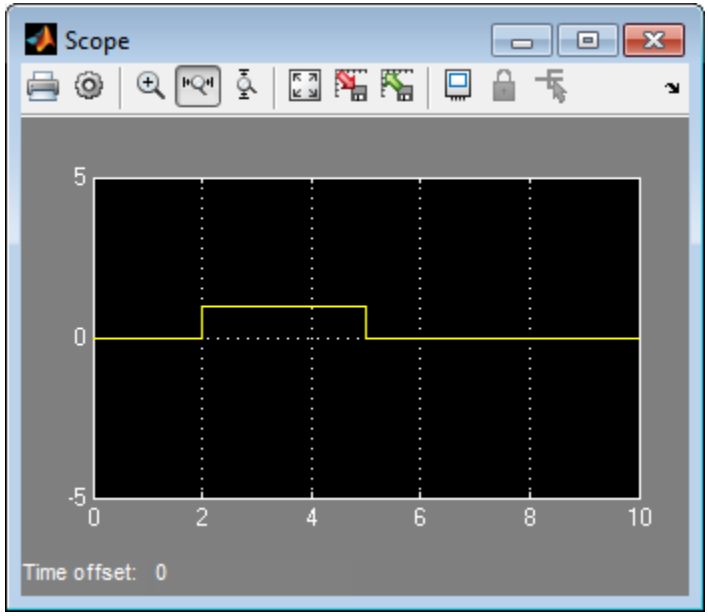
The coordinates for the origin of the moving test rectangle are represented by persistent data `x1` and `y1`, which are both initialized to `-1`. For the first sample, `x1` and `y1` both increase to `0`. From then on, the progression of rectangle arguments during simulation is as shown in the following graphic.



The fixed rectangle is shown in bold with a lower-left origin of `(2,4)` and a width and height of 2. At time `t = 0`, the first test rectangle has an origin of `(0,0)` and a width and height of 2. For each succeeding sample, the origin of the test rectangle increments by `(1,1)`. The rectangles at sample times `t = 2`, 3, and 4 intersect with the test rectangle.

The local function `rect_intersect` checks to see if its two rectangle arguments intersect. Each argument consists of coordinates for the lower-left corner of the rectangle (origin), and its width and height. `x` values for the left and right sides and `y` values for the top and bottom are calculated for each rectangle and compared in nested `if-else` decisions. The function returns a logical value of 1 if the rectangles intersect and 0 if they do not.

Scope output during simulation, which plots the return value against the sample time, confirms the intersecting rectangles for sample times 2, 3, and 4 .



After the simulation, the model coverage report appears in a browser window. After the summary in the report, the Details section of the model coverage report reports on each parts of the model.

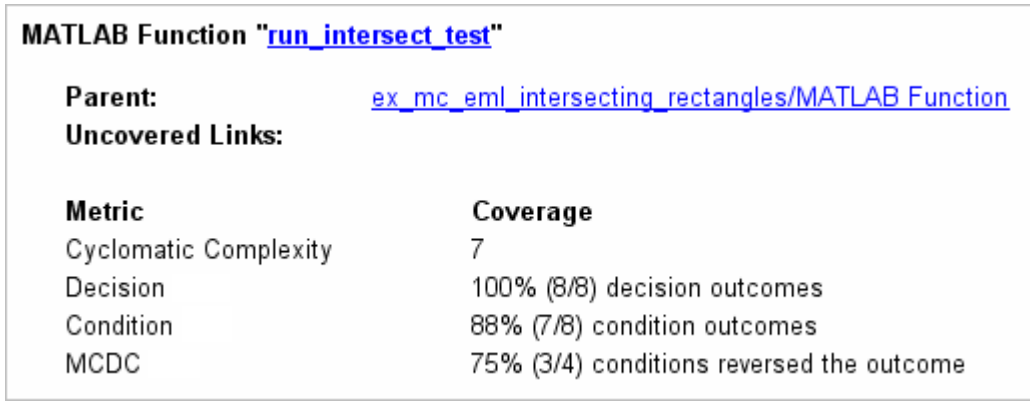
The model coverage report for the MATLAB Function block shows that the block itself has no decisions of its own apart from its function.

The following sections examine the model coverage report for the example model in reverse function-block-model order. Reversing the order helps you make sense of the summary information at the top of each section.

Coverage for the MATLAB Function run_intersect_test

Model coverage for the MATLAB Function block function `run_intersect_test` appears under the linked name of the function. Clicking this link opens the function in the editor.

Below the linked function name is a link to the model coverage report for the parent MATLAB Function block that contains the code for `run_intersect_test`.



MATLAB Function "run_intersect_test"

Parent: ex_mc_eml_intersecting_rectangles/MATLAB Function
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	7
Decision	100% (8/8) decision outcomes
Condition	88% (7/8) condition outcomes
MCDC	75% (3/4) conditions reversed the outcome

The top half of the report for the function summarizes its model coverage results. The coverage metrics for `run_intersect_test` include decision, condition, and MCDC coverage. You can best understand these metrics by examining the code for `run_intersect_test`.

```
 1  function out = run_intersect_test
 2  % Call rect_intersect to see if a moving test rectangle
 3  % and a stationary rectangle intersect
 4
 5  persistent x1 y1;
 6  if isempty(x1)
 7      x1 = -1, y1 = -1;
 8  end
 9
10  x1 = x1 + 1;
11  y1 = y1 + 1;
12  out = rect_intersect([x1 y1 2 2]', [2 4 2 2]');
13
14  function out = rect_intersect(rect1, rect2);
15  % Return 1 if two rectange arguments intersect, 0 if not.
16
17  left1 = rect1(1);
18  bottom1 = rect1(2);
19  right1 = left1 + rect1(3);
20  top1 = bottom1 + rect1(4);
21
22  left2 = rect2(1);
23  bottom2 = rect2(2);
24  right2 = left2 + rect2(3);
25  top2 = bottom2 + rect2(4);
26
27  if (top1 < bottom2 || top2 < bottom1)
28      out = 0;
29  else
30      if (right1 < left2 || right2 < left1)
31          out = 0;
32      else
33          out = 1;
34      end
35  end
```

Lines with coverage elements are marked by a highlighted line number in the listing:

- Line 1 receives decision coverage on whether the top-level function `run_intersect_test` is executed.
- Line 6 receives decision coverage for its `if` statement.
- Line 14 receives decision coverage on whether the local function `rect_intersect` is executed.

- Lines 27 and 30 receive decision, condition, and MCDC coverage for their `if` statements and conditions.

  Each of these lines is the subject of a report that follows the listing.

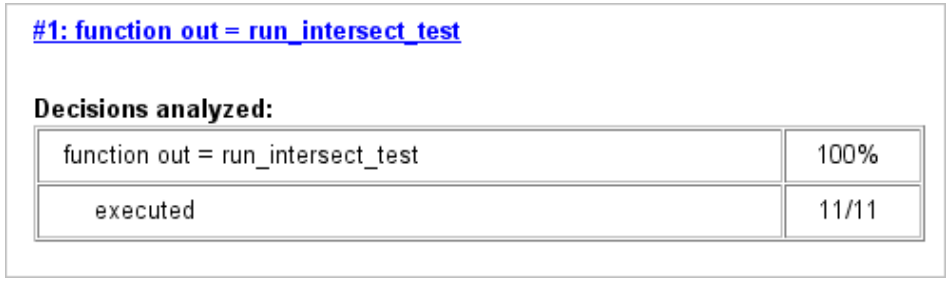
  The condition `right1 < left2` in line 30 is highlighted in red. This means that this condition was not tested for all of its possible outcomes during simulation. Exactly which of the outcomes was not tested is in the report for the decision in line 30.

The following sections display the coverage for each `run_intersect_test` decision line. The coverage for each line is titled with the line itself, which if clicked, opens the editor to the designated line.

Coverage for Line 1

The coverage metrics for line 1 are part of the coverage data for the function `run_intersect_test`.

The first line of every MATLAB function configured for code generation receives coverage analysis indicative of the decision to run the function in response to a call. Coverage for `run_intersect_test` indicates that it executed at least once during simulation.

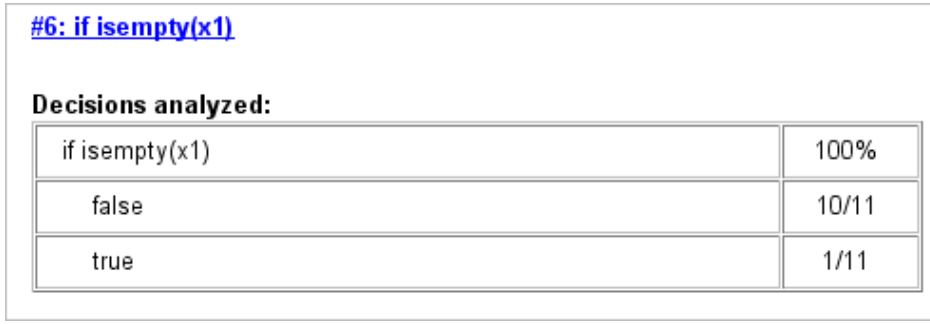

#1: function out = run_intersect_test

Decisions analyzed:

function out = run_intersect_test	100%
executed	11/11

Coverage for Line 6

The *Decisions analyzed* table indicates that the decision in line 6, `if isempty(x1)`, executed a total of eight times. The first time it executed, the decision evaluated to `true`, enabling `run_intersect_test` to initialize the values of its persistent data. The remaining seven times the decision executed, it evaluated to `false`. Because both possible outcomes occurred, decision coverage is 100%.



#6: if isempty(x1)

Decisions analyzed:

if isempty(x1)	100%
false	10/11
true	1/11

Coverage for Line 14

The *Decisions analyzed* table indicates that the local function `rect_intersect` executed during testing, thus receiving 100% coverage.

#14: function out = rect_intersect(rect1, rect2);

Decisions analyzed:

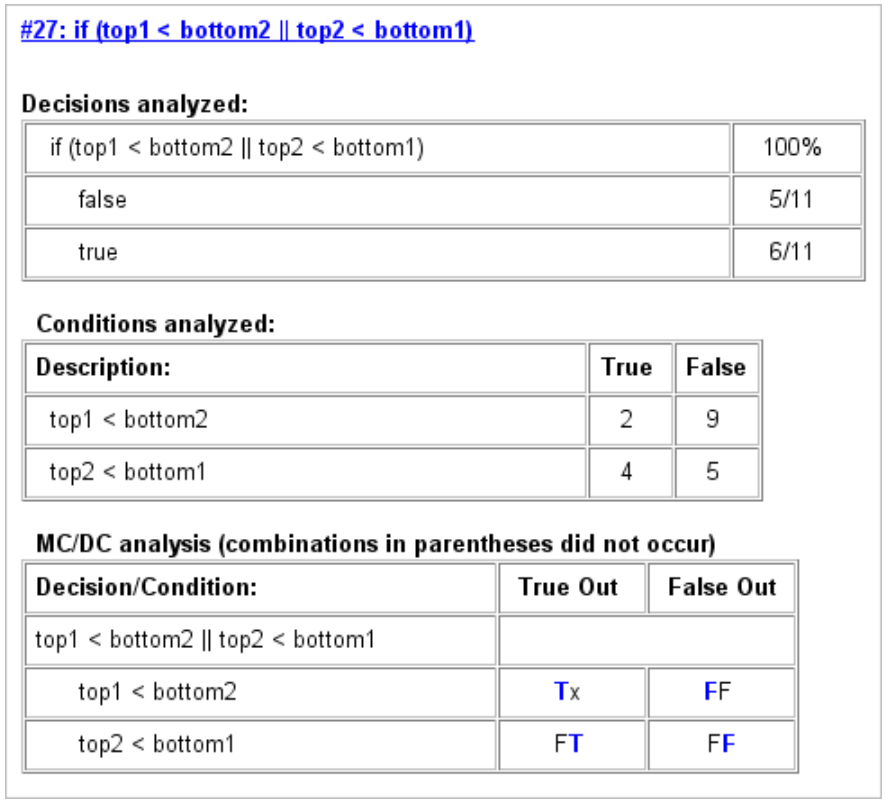
function out = rect_intersect(rect1, rect2);	100%
executed	11/11

Coverage for Line 27

The *Decisions analyzed* table indicates that there are two possible outcomes for the decision in line 27: `true` and `false`. Five of the eight times it was executed, the decision evaluated to `false`. The remaining three times, it evaluated to `true`. Because both possible outcomes occurred, decision coverage is 100%.

The *Conditions analyzed* table sheds some additional light on the decision in line 27. Because this decision consists of two conditions linked by a logical OR (`||`) operation, only one condition must evaluate `true` for the decision to be `true`. If the first condition evaluates to `true`, there is no need to evaluate the second condition. The first condition, `top1 < bottom2`, was evaluated eight times, and was `true` twice. This means that the second condition was evaluated only six times. In only one case was it `true`, which brings the total `true` occurrences for the decision to three, as reported in the *Decisions analyzed* table.

MCDC coverage looks for decision reversals that occur because one condition outcome changes from T to F or from F to T. The *MCDC analysis* table identifies all possible combinations of outcomes for the conditions that lead to a reversal in the decision. The character x is used to indicate a condition outcome that is irrelevant to the decision reversal. Decision-reversing condition outcomes that are not achieved during simulation are marked with a set of parentheses. There are no parentheses, therefore all decision-reversing outcomes occurred and MCDC coverage is complete for the decision in line 27.



#27: if (top1 < bottom2 || top2 < bottom1)

Decisions analyzed:

if (top1 < bottom2 || top2 < bottom1)	100%
false	5/11
true	6/11

Conditions analyzed:

Description:	True	False
top1 < bottom2	2	9
top2 < bottom1	4	5

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition:	True Out	False Out
top1 < bottom2 || top2 < bottom1		
top1 < bottom2	Tx	FF
top2 < bottom1	FT	FF

Coverage for Line 30

The line 30 decision, `if (right1 < left2 || right2 < left1)`, is nested in the `if` statement of the line 27 decision and is evaluated only if the line 27 decision is `false`. Because the line 27 decision evaluated `false` five times, line 30 is evaluated five times, three of which are `false`. Because both the `true` and `false` outcomes are achieved, decision coverage for line 30 is 100%.

Because line 30, like line 27, has two conditions related by a logical OR operator (`||`), condition 2 is tested only if condition 1 is `false`. Because condition 1 tests `false` five times, condition 2 is tested five times. Of these, condition 2 tests `true` two times and `false` three times, which accounts for the two occurrences of the `true` outcome for this decision.

Because the first condition of the line 30 decision does not test `true`, both outcomes do not occur for that condition and the condition coverage for the first condition is highlighted with a rose color. MCDC coverage is also highlighted in the same way for a decision reversal based on the `true` outcome for that condition.

#30: if (right1 < left2 || right2 < left1)

Decisions analyzed:

if (right1 < left2 || right2 < left1)	100%
false	3/5
true	2/5

Conditions analyzed:

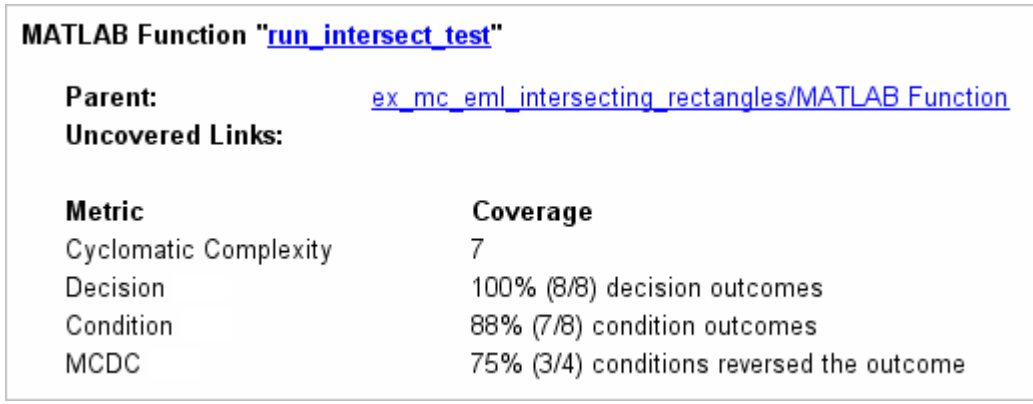
Description:	True	False
right1 < left2	0	5
right2 < left1	2	3

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition:	True Out	False Out
right1 < left2 || right2 < left1		
right1 < left2	(Tx)	FF
right2 < left1	FT	FF

Coverage for run_intersect_test

On the *Details* tab, the metrics that summarize coverage for the entire `run_intersect_test` function are reported and repeated as shown.



MATLAB Function "run_intersect_test"

Parent: ex_mc_eml_intersecting_rectangles/MATLAB Function
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	7
Decision	100% (8/8) decision outcomes
Condition	88% (7/8) condition outcomes
MCDC	75% (3/4) conditions reversed the outcome

The results summarized in the coverage metrics summary can be expressed in the following conclusions:

- There are eight decision outcomes reported for `run_intersect_test` in the line reports:
  - One for line 1 (executed)
  - Two for line 6 (`true` and `false`)
  - One for line 14 (executed)
  - Two for line 27 (`true` and `false`)
  - Two for line 30 (`true` and `false`).

  The decision coverage for each line shows 100% decision coverage. This means that decision coverage for `run_intersect_test` is eight of eight possible outcomes, or 100%.
- There are four conditions reported for `run_intersect_test` in the line reports. Lines 27 and 30 each have two conditions, and each condition has two condition outcomes (`true` and `false`), for a total of eight condition outcomes in `run_intersect_test`. All conditions tested positive for both the `true` and `false` outcomes except the first condition of line 30 (`right1 < left2`). This means that condition coverage for `run_intersect_test` is seven of eight, or 88%.

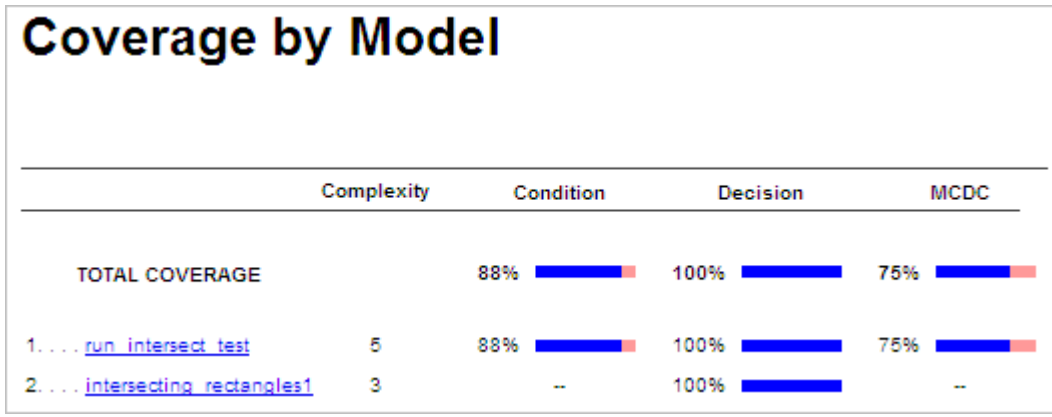
- The MCDC coverage tables for decision lines 27 and 30 each list two cases of decision reversal for each condition, for a total of four possible reversals. Only the decision reversal for a change in the evaluation of the condition `right1 < left2` of line 30 from `true` to `false` did not occur during simulation. This means that three of four, or 75% of the possible reversal cases were tested for during simulation, for a coverage of 75%.

Model Coverage for MATLAB Functions in an External File

Using the same model in “Model Coverage for MATLAB Function Blocks” on page 5-32, suppose the MATLAB functions `run_intersect_test` and `rect_intersect` are stored in an external MATLAB file named `run_intersect_test.m`.

To collect coverage for MATLAB functions in an external file, on the **Coverage** pane of the Configuration Parameters dialog box, select **Coverage for MATLAB files**.

After simulation, the model coverage report summary contains sections for the top-level model and for the external function.



The model coverage report for `run_intersect_test.m` reports the same coverage data as if the functions were stored in the MATLAB Function block.

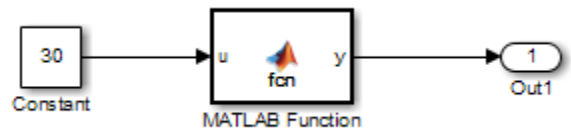
For a detailed example of a model coverage report for a MATLAB function in an external file, see “External MATLAB File Coverage Report” on page 6-5.

Model Coverage for Simulink Design Verifier MATLAB Functions

If the MATLAB code includes any of the following Simulink Design Verifier functions configured for code generation, you can measure coverage:

- `sldv.condition`
- `sldv.test`
- `sldv.assume`
- `sldv.prove`

For this example, consider the following model that contains a MATLAB Function block.



The MATLAB Function block contains the following code:

```
function y = fcn(u)
% This block supports MATLAB for code generation.

sldv.condition(u > -30)
sldv.test(u == 30)
y = 1;
```

To collect coverage for Simulink Design Verifier MATLAB functions, on the **Coverage** pane of the Configuration Parameters dialog box, under **Other metrics**, select **Objectives and Constraints**.

After simulation, the model coverage report listed coverage for the `sldv.condition` and `sldv.test` functions. For `sldv.condition`, the expression `u > -30` evaluated to `true` 51 times. For `sldv.test`, the expression `u == 30` evaluated to `true` 51 times.

eM Function "fcn"

Parent: ex_mc_eml_sldv_blocks/MATLAB Function

Metric	Coverage
Cyclomatic Complexity	1
Decision	100% (1/1) decision outcomes
Test Objective	100% (1/1) objective outcomes
Test Condition	100% (1/1) objective outcomes

```
1  function y = fcn(u)
2  % This block supports MATLAB for code generation.
3
4  sldv.condition(u > -30)
5  sldv.test(u == 30)
6  y = 1;
```

#1: function y = fcn(u)

Decisions analyzed:

function y = fcn(u)	100%
executed	51/51

#4: sldv.condition(u > -30)

Test Condition analyzed:

sldv.condition(u > -30)	51/51

#5: sldv.test(u == 30)

Test Objective analyzed:

sldv.test(u == 30)	51/51

For an example of model coverage data for Simulink Design Verifier blocks, see “Objectives and Constraints Coverage” on page 1-7.

Coverage for C and C++ S-Functions

When you record coverage for models containing supported C/C++ S-Functions, coverage is recorded for the code within the C/C++ S-Functions. The coverage results for S-Function blocks can be viewed in the same report as the rest of the model. For each S-Function block, the report links to a detailed coverage report for the C/C++ code in the block.

To generate coverage report for S-Functions:

1. When creating S-Functions, enable support for coverage.
2. When generating coverage report, enable support for S-Functions.

In this section...
“Make S-Function Compatible with Model Coverage” on page 5-46
“Generate Coverage Report for S-Function” on page 5-47

Make S-Function Compatible with Model Coverage

If you use the `legacy_code` function, S-Function Builder block or `mex` function to create your S-Functions, adapt your method appropriately to make the S-Function compatible with model coverage.

For more information on the three approaches, see “Creating C MEX S-Functions” (Simulink).

- “S-Function Using legacy_code Function” on page 5-46
- “S-Function Using S-Function Builder” on page 5-47
- “S-Function Using mex Function” on page 5-47

S-Function Using legacy_code Function

1. Initialize a MATLAB structure with fields that represent Legacy Code Tool properties.

   ```
   def = legacy_code('initialize')
   ```

2. To enable model coverage, turn on the option `def.Options.supportCoverage`.

   ```
   def.Options.supportCoverage = true;
   ```

3 Use the structure `def` in the usual way to generate an S-function. For an example, see “Coverage for S-Functions”.

S-Function Using S-Function Builder

1 Copy an instance of the S-Function Builder block from the **User-Defined Functions** library in the Library Browser into the your model.
2 Double-click the block to open the S-Function Builder dialog box.
3 On the **Build Info** tab, select **Enable support for coverage**.

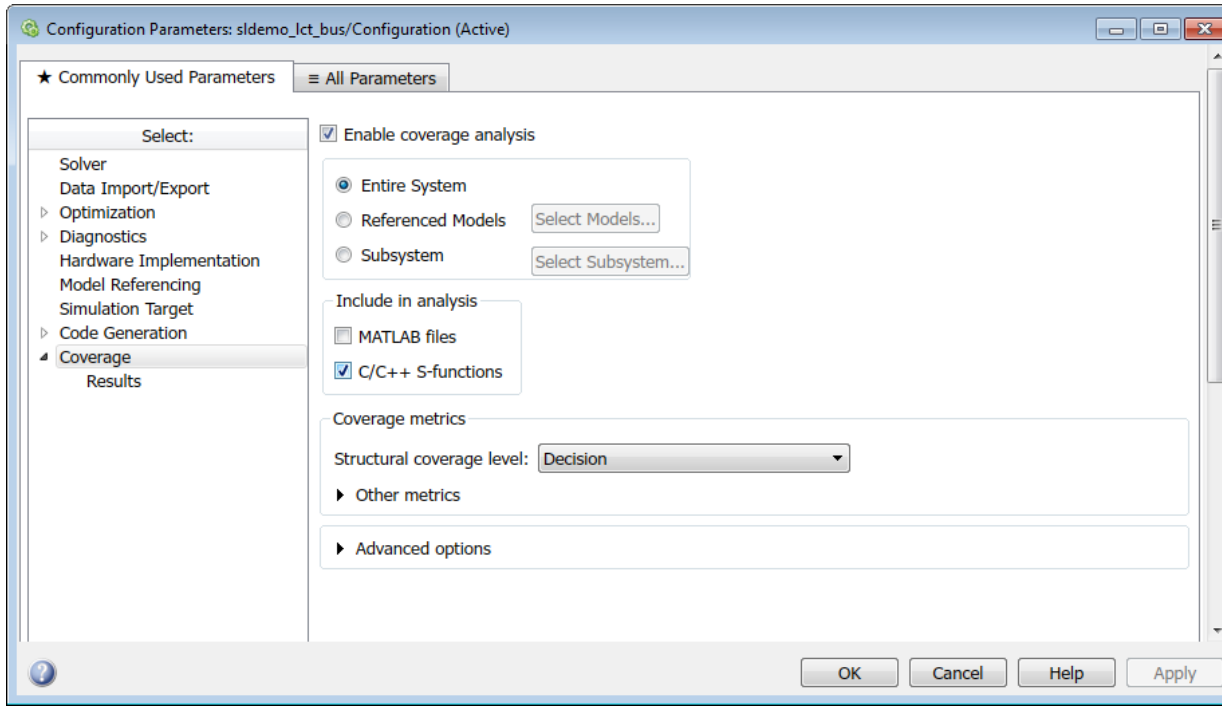
S-Function Using mex Function

If you use the `mex` function to compile and link your source files, use the `slcovmex` function instead. The `slcovmex` function compiles your source code and also makes it compatible with coverage.

This function has the same syntax and takes the same options as the `mex` function. In addition, you can provide some options relevant for model coverage. For more information, see `slcovmex`.

Generate Coverage Report for S-Function

1 Select **Analysis** > **Coverage** > **Settings**.
2 On the **Coverage** pane of the Configuration Parameters dialog box, select **C/C++ S-functions**.



When you run a simulation, the coverage report contains coverage metrics for C/C++ S-Function blocks in your model. For each S-Function block, the report links to a detailed coverage report for the C/C++ code in the block.

See Also

Related Examples

- “View Coverage Results for C/C++ Code in S-Function Blocks” on page 5-49

More About

- “C/C++ S-Function” on page 2-25

View Coverage Results for C/C++ Code in S-Function Blocks

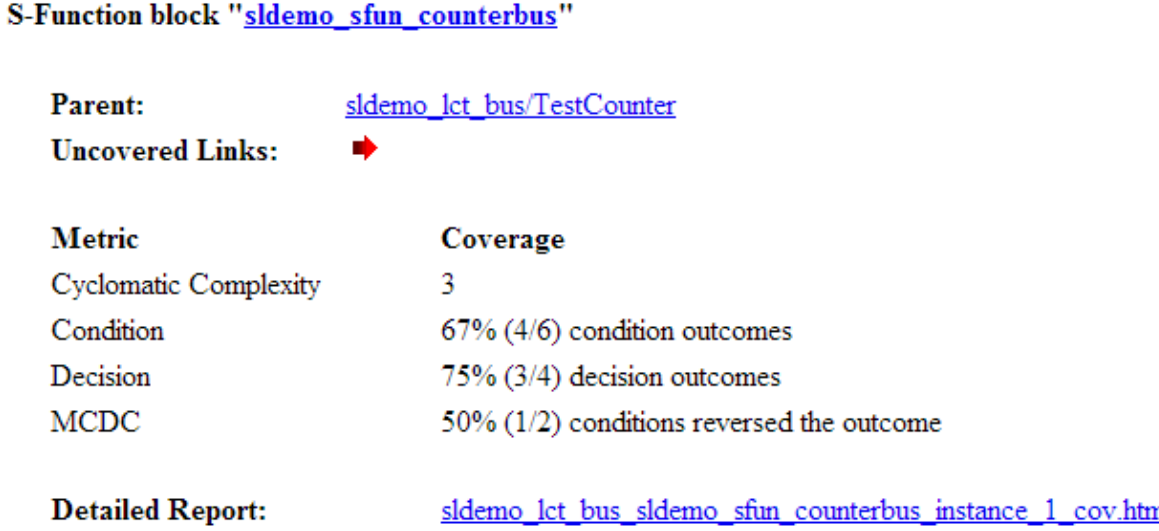
This example shows how to view coverage results for the C/C++ code in S-Function blocks in your model. To view coverage results for the C/C++ code in the blocks:

- Enable support for S-Function coverage. For more information, see “Coverage for C and C++ S-Functions” on page 5-46.
- Run simulation and view the coverage report.

  The coverage results for S-Function blocks can be viewed in the same report as the rest of the model. For each S-Function block, the report links to a detailed coverage report for the C/C++ code in the block.

To view the full code coverage report used in this example, follow the steps in “Coverage for S-Functions”.

1 In the coverage report, view the coverage metrics for the S-Function block.



S-Function block "sldemo_sfun_counterbus"

Parent:	sldemo_lct_bus/TestCounter
Uncovered Links:	

Metric	**Coverage**
Cyclomatic Complexity	3
Condition	67% (4/6) condition outcomes
Decision	75% (3/4) decision outcomes
MCDC	50% (1/2) conditions reversed the outcome

Detailed Report:	sldemo_lct_bus_sldemo_sfun_counterbus_instance_1_cov.html

For more information on the coverage report format, see “Top-Level Model Coverage Report” on page 6-12.

2. Select the **Detailed Report** link. The code coverage report for the S-Function block opens.
3. Select each of the links in **Table Of Contents** to navigate to various sections of the report.

Code Coverage Report for S-Function sldemo_sfun_counterbus

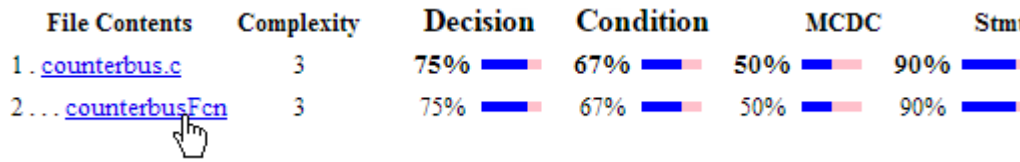
Table Of Contents

1. Analysis Information
2. Tests
3. Summary
4. Details
5. Code

Section Title	Purpose	
Analysis information	Contains information such as time when model was created and last modified, and file size.	
Tests	Contains information about the simulation such as start and end time.	
Summary	Contains coverage information about the files and functions in the S-Function block. For each file and function, the percentage coverage is displayed. The coverage types relevant for the code are the following:	
	Coverage Type	**Label**
	"Cyclomatic Complexity for Code Coverage" on page 4-5	**Complexity**
	"Condition Coverage for Code Coverage" on page 4-3	**Condition**.
	"Decision Coverage for Code Coverage" on page 4-3	**Decision**

Section Title	Purpose	
	"Modified Condition/ Decision Coverage (MCDC) for Code Coverage" on page 4-4	**MCDC**
	"Relational Boundary for Code Coverage" on page 4-5	**Relational Boundary**
	Percentage of statements covered	**Stmt**
Details	Contains coverage information about the statements that receive condition, decision or MCDC coverage. The information is grouped by file and function.	
Code	Contains the C/C++ code. Statements that are not covered are highlighted in pink.	

4 In the **Summary** section, select each file or function name to see details of coverage for statements in the file or function.



5 The condition, decision or MCDC outcomes that were not tested during simulation are highlighted in pink. Within the details for a file or function, scroll down to note these cases and investigate them further.

2.1 Decision/Condition `(u1->limits.upper_saturation_limit >= limit) && inputGElower`

Function: `counterbusFcn` (line 6)

Metric	Coverage
Decision	100% (2/2) decision outcomes
Condition	75% (3/4) condition outcomes
MCDC	50% (1/2) conditions reversed the outcome

Decisions analyzed:

`(u1->limits.upper_saturation_limit >= limit) && inputGElower`	100%
false	61/201
true	140/201

Conditions analyzed:

Description:	True	False
`u1->limits.upper_saturation_limit >= limit`	140	61
`inputGElower`	140	0

6 To obtain an overview of the statements that were not covered, navigate to the **Code** section. This section contains your code with the uncovered statements highlighted in pink.

Code

```
1  /* Copyright 2005-2006 The MathWorks, Inc. */
2
3
4  #include "counterbus.h"
5
6  void counterbusFcn(COUNTERBUS *u1, int32_T u2, COUNTERBUS *y1, int32_T *y2)
7  {
8    int32_T limit;
9    boolean_T inputGElower;
10
11   limit = u1->inputsignal.input + u2;
12
13   inputGElower = (limit >= u1->limits.lower_saturation_limit);
14
15   if((u1->limits.upper_saturation_limit >= limit) && inputGElower) {
16     *y2 = limit;
17   } else {
18
19     if(inputGElower) {
20       limit = u1->limits.upper_saturation_limit;
21     } else {
22       limit = u1->limits.lower_saturation_limit;
23     }
24     *y2 = limit;
25   }
26
27   y1->inputsignal.input = *y2;
28   y1->limits = u1->limits;
29
30 }
31
```

See Also

More About

- “C/C++ S-Function” on page 2-25

Model Coverage for Stateflow Charts

In this section...
“How Model Coverage Reports Work for Stateflow Charts” on page 5-54
“Specify Coverage Report Settings for Stateflow Charts” on page 5-55
“Cyclomatic Complexity for Stateflow Charts” on page 5-55
“Decision Coverage for Stateflow Charts” on page 5-56
“Condition Coverage for Stateflow Charts” on page 5-59
“MCDC Coverage for Stateflow Charts” on page 5-60
“Relational Boundary Coverage for Stateflow Charts” on page 5-60
“Simulink Design Verifier Coverage for Stateflow Charts” on page 5-60
“Model Coverage Reports for Stateflow Charts” on page 5-62
“Model Coverage for Stateflow State Transition Tables” on page 5-71
“Model Coverage for Stateflow Atomic Subcharts” on page 5-72
“Model Coverage for Stateflow Truth Tables” on page 5-75
“Colored Stateflow Chart Coverage Display” on page 5-80

How Model Coverage Reports Work for Stateflow Charts

To generate a Model Coverage report, select **Analysis > Coverage > Settings** and specify the desired options on the **Coverage > Results** pane of the **Coverage** pane of the Configuration Parameters dialog box. For Stateflow charts, the Simulink Coverage software records the execution of the chart itself and the execution of states, transition decisions, and individual conditions that compose each decision. After simulation ends, the model coverage reports on how thoroughly a model was tested. The report shows:

- How many times each exclusive substate is executed or exited from its parent superstate and entered due to parent superstate history
- How many times each transition decision has been evaluated as true or false
- How many times each condition has been evaluated as true or false

---

Note To measure model coverage data for a Stateflow chart, you must:

- Have a Stateflow license.
- Have debugging/animation enabled for the chart.

Specify Coverage Report Settings for Stateflow Charts

To specify coverage recording settings, select **Analysis > Coverage > Settings** in the Simulink Editor. Then select **Enable coverage analysis**.

By selecting the **Generate report automatically after analysis** option in the **Coverage > Results** pane of the Configuration Parameters dialog box, you can create an HTML report containing the coverage data generated during simulation of the model. The report appears in the MATLAB Help browser at the end of simulation.

Enabling coverage analysis also enables the selection of different coverages that you can specify for your reports. The following sections address only coverage metrics that affect reports for Stateflow charts. These metrics include decision coverage, condition coverage, and MCDC coverage.

Cyclomatic Complexity for Stateflow Charts

Cyclomatic complexity is a measure of the complexity of a software module based on its edges, nodes, and components within a control-flow chart. It provides an indication of how many times you need to test the module.

The calculation of cyclomatic complexity is as follows:

```
CC = E - N + p
```

where `CC` is the cyclomatic complexity, `E` is the number of edges, `N` is the number of nodes, and `p` is the number of components.

Within the Model Coverage tool, each decision is exactly equivalent to a single control flow node, and each decision outcome is equivalent to a control flow edge. Any additional structure in the control-flow chart is ignored since it contributes the same number of nodes as edges and therefore has no effect on the complexity calculation. Therefore, you can express cyclomatic complexity as follows:

```
CC = OUTCOMES - DECISIONS + p
```

For analysis purposes, each chart counts as a single component.

Decision Coverage for Stateflow Charts

Decision coverage interprets a model execution in terms of underlying decisions where behavior or execution must take one outcome from a set of mutually exclusive outcomes.

Note Full coverage for an object of decision means that every decision has had at least one occurrence of each of its possible outcomes.

Decisions belong to an object making the decision based on its contents or properties. The following table lists the decisions recorded for model coverage for the Stateflow objects owning them. The sections that follow the table describe these decisions and their possible outcomes.

Object	Possible Decisions
Chart	If a chart is a triggered Simulink block, it must decide whether or not to execute its block. If a chart contains exclusive (OR) substates, it must decide which of its states to execute.
State	If a state is a superstate containing exclusive (OR) substates, it must decide which substate to execute. If a state has `on` *`event name`* actions (which might include temporal logic operators), the state must decide whether or not to execute the actions.
Transition	If a transition is a conditional transition, it must decide whether or not to exit its active source state or junction and enter another state or junction.

Chart as a Triggered Simulink Block Decision

If the chart is a triggered block in a Simulink model, the decision to execute the block is tested. If the block is not triggered, there is no decision to execute the block, and the measurement of decision coverage is not applicable (`NA`).

Chart Containing Exclusive OR Substates Decision

If the chart contains exclusive (OR) substates, the decision on which substate to execute is tested. If the chart contains only parallel AND substates, this coverage measurement is not applicable (NA).

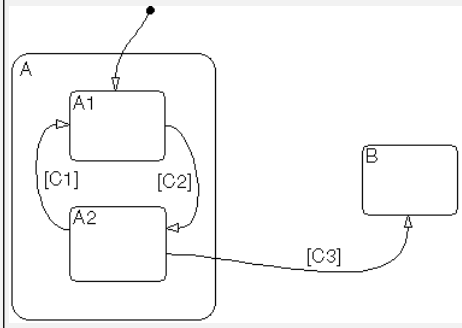
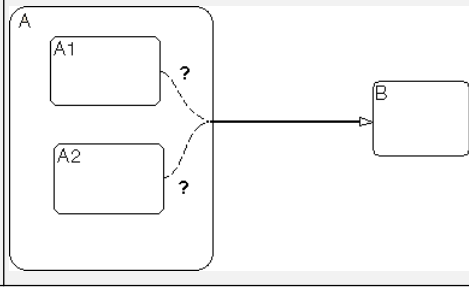
Superstate Containing Exclusive OR Substates Decision

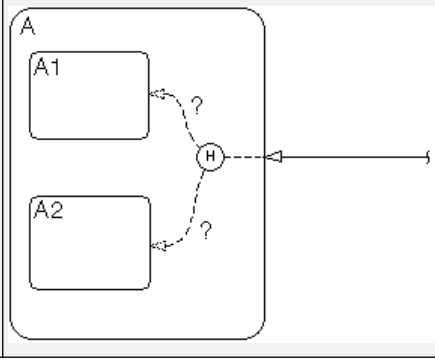
Since a chart is hierarchically processed from the top down, procedures such as exclusive (OR) substate entry, exit, and execution are sometimes decided by the parenting superstate.

Note Decision coverage for superstates applies only to exclusive (OR) substates. A superstate makes no decisions for parallel (AND) substates.

Since a superstate must decide which exclusive (OR) substate to process, the number of decision outcomes for the superstate is the number of exclusive (OR) substates that it contains. In the examples that follow, the choice of which substate to process can occur in one of three possible contexts.

Note Implicit transitions appear as dashed lines in the following examples.

Context	Example	Decisions That Occur
Active call	States A and A1 are active. 	• The parent of states A and B must decide which of these states to process. This decision belongs to the parent. Since A is active, it is processed. • State A, the parent of states A1 and A2, must decide which of these states to process. This decision belongs to state A. Since A1 is active, it is processed. During processing of state A1, all outgoing transitions are tested. This decision belongs to the transition and not to the parent state A. In this case, the transition marked by condition C2 is tested and a decision is made whether to take the transition to A2 or not.
Implicit substate exit	A transition takes place whose source is superstate A and whose destination is state B. 	If the superstate has two exclusive (OR) substates, it is the decision of superstate A which substate performs the implicit transition from substate to superstate.

Context	Example	Decisions That Occur
Substate entry with a history junction	A history junction records which substate was last active before the superstate was exited. 	If that superstate becomes the destination of one or more transitions, the history junction decides which previously active substate to enter.

For more information, see “State Details Report Section” on page 5-65.

State with On Event_Name Action Statement Decision

A state that has an `on` *`event_name`* action statement must decide whether to execute that statement based on the reception of a specified event or on an accumulation of the specified event when using temporal logic operators.

Conditional Transition Decision

A conditional transition is a transition with a triggering event and/or a guarding condition. In a conditional transition from one state to another, the decision to exit one state and enter another is credited to the transition itself.

Note Only conditional transitions receive decision coverage. Transitions without decisions are not applicable to decision coverage.

Condition Coverage for Stateflow Charts

Condition coverage reports on the extent to which all possible outcomes are achieved for individual subconditions composing a transition decision or for logical expressions in assignment statements in states and transitions.

For example, for the decision [A & B & C] on a transition, condition coverage reports on the true and false occurrences of each of the subconditions A, B, and C. This results in eight possible outcomes: true and false for each of three subconditions.

Outcome	A	B	C
1	T	T	T
2	T	T	F
3	T	F	T
4	T	F	F
5	F	T	T
6	F	T	F
7	F	F	T
8	F	F	F

For more information, see “Transition Details Report Section” on page 5-68.

MCDC Coverage for Stateflow Charts

The Modified Condition Decision/Coverage (MCDC) option reports a test's coverage of occurrences in which changing an individual subcondition within a logical expression results in changing the entire expression from true to false or false to true.

For example, if a transition executes on the condition [`C1 & C2 & C3 | C4 & C5`], the MCDC report for that transition shows actual occurrences for each of the five subconditions (`C1, C2, C3, C4, C5`) in which changing its result from true to false is able to change the result of the entire condition from true to false.

Relational Boundary Coverage for Stateflow Charts

If a transition in a Stateflow chart involves a relational operation, it receives relational boundary coverage. For more information, see “Relational Boundary Coverage” on page 1-8.

Simulink Design Verifier Coverage for Stateflow Charts

You can use the following Simulink Design Verifier functions inside Stateflow charts:

- `sldv.condition`
- `sldv.test`
- `sldv.assume`
- `sldv.prove`

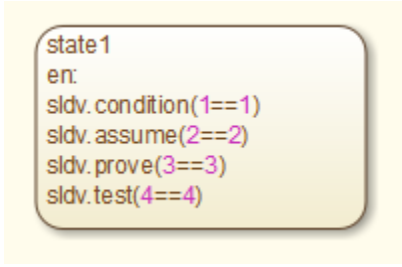
If you do not have a Simulink Design Verifier license, you can collect model coverage for a Stateflow chart containing these functions, but you cannot analyze the model using the Simulink Design Verifier software.

When you specify the **Objectives and Constraints** coverage metric in the **Coverage** pane of the Configuration Parameters dialog box, the Simulink Coverage software records coverage for these functions.

Each of these functions evaluates an expression *`expr`*, for example, `sldv.test(`*`expr`*`)`, where *`expr`* is any valid Boolean MATLAB expression. Simulink Design Verifier coverage measures the number of time steps that the expression *`expr`* evaluates to `true`.

If *`expr`* is `true` for at least one time step, Simulink Design Verifier coverage for that function is 100%. Otherwise, the Simulink Coverage software reports coverage for that function as 0%.

Consider a model that contains this Stateflow chart:



To collect coverage for Simulink Design Verifier functions, on the **Coverage** pane of the Configuration Parameters dialog box, select **Objectives and Constraints**.

After simulation, the model coverage report lists coverage for the `sldv.condition`, `sldv.assume`, `sldv.prove`, and `sldv.test` functions.

Metric	Coverage
Cyclomatic Complexity	0
Proof Assumption	100% (1/1) objective outcomes
Test Condition	100% (1/1) objective outcomes
Proof Objective	100% (1/1) objective outcomes
Test Objective	100% (1/1) objective outcomes

Proof Assumption analyzed:

sldv.assume(2==2)	1/1

Test Condition analyzed:

sldv.condition(1==1)	1/1

Proof Objective analyzed:

sldv.prove(3==3)	1/1

Test Objective analyzed:

sldv.test(4==4)	1/1

Model Coverage Reports for Stateflow Charts

- “Summary Report Section” on page 5-62
- “Subsystem and Chart Details Report Sections” on page 5-63
- “State Details Report Section” on page 5-65
- “Transition Details Report Section” on page 5-68

The following sections of a Model Coverage report were generated by simulating the `sf_boiler` model, which includes the Bang-Bang Controller chart. The coverage metrics for **MCDC** are enabled for this report.

Summary Report Section

The Summary section shows coverage results for the entire test and appears at the beginning of the Model Coverage report.

Summary

Model Hierarchy/Complexity:		Test 1		
		D1	C1	MCDC
1. sf_boiler	**20**	89%	71%	43%
2. . . . Bang-Bang Controller	**16**	95%	71%	43%
3. . . . . SF: Bang-Bang Controller	15	95%	71%	43%
4. . . . . SF: Heater	12	94%	71%	43%
5. . . . . SF: Off	2	100%	75%	50%
6. . . . . SF: On	4	88%	NA	NA
7. . . . . SF: flash_LED	1	100%	NA	NA
8. . . . . SF: turn_boiler	1	100%	NA	NA
9. . . . Boiler Plant model	3	67%	NA	NA
10. . . . . digital thermometer	2	50%	NA	NA
11. . . . . ADC	2	50%	NA	NA

Each line in the hierarchy summarizes the coverage results at that level and the levels below it. You can click a hyperlink to a later section in the report with the same assigned hierarchical order number that details that coverage and the coverage of its children.

The top level, `sf_boiler`, is the Simulink model itself. The second level, Bang-Bang Controller, is the Stateflow chart. The next levels are superstates within the chart, in order of hierarchical containment. Each superstate uses an SF: prefix. The bottom level, Boiler Plant model, is an additional subsystem in the model.

Subsystem and Chart Details Report Sections

When recording coverage for a Stateflow chart, the Simulink Coverage software reports two types of coverage for the chart—Subsystem and Chart.

- *Subsystem* — This section reports coverage for the chart:
  - *Coverage (this object)*: Coverage data for the chart as a container object
  - *Coverage (inc.) descendants*: Coverage data for the chart and the states and transitions in the chart.

  If you click the hyperlink of the subsystem name in the section title, the Bang-Bang Controller block is highlighted in the block diagram.

  Decision coverage is not applicable (`NA`) because this chart does not have an explicit trigger. Condition coverage and MCDC are not applicable (`NA`) for a chart, but apply to its descendants.

2. SubSystem block "Bang-Bang Controller"

Parent: /sf_boiler
Child Systems: Bang-Bang Controller

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	**1**	**16**
Condition (C1)	NA	71% (10/14) condition outcomes
Decision (D1)	NA	95% (21/22) decision outcomes
MCDC (C1)	NA	43% (3/7) conditions reversed the outcome

- *Chart* — This section reports coverage for the chart:
  - *Coverage (this object)*: Coverage data for the chart and its inputs
  - *Coverage (inc.) descendants*: Coverage data for the chart and the states and transitions in the chart.

  If you click the hyperlink of the chart name in the section title, the chart opens in the Stateflow Editor.

  Decision coverage is listed appears for the chart and its descendants. Condition coverage and MCDC are not applicable (`NA`) for a chart, but apply to its descendants.

3. Chart "Bang-Bang Controller"

Parent: sf_boiler/Bang-Bang Controller
Child Systems: Heater, flash_LED, turn_boiler

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	15
Condition (C1)	NA	71% (10/14) condition outcomes
Decision (D1)	100% (2/2) decision outcomes	95% (21/22) decision outcomes
MCDC (C1)	NA	43% (3/7) conditions reversed the outcome

Decisions analyzed:

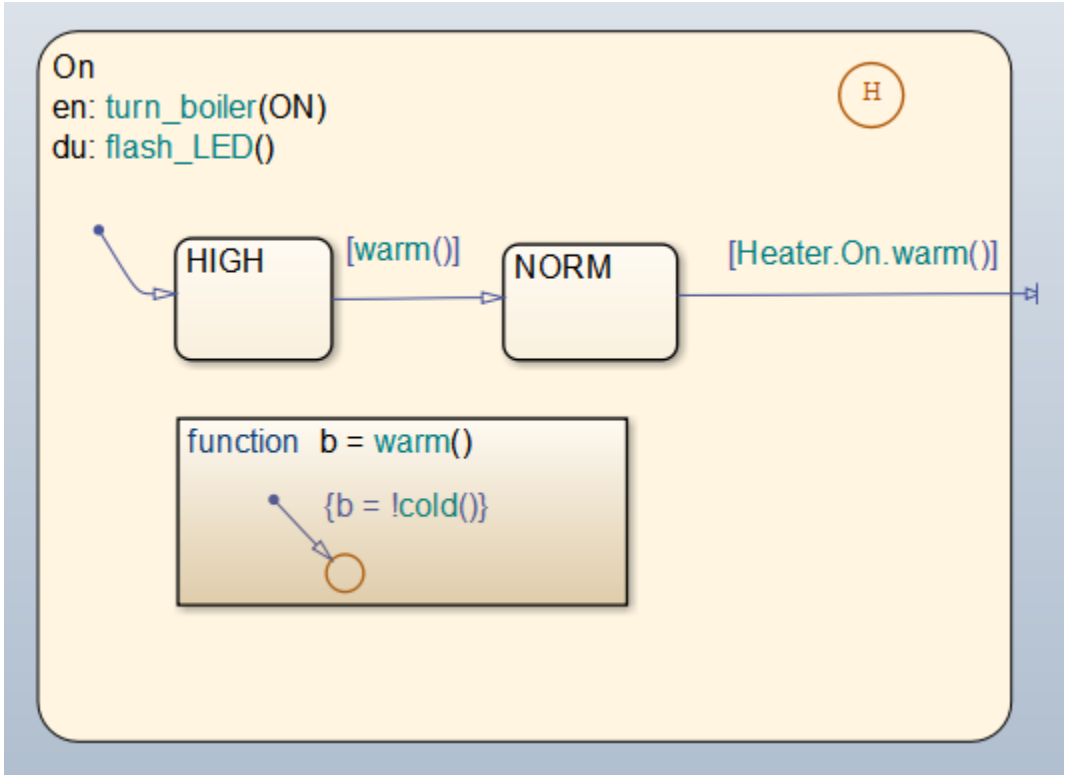
Substate executed	100%
State "Off"	1160/1400
State "On"	240/1400

State Details Report Section

For each state in a chart, the coverage report includes a *State* section with details about the coverage recorded for that state.

In the `sf_boiler` model, the state `On` resides in the box `Heater`. `On` is a superstate that contains:

- Two substates `HIGH` and `NORM`
- A history junction
- The function `warm`



The coverage report includes a *State* section on the state `On`.

6. State "On"

Parent: sf_boiler/Bang-Bang Controller.Heater
Uncovered Links:

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	3	4
Decision (D1)	83% (5/6) decision outcomes	88% (7/8) decision outcomes

Decisions analyzed:

Substate executed	100%
State "HIGH"	150/233
State "NORM"	83/233
Substate exited when parent exits	50%
State "HIGH"	7/7
State "NORM"	0/7
Previously active substate entered due to history	100%
State "HIGH"	7/28
State "NORM"	21/28

The decision coverage for the On state tests the decision of which substate to execute.

The three decisions are listed in the report:

- Under *Substate executed*, which substate to execute when On executes.

- Under *Substate exited when parent exited*, which substate is active when `On` exits. `NORM` is listed as never being active when `On` exits because the coverage tool sees the supertransition from `NORM` to `Off` as a transition from `On` to `Off`.
- Under *Previously active substate entered due to history*, which substate to reenter when `On` re-executes. The history junction records the previously active substate.

Because each decision can result in either `HIGH` or `NORM`, the total possible outcomes are $3 \times 2 = 6$. The results indicate that five of six possible outcomes were tested during simulation.

Cyclomatic complexity and decision coverage also apply to descendants of the `On` state. The decision required by the condition `[warm()]` for the transition from HIGH to NORM brings the total possible decision outcomes to 8. Condition coverage and MCDC are not applicable (`NA`) for a state.

Note Nodes and edges that make up the cyclomatic complexity calculation have no direct relationship with model objects (states, transitions, and so on). Instead, this calculation requires a graph representation of the equivalent control flow.

Transition Details Report Section

Reports for transitions appear under the report sections of their owning objects. Transitions do not appear in the model hierarchy of the Summary section, since the hierarchy is based on superstates that own other Stateflow objects.

Transition "after(40,sec) [cold()]" from "Off" to "On"

Parent: sf_boiler/Bang-Bang Controller.Heater
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	3
Condition (C1)	67% (4/6) condition outcomes
Decision (D1)	100% (2/2) decision outcomes
MCDC (C1)	33% (1/3) conditions reversed the outcome

Decisions analyzed:

Transition trigger expression	100%
false	1131/1160
true	29/1160

Conditions analyzed:

Description:	True	False
Condition 1, "sec"	1160	0
Condition 2, "after(40,sec)"	29	1131
Condition 3, "cold()"	29	0

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition:	True Out	False Out
Transition trigger expression		
Condition 1, "sec"	TTT	(Fxx)
Condition 2, "after(40,sec)"	TTT	TFx
Condition 3, "cold()"	TTT	(TTF)

The decision for this transition depends on the time delay of 40 seconds and the condition `[cold()]`. If, after a 40 second delay, the environment is cold (`cold() = 1`), the

decision to execute this transition and turn the Heater on is made. For other time intervals or environment conditions, the decision is made not to execute.

For decision coverage, both true and false outcomes occurred. Because two of two decision outcomes occurred, coverage was full or 100%.

Condition coverage shows that only 4 of 6 condition outcomes were tested. The temporal logic statement `after(40,sec)` represents two conditions: the occurrence of `sec` and the time delay `after(40,sec)`. Therefore, three conditions on the transition exist: `sec`, `after(40,sec)`, and `cold()`. Since each of these decisions can be true or false, six possible condition outcomes exist.

The **Conditions analyzed** table shows each condition as a row with the recorded number of occurrences for each outcome (true or false). Decision rows in which a possible outcome did not occur are shaded. For example, the first and the third rows did not record an occurrence of a false outcome.

In the MCDC report, all sets of occurrences of the transition conditions are scanned for a particular pair of decisions for each condition in which the following are true:

- The condition varies from true to false.
- All other conditions contributing to the decision outcome remain constant.
- The outcome of the decision varies from true to false, or the reverse.

For three conditions related by an implied AND operator, these criteria can be satisfied by the occurrence of these conditions.

Condition Tested	True Outcome	False Outcome
1	**T**TT	**F**xx
2	T**T**T	T**F**x
3	TT**T**	TT**F**

Notice that in each line, the condition tested changes from true to false while the other condition remains constant. Irrelevant contributors are coded with an "x" (discussed below). If both outcomes occur during testing, coverage is complete (100%) for the condition tested.

The preceding report example shows coverage only for condition 2. The false outcomes required for conditions 1 and 3 did not occur, and are indicated by parentheses for both

conditions. Therefore, condition rows 1 and 3 are shaded. While condition 2 has been tested, conditions 1 and 3 have not and MCDC is 33%.

For some decisions, the values of some conditions are irrelevant under certain circumstances. For example, in the decision [`C1 & C2 & C3 | C4 & C5`] the left side of the `|` is false if any one of the conditions `C1`, `C2`, or `C3` is false. The same applies to the right side result if either `C4` or `C5` is false. When searching for matching pairs that change the outcome of the decision by changing one condition, holding some of the remaining conditions constant is irrelevant. In these cases, the MCDC report marks these conditions with an "x" to indicate their irrelevance as a contributor to the result. These conditions appear as shown.

Transition "[c1&c2&c3 | c4&c5]" . . .

⋮

MC/DC analysis (combinations in parentheses did not occur)

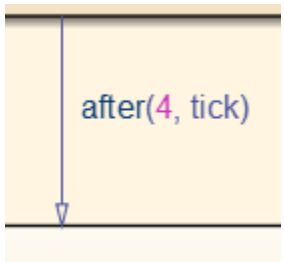
Decision/Condition:	#1 True Out	#1 False Out
Transition trigger expression		
Condition 1, "c1"	TTTxx	FxxFx
Condition 2, "c2"	TTTxx	TFxFx
Condition 3, "c3"	TTTxx	TTFFx
Condition 4, "c4"	FxxTT	FxxFx
Condition 5, "c5"	FxxTT	FxxTF

Consider the first matched pair. Since condition 1 is true in the **True** outcome column, it must be false in the matching **False** outcome column. This makes the conditions `C2` and `C3` irrelevant for the false outcome since `C1 & C2 & C3` is always false if `C1` is false. Also, since the false outcome is required to evaluate to false, the evaluation of `C4 & C5` must also be false. In this case, a match was found with `C4` = F, making condition `C5` irrelevant.

Model Coverage for Stateflow State Transition Tables

State transition tables are an alternative way of expressing modal logic in Stateflow. Stateflow charts represent modal logic graphically, and state transition tables can represent equivalent modal logic in tabular form. For more information, see “Tabular Expression of Modal Logic” (Stateflow).

Coverage results for state transition tables are the same as coverage results for equivalent Stateflow charts, except for a slight difference that arises in coverage of temporal logic. For example, consider the temporal logic expression `after(4, tick)` in the Mode Logic chart of the `slvnvdemo_covfilt` example model.



In chart coverage, the `after(4, tick)` transition represents two conditions: the occurrence of `tick` and the time delay `after(4, tick)`. Since the temporal event `tick` is never false, the first condition is not satisfiable, and you cannot record 100% condition and MCDC coverage for the transition `after(4, tick)`.

In state transition table coverage, the `after(4, tick)` transition represents a single decision, with no subcondition for the occurrence of `tick`. Therefore, only decision coverage is recorded.

For state transition tables containing temporal logic decisions, as in the above example, condition coverage and MCDC is not recorded.

Model Coverage for Stateflow Atomic Subcharts

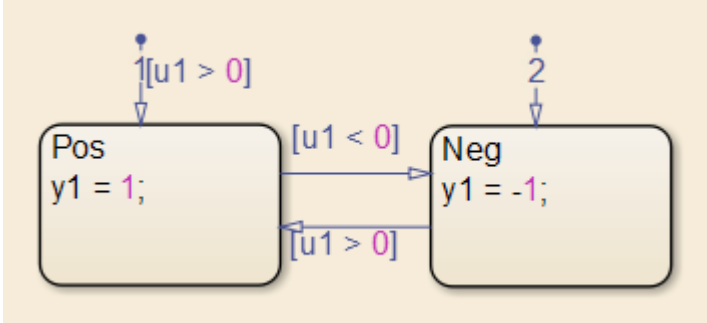
In a Stateflow chart, an atomic subchart is a graphical object that allows you to reuse the same state or subchart across multiple charts and models.

When you specify to record coverage data for a model during simulation, the Simulink Coverage software records coverage for any atomic subcharts in your model. The coverage data records the execution of the chart itself, and the execution of states, transition decisions, and individual conditions that compose each decision in the atomic subchart.

Simulate the `doc_atomic_subcharts_map_iodata` example model and record decision coverage:

1. Open the `doc_atomic_subcharts_map_iodata` model.

   This model contains two Sine Wave blocks that supply input signals to the Stateflow chart. Chart contains two atomic subcharts—A and B—that are linked from the same library chart, also named A. The library chart contains the following objects:



2. In the Simulink Editor, select **Analysis > Coverage > Settings**

   The **Coverage** pane of the Configuration Parameters dialog box appears.
3. Select **Enable coverage analysis** and then select **Entire System**.
4. On the **Coverage > Results** pane, select **Generate report automatically after analysis**.
5. Click **OK** to close the Configuration Parameters dialog box.
6. Simulate the `doc_atomic_subcharts_map_iodata` model.

   When the simulation completes, the coverage report opens.

The report provides coverage data for atomic subcharts A and B in the following forms:

- For the atomic subchart instance and its contents. Decision coverage is not applicable (`NA`) because this chart does not have an explicit trigger.

4. Atomic Subchart "A"

Parent:	doc_atomic_subcharts_map_iodata/Chart
Child Systems:	A

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	4
Decision (D1)	NA	88% (7/8) decision outcomes

- For the library chart A and its contents. The chart itself achieves 100% coverage on the input `u1`, and 88% coverage on the states and transitions inside the library chart.

5. Chart "A"

Parent:	doc_atomic_subcharts_map_iodata/Chart.A

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	4
Decision (D1)	100% (2/2) decision outcomes	88% (7/8) decision outcomes

Decisions analyzed:

Substate executed	100%
State "Neg"	4/10
State "Pos"	6/10

Atomic subchart B is a copy of the same library chart A. The coverage of the contents of subchart B is identical to the coverage of the contents of subchart A.

Model Coverage for Stateflow Truth Tables

- “Types of Coverage in Stateflow Truth Tables” on page 5-75
- “Analyze Coverage in Stateflow Truth Tables” on page 5-75

Types of Coverage in Stateflow Truth Tables

Simulink Coverage software reports model coverage for the decisions the objects make in a Stateflow chart during model simulation. The report includes coverage for the decisions the truth table functions make.

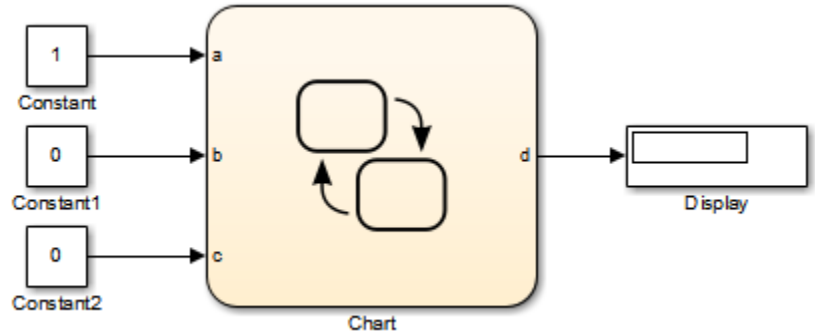
For this type of truth table...	The report includes coverage data for...
Stateflow Classic	Conditions only.
MATLAB	Conditions and only those actions that have decision points. **Note** With the MATLAB for code generation action language, you can specify decision points in actions using control flow constructs, such as loops and switch statements.

Note To measure model coverage data for a Stateflow truth table, you must have a Stateflow license. For more information about Stateflow truth tables, see “Decision Logic” (Stateflow) .

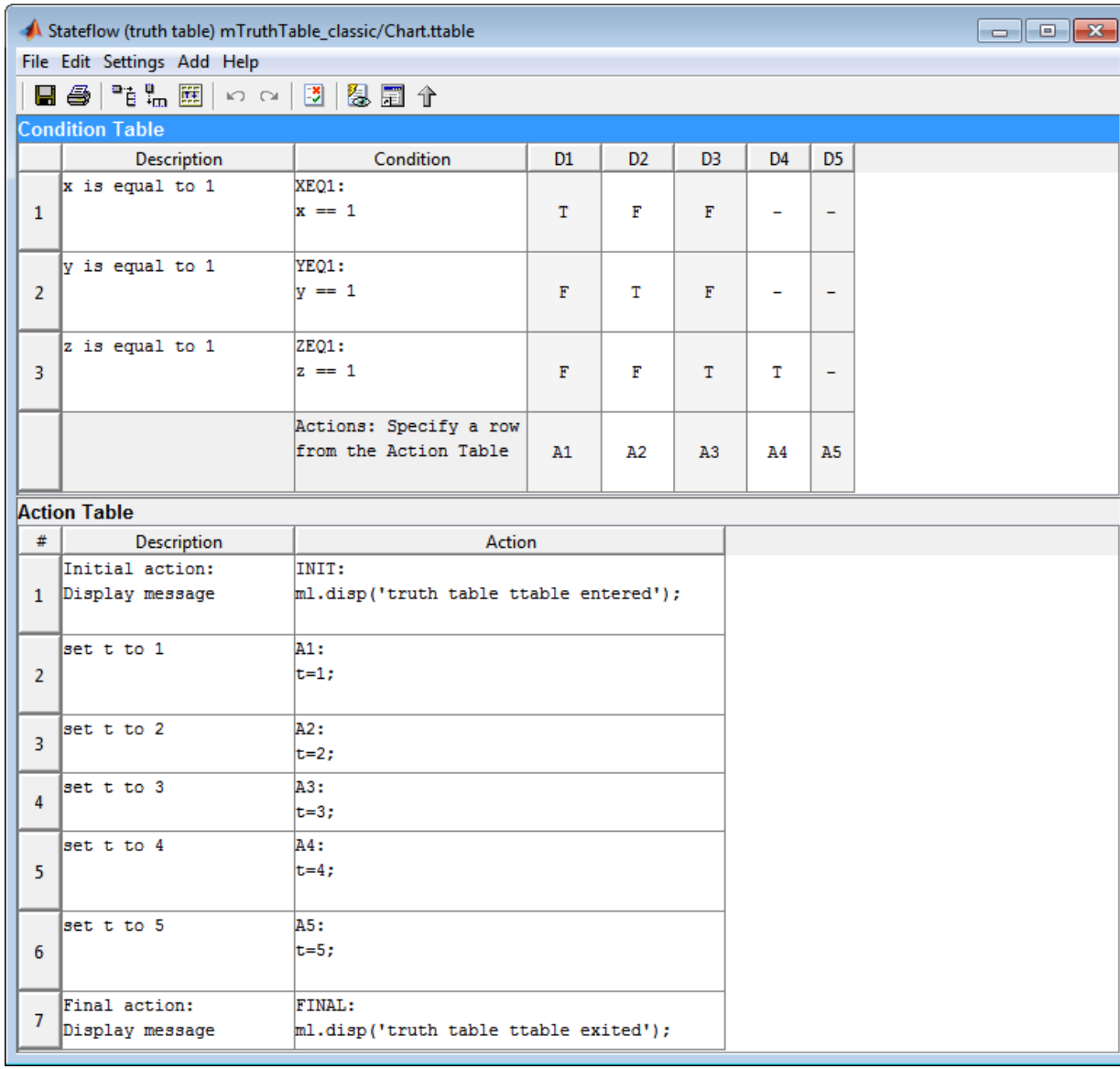
Analyze Coverage in Stateflow Truth Tables

If you have a Stateflow license, you can generate a model coverage report for a truth table.

Consider the following model.



The Stateflow chart contains the following truth table:



	Description	Condition	D1	D2	D3	D4	D5
1	x is equal to 1	XEQ1: x == 1	T	F	F	-	-
2	y is equal to 1	YEQ1: y == 1	F	T	F	-	-
3	z is equal to 1	ZEQ1: z == 1	F	F	T	T	-
		Actions: Specify a row from the Action Table	A1	A2	A3	A4	A5

#	Description	Action
1	Initial action: Display message	INIT: ml.disp('truth table ttable entered');
2	set t to 1	A1: t=1;
3	set t to 2	A2: t=2;
4	set t to 3	A3: t=3;
5	set t to 4	A4: t=4;
6	set t to 5	A5: t=5;
7	Final action: Display message	FINAL: ml.disp('truth table ttable exited');

When you simulate the model and collect coverage, the model coverage report includes the following data:

4. Truth Table "ttable"

Parent: mTruthTable_classic/Chart

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	10
Condition (C1)	NA	17% (3/18) condition outcomes
Decision (D1)	NA	13% (1/8) decision outcomes
MCDC (C1)	NA	0% (0/9) conditions reversed the outcome

Condition table analysis (missing values are in parentheses)

x is equal to 1	XEQ1: x == 1	T (F)	F (TF)	F (TF)	-	-
y is equal to 1	YEQ1: y == 1	F (T)	T (TF)	F (TF)	-	-
z is equal to 1	ZEQ1: z == 1	F (T)	F (TF)	T (TF)	T (TF)	-
	Actions	A1 (F)	A2 (TF)	A3 (TF)	A4 (TF)	A5

The **Coverage (this object)** column shows no coverage. The reason is that the container object for the truth table function—the Stateflow chart—does not decide whether to execute the `ttable` truth table.

The **Coverage (inc. descendants)** column shows coverage for the graphical function. The graphical function has the decision logic that makes the transitions for the truth table. The transitions in the graphical function contain the decisions and conditions of the truth table. Coverage for the descendants in the **Coverage (inc. descendants)** column

includes coverage for these conditions and decisions. Function calls to the truth table test the model coverage of these conditions and decisions.

Note See “How Stateflow Generates Content for Truth Tables” (Stateflow) for a description of the graphical function for a truth table.

Coverage for the decisions and their individual conditions in the `ttable` truth table function are as follows.

Coverage	Explanation
No model coverage for the default decision, **D5**	All logic that leads to taking a default decision is based on a false outcome for all preceding decisions. This means that the default decision requires no logic, so there is no model coverage.
13% (1/8) decision coverage	The three constants that are inputs to the truth table (`1`, `0`, `0`) cause only decision *D1* to be true. These inputs satisfy only one of the eight decisions (*D1* through *D4*, `T` or `F`). Because each condition can have an outcome value of `T` or `F`, three conditions can have six possible values. However, decision *D4* has only decision coverage, not condition coverage or MCDC coverage, because it represents a decision with a single predicate.
3 of the 18 (17%) condition coverage	Three decisions *D1*, *D2*, and *D3* have condition coverage, because the set of inputs (`1`, `0`, `0`) make only decision *D1* true.
No (0/9) MCDC coverage	MCDC coverage looks for decision reversals that occur because one condition outcome changes from `T` to `F` or `F` to `T`. The simulation tests only one set of inputs, so the model reverses no decisions.

Coverage	Explanation
Missing coverage	The red letters T and F indicate that model coverage is missing for those conditions. For decision *D1*, only the T decision is satisfied. For decisions *D2*, *D3*, and *D4*, none of the conditions are satisfied.

Colored Stateflow Chart Coverage Display

The Model Coverage tool displays model coverage results for individual blocks directly in Simulink diagrams. If you enable this feature, the Model Coverage tool:

- Highlights Stateflow objects that receive model coverage during simulation
- Provides a context-sensitive display of summary model coverage information for each object

> **Note** The coverage tool changes colors only for open charts at the time coverage information is reported. When you interact with the chart, such as selecting a transition or a state, colors revert to default values.

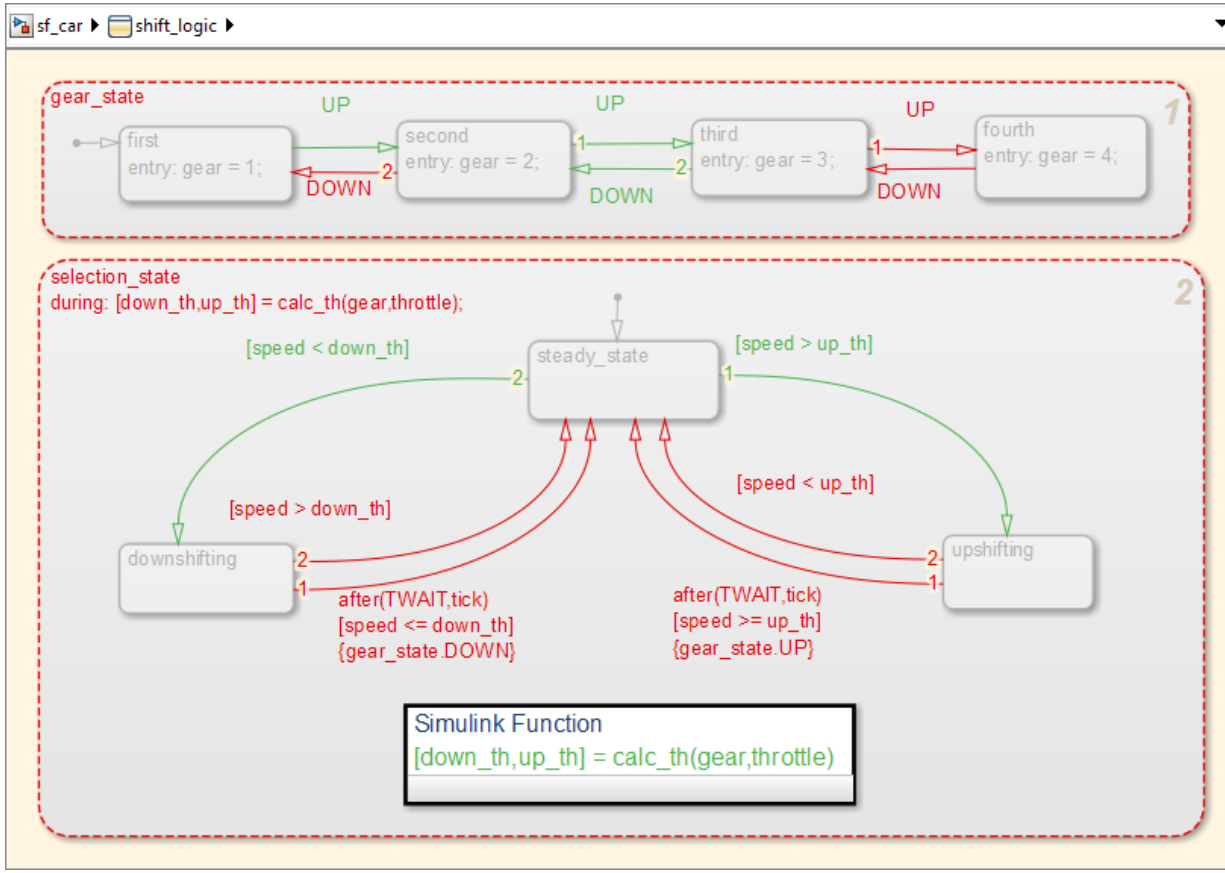
For details on enabling and selecting this feature in the Simulink window, see “Enable Coverage Highlighting” on page 5-13.

Display Model Coverage with Model Coloring

Once you enable display coverage with model coloring, anytime that the model generates a model coverage report, individual chart objects receiving coverage appear highlighted with light green or light red.

1. Open the `sf_car` model.
2. Select **Analysis** > **Coverage** > **Settings**.
3. In the **Coverage** pane of the Configuration Parameters dialog box, select **Enable coverage analysis**.
4. In the **Coverage** > **Results** pane, select **Display coverage results using model coloring**.
5. Click **OK**.
6. Simulate the model.

After simulation ends, chart objects with coverage appear highlighted.



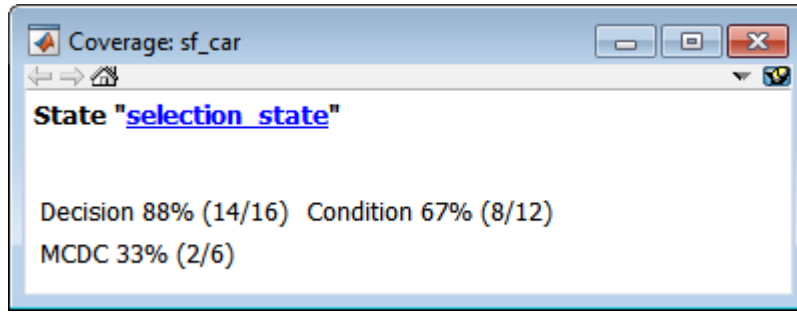
Object highlighting indicates coverage as follows:

- Light green for full coverage
- Light red for partial coverage
- No color for zero coverage

Note To revert the chart to show original colors, select and deselect any objects.

7 Click `selection_state` in the chart.

The following summary report appears.



When you click a highlighted Stateflow object, the summarized coverage for that object appears in the Coverage Display Window. Clicking the hyperlink opens the section of the coverage report for this object.

Tip You can set the Coverage Display Window to appear for a block in response to a hovering mouse cursor instead of a mouse click in one of two ways:

- Select the downward arrow on the right side of the Coverage Display Window and select **Focus**.
- Right-click a colored block and select **Coverage > Display details on mouse-over**.

Results Review

- “Types of Coverage Reports” on page 6-2
- “Top-Level Model Coverage Report” on page 6-12
- “Export Model Coverage Web View” on page 6-48

Types of Coverage Reports

If you choose to generate a coverage report automatically after analysis from the **Coverage > Results** pane of the Configuration Parameters dialog box or you generate a report from the Results Explorer, the Simulink Coverage software creates one or more model coverage reports after a simulation.

Report Type	Description	HTML Report File Name
"Top-Level Model Coverage Report" on page 6-12	Provides coverage information for all model elements, including the model itself.	*model_name*_cov.html
"Model Summary Report" on page 6-3	Provides links to coverage results for referenced models and external MATLAB files in the model hierarchy. Created when the top-level model includes Model blocks or calls one or more external files.	*model_name*_summary_cov.html
"Model Reference Coverage Report" on page 6-4	Created for each referenced model in the model hierarchy; has the same format as the model coverage report.	*reference_model_name*_cov.html
"External MATLAB File Coverage Report" on page 6-5	Provides detailed coverage information about any external MATLAB file that the model calls. There is one report for each external file called from the model.	*MATLAB_file_name*_cov.html
"Subsystem Coverage Report" on page 6-9	Model coverage report includes only coverage results for the subsystem, if you select one.	*model_name*_cov.html; *model_name* is the name of the top-level model
"Code Coverage Report" on page 6-11	Provides coverage information for C/C++ code in S-Function blocks, or for models in SIL mode.	*model_name_block_name*_instance_*n*_cov.html, or *model_name*_cov.html

Model Summary Report

If the top-level model contains Model blocks or calls external files, the software creates a model summary coverage report named *`model_name`*`_summary_cov.html`. The title of this report is *Coverage by Model*.

The summary report lists and provides links to coverage reports for Model block referenced models and external files called by MATLAB code in the model. For more information, see “External MATLAB File Coverage Report” on page 6-5.

The following graphic shows an example of a model summary report. It contains links to the model coverage report (`mExternalMfile`), a report for the Model block (`mExternalMfileRef`), and three external files called from the model (`externalmfile`,I `externalmfile1`, and`externalmfile2`).

Coverage Report by Model

Top Model: mExternalMfile

	Complexity	Decision	Condition	MCDC
TOTAL COVERAGE		**90%**	**75%**	**25%**
1. . . . mExternalMfile	5	50%	--	--
2. . . . externalmfile1	5	88%	75%	0%
3. . . . mExternalMfileRef	3	100%	--	--
4. . . . externalmfile	5	100%	75%	50%
5. . . . externalmfile2	2	100%	--	--

The following models have signal range coverage:

mExternalMfile

mExternalMfileRef

Model Reference Coverage Report

If your top-level model references a model in a Model block, the software creates a separate report, named *`reference_model_name`*`_cov.html`, that includes coverage for the referenced model. This report has the same format as the “Top-Level Model Coverage Report” on page 6-12. Coverage results are recorded as if the referenced model was a standalone model; the report gives no indication that the model is referenced in a Model block.

External MATLAB File Coverage Report

If your top-level model calls any external MATLAB files, select **MATLAB files** on the **Coverage** pane of the Configuration Parameters dialog box. The software creates a report, named *MATLAB_file_name*_cov.html, for each distinct file called from the model. When there are several calls to a given file from the model, the software creates only one report for that file, but it accumulates coverage from all the calls to the file. The external MATLAB file coverage report does not include information about what parts of the model call the external file.

The first section of the external MATLAB file coverage report contains summary information about the external file, similar to the model coverage report.

Coverage Report for externalmfile1

Table of Contents

1. Analysis Information
2. Tests
3. Summary
4. Details

Analysis Information

MATLAB Function File Information

Last saved	13-Nov-2008 12:39:55

Simulation Optimization Options

Inline parameters	off
Block reduction	forced off
Conditional branch optimization	on

Coverage Options

Analyzed model	externalmfile1
Logic block short circuiting	off

Tests

Test 1

Started execution 20-Dec-2013 15:45:08
Ended execution 20-Dec-2013 15:45:09

Summary

Model Hierarchy/Complexity		Test 1				
		D1		C1		MCDC
1. externalmfile1	5	88%		75%		0%

The *Details* section reports coverage for the external file and the function in that file.

Details

1. MATLAB Function file "externalmfile1"

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	**1**	**5**
Condition (C1)	NA	75% (3/4) condition outcomes
Decision (D1)	NA	88% (7/8) decision outcomes
MCDC (C1)	NA	0% (0/2) conditions reversed the outcome

MATLAB Function "externalmfile1"

Parent: externalmfile1
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	4
Condition (C1)	75% (3/4) condition outcomes
Decision (D1)	88% (7/8) decision outcomes
MCDC (C1)	0% (0/2) conditions reversed the outcome

The *Details* section also lists the content of the file, highlighting the code lines that have decision points or function definitions.

```
 1  %#eml
 2  function y = externalmfile1(u)
 3
 4  %   Copyright 2008 The MathWorks, Inc.
 5
 6  if u>1 && u<5
 7      a = 2;
 8  else
 9      a = 3;
10  end
11
12  for i=1:5
13      a = a+2;
14  end
15
16  y = a+localtest(a);
17
18  [x,y] = pol2cart(u,u);
19  [y2,y3] = cart2pol(x,y);
20
21  function y = localtest(u)
22
23  y = 0;
24  flg = true;
25  while flg
26      u = u/2;
27      y = y+1;
28      flg = u>2;
29  end
30
```

Coverage results for each of the highlighted code lines follow in the report. The following graphic shows a portion of these coverage results from the preceding code example.

#2: function y = externalmfile1(u)

Decisions analyzed

function y = externalmfile1(u)	100%
executed	102/102

#6: if u>1 && u<5

Decisions analyzed

if u>1 && u<5	50%
false	102/102
true	0/102

Subsystem Coverage Report

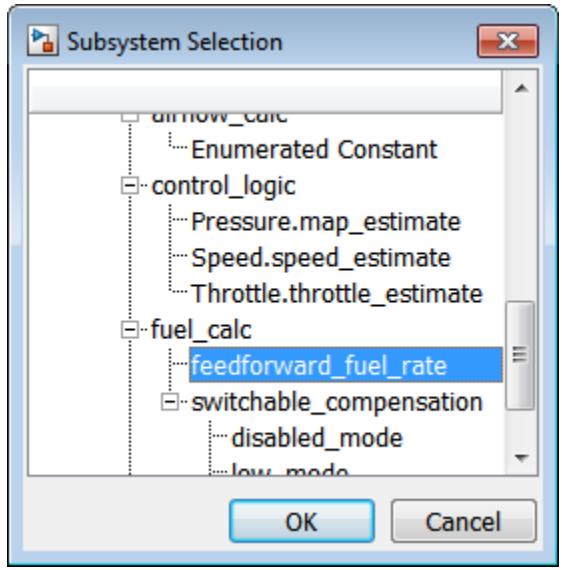
In the **Coverage** pane of the Configuration Parameters dialog box, when you select **Enable coverage analysis**, you can click **Select Subsystem** to request coverage for only the selected subsystem in the model. The software creates a model coverage report for the top-level model, but includes coverage results only for the subsystem.

However, if the top-level model calls any external files and you select **MATLAB files** in the **Coverage** pane of the Configuration Parameters dialog box, the results include coverage for all external files called from:

- The subsystem for which you are recording coverage
- The top-level model that includes the subsystem

If the subsystem parameter **Read/Write Permissions** is set to `NoReadOrWrite`, the software does not record coverage for that subsystem.

For example, in the `fuelsys` model, you click **Select Subsystem**, and select coverage for the `feedforward_fuel_rate` subsystem.



The report is similar to the model coverage report, except that it includes only results for the `feedforward_fuel_rate` subsystem and its contents.

Summary

Model Hierarchy/Complexity:	Test 1 D1
1. feedforward_fuel_rate	3 33%

Details:

1. SubSystem block "feedforward_fuel_rate"

Parent: sldemo_fuelsys/fuel_rate_control/fuel_calc

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	**1**	**3**
Decision (D1)	NA	33% (1/3) decision outcomes

Code Coverage Report

For each S-Function block, the model coverage report links to a detailed code coverage report for the C/C++ code in the block. For more information on how to navigate the report, see “View Coverage Results for C/C++ Code in S-Function Blocks” on page 5-49.

If you have Embedded Coder installed, you can also generate code coverage reports from models in SIL or PIL mode. For more information on how to generate code coverage reports for models in SIL or PILmode, see “Code Coverage for Models in Software-in-the-Loop (SIL) Mode and Processor-in-the-Loop (PIL) Mode” on page 4-7.

Top-Level Model Coverage Report

The Simulink Coverage software always creates a model coverage report for the top-level model named *model_name*_cov.html. The model coverage report contains several sections:

In this section...
“Coverage Summary” on page 6-12
“Details” on page 6-14
“Cyclomatic Complexity” on page 6-23
“Decisions Analyzed” on page 6-25
“Conditions Analyzed” on page 6-27
“MCDC Analysis” on page 6-27
“Cumulative Coverage” on page 6-29
“N-Dimensional Lookup Table” on page 6-31
“Block Reduction” on page 6-37
“Relational Boundary” on page 6-38
“Saturate on Integer Overflow Analysis” on page 6-42
“Signal Range Analysis” on page 6-43
“Signal Size Coverage for Variable-Dimension Signals” on page 6-45
“Simulink Design Verifier Coverage” on page 6-46

Coverage Summary

The coverage summary section contains basic information about the model being analyzed:

- **Model Information**
- **Simulation Optimization Options**
- **Coverage Options**

Coverage Report for sldemo_fuelsys

Table of Contents

1. Analysis Information
2. Tests
3. Summary
4. Details
5. Signal Ranges

Analysis Information

Model Information

Model version	1.736
Author	The MathWorks, Inc.
Last saved	Fri Sep 23 19:02:53 2016

Simulation Optimization Options

Default parameter behavior	inlined
Block reduction	forced off
Conditional branch optimization	on

Coverage Options

Analyzed model	sldemo_fuelsys
Logic block short circuiting	off

The coverage summary has two subsections:

- *Tests* — The simulation start and stop time of each test case and any setup commands that preceded the simulation. The heading for each test case includes any test case label specified using the `cvtest` command.
- *Summary* — Summaries of the subsystem results. To see detailed results for a specific subsystem, in the Summary subsection, click the subsystem name.

Tests

Test#	Started execution	Ended execution	Description
Test 1	07-Oct-2016 09:06:06	07-Oct-2016 09:08:25	This is a model of a fuel control system where Stateflow(R) is used to handle the fault management of the system. The system contains four separate sensors: a throttle sensor, a speed sensor, an oxygen sensor, and a pressure sensor. Each of these sensors is represented by a parallel state in Stateflow. Each parallel state contains two substates, a normal state and a failed state (the exception being the oxygen sensor, which also contains a warmup state). If any of the sensor readings is outside an acceptable range, then a fault is registered in Stateflow, and the substate of the corresponding subsystem transitions to the failed state. If a subsystem recovers, it can transition back to the normal state. The number of failures in the system at any given time is represented in the Fail parallel state. The last parallel state in the Stateflow chart is called Fueling_Mode. This state regulates the oxygen to fuel mixture ratio. If a failure is detected, then the oxygen to fuel ratio is increased. If multiple failures are detected, then the fuel system is disabled until there are no longer multiple failures in the system.

Summary

Model Hierarchy/Complexity	Test 1											
	Decision		Condition		MCDC		Execution		Relational Boundary		Saturation on integer overflow	
1. sldemo_fuelsys	80	34%		34%		7%		90%		10%		50%
2. . . . Engine Gas Dynamics	13	71%		NA		NA		100%		50%		50%
3. . . . . Mixing & Combustion	3	67%		NA		NA		100%		NA		50%
4. . . . . EGO Sensor	2	100%		NA		NA		NA		NA		NA
5. . . . . System Lag		NA		NA		NA		100%		NA		NA
6. . . . . Throttle & Manifold	10	73%		NA		NA		100%		50%		50%
7. . . . . Intake Manifold	2	100%		NA		NA		100%		NA		50%
8. . . . . MATLAB Function	2	100%		NA		NA		NA		NA		NA
9. . . . . Throttle	6	83%		NA		NA		100%		100%		50%

Details

The Details section reports the detailed model coverage results. Each section of the detailed report summarizes the results for the metrics that test each object in the model:

- “Filtered Objects” on page 6-15
- “Model Details” on page 6-15
- “Subsystem Details” on page 6-16
- “Block Details” on page 6-17

- “Chart Details” on page 6-19
- “Coverage Details for MATLAB Functions and Simulink Design Verifier Functions” on page 6-20

You can also access a model element Details subsection as follows:

1. Right-click a Simulink element.
2. In the context menu, select **Coverage > Report**.

Filtered Objects

The Filtered Objects section lists all the objects in the model that were filtered from coverage recording, and the rationale you specified for filtering those objects. If the filter rule specifies that all blocks of a certain type be filtered, all those blocks are listed here.

In the following graphic, several blocks, subsystems, and transitions were filtered. Two library-linked blocks, protected division and protected division1, were filtered because their block library was filtered.

Blocks Eliminated from Coverage Analysis

Model Object	Rationale
slvnvdemo_covfilt/Saturation	It might not be executed because of Conditional input branch optimization
slvnvdemo_covfilt/protected division/Compare To Zero/Compare	It might not be executed because of Conditional input branch optimization
slvnvdemo_covfilt/protected division/Switch	It might not be executed because of Conditional input branch optimization
slvnvdemo_covfilt/protected division/Switch1	It might not be executed because of Conditional input branch optimization
slvnvdemo_covfilt/protected division1/Switch	It might not be executed because of Conditional input branch optimization

Model Details

The Details section contains a results summary for the model as a whole, followed by a list of elements. Click the model element name to see its coverage results.

The following graphic shows the Details section for the `sldemo_fuelsys` example model.

Details

1. Model "sldemo_fuelsys"

Child Systems: Engine Gas Dynamics, Throttle Command, To Controller, To Plant, fuel_rate_control

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	80
Condition	NA	34% (11/32) condition outcomes
Decision	NA	34% (41/122) decision outcomes
MCDC	NA	7% (1/14) conditions reversed the outcome
Lookup Table	NA	1% (13/1511)interpolation/extrapolation intervals
Execution	NA	90% (64/71) objective outcomes
Relational Boundary	NA	10% (5/50) objective outcomes
Saturation on integer overflow	NA	50% (10/20) objective outcomes

Subsystem Details

Each subsystem Details section contains a summary of the test coverage results for the subsystem and a list of the subsystems it contains. The overview is followed by sections for blocks, charts, and MATLAB functions, one for each object that contains a decision point in the subsystem.

The following graphic shows the coverage results for the Engine Gas Dynamics subsystem in the `sldemo_fuelsys` example model.

2. SubSystem block "Engine Gas Dynamics"

Justify or Exclude

Parent: /sldemo_fuelsys

Child Systems: Mixing & Combustion, Throttle & Manifold

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	13
Decision	NA	71% (10/14) decision outcomes
Execution	NA	100% (17/17) objective outcomes
Relational Boundary	NA	50% (3/6) objective outcomes
Saturation on integer overflow	NA	50% (10/20) objective outcomes

Block Details

The following graphic shows decision coverage results for the MinMax block in the Mixing & Combustion subsystem of the Engine Gas Dynamics subsystem in the `sldemo_fuelsys` example model.

MinMax block "MinMax"

Justify or Exclude

Parent: sldemo_fuelsys/Engine Gas Dynamics/Mixing & Combustion

Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	1
Decision	50% (1/2) decision outcomes
Execution	100% (1/1) objective outcomes

Decisions analyzed

Logic to determine output	50%
input 1 is the maximum	204508/204508
input 2 is the maximum	0/204508

The *Uncovered Links* element first appears in the Block Details section of the first block in the model hierarchy that does not achieve 100% coverage. The first *Uncovered Links* element has an arrow that links to the Block Details section in the report of the *next* block that does not achieve 100% coverage.

Subsequent blocks that do not achieve 100% coverage have links to the Block Details sections in the report of the previous and next blocks that do not achieve 100% coverage.

Saturate block "Limit to Positive"

Parent: sldemo_fuelsys/Engine Gas Dynamics/Throttle & Manifold

Uncovered Links:

Chart Details

The following graphic shows the coverage results for the Stateflow chart control_logic in the `sldemo_fuelsys` example model.

17. SubSystem block "control_logic"

Justify or Exclude

Parent: sldemo_fuelsys/fuel_rate_control
Child Systems: fuel_rate_control/control_logic

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	1	56
Condition	NA	21% (5/24) condition outcomes
Decision	NA	25% (23/92) decision outcomes
MCDC	NA	0% (0/12) conditions reversed the outcome
Lookup Table	NA	0% (0/1082)interpolation/extrapolation intervals
Execution	NA	0% (0/4) objective outcomes
Relational Boundary	NA	0% (0/34) objective outcomes

18. Chart "fuel_rate_control/control_logic"

Justify or Exclude

Parent: sldemo_fuelsys/fuel_rate_control/control_logic
Child Systems: Fail, Fueling_Mode, O2, Pressure, Speed, Throttle

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	55
Condition	NA	21% (5/24) condition outcomes
Decision	NA	25% (23/92) decision outcomes
MCDC	NA	0% (0/12) conditions reversed the outcome
Lookup Table	NA	0% (0/1082)interpolation/extrapolation intervals
Execution	NA	0% (0/4) objective outcomes
Relational Boundary	NA	0% (0/34) objective outcomes

For more information about model coverage reports for Stateflow charts and their objects, see “Model Coverage for Stateflow Charts” on page 5-54.

Coverage Details for MATLAB Functions and Simulink Design Verifier Functions

By default, Simulink Coverage records coverage for all MATLAB functions in a model. MATLAB functions are in MATLAB Function blocks, Stateflow charts, or external MATLAB files.

Note For a detailed example of coverage reports for external MATLAB files, see “External MATLAB File Coverage Report” on page 6-5.

To record Simulink Design Verifier coverage for `sldv.*` functions called by MATLAB functions, and any Simulink Design Verifier blocks, select **Objectives and Constraints** on the **Coverage** pane of the Configuration Parameters dialog box.

The following example shows coverage details for a MATLAB function, `hFcnsInExternalEML`, that calls four Simulink Design Verifier functions. In this example, the code for `hFcnsInExternalEML` resides in an external file.

This example also shows Simulink Design Verifier coverage details for the following functions:

- `sldv.assume`
- `sldv.condition`
- `sldv.prove`
- `sldv.test`

In the coverage results, code that achieves 100% coverage is green. Code that achieves less than 100% coverage is red.

Embedded MATLAB function "hfcnsinexternaleml"

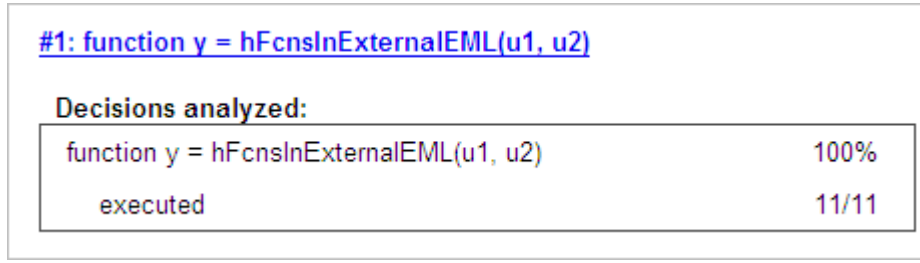
Parent: hfcnsinexternaleml
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	4
Decision (D1)	40% (2/5) decision outcomes
Test Objective	50% (1/2) objective outcomes
Proof Objective	0% (0/1) objective outcomes
Test Condition	100% (1/1) objective outcomes
Proof Assumption	0% (0/1) objective outcomes

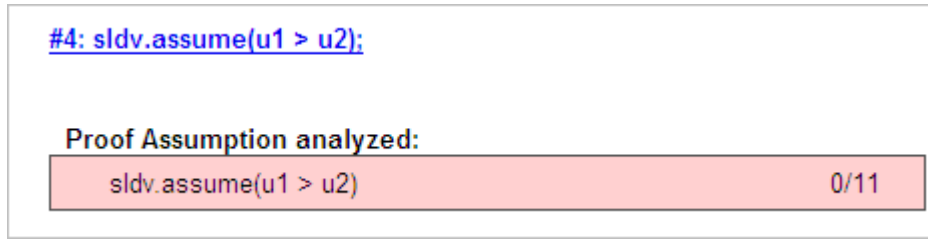
```
 1  function y = hFcnsInExternalEML(u1, u2)
 2  % use all four functions.
 3  %#eml
 4  sldv.assume(u1 > u2);
 5  sldv.condition(u1 == 0);
 6  switch u1
 7      case 0
 8          y = u2;
 9      case 1
10          y = 3;
11      case 2
12          y = 0;
13      otherwise
14          y = 0;
15          sldv.prove(u2 < u1);
16  end
17  sldv.test(y > u1); sldv.test(y == 4);
18
```

Coverage for the `hFcnsInExternalEML` function and the `sldv.*` calls is:

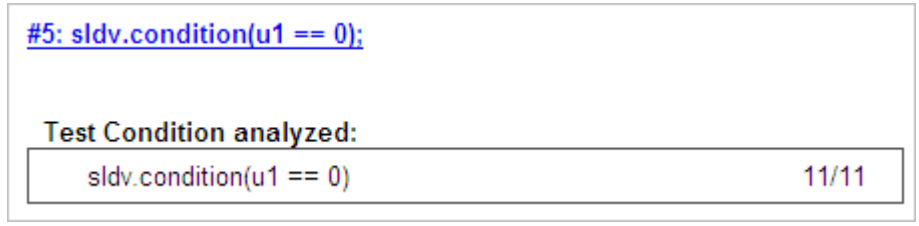
- Line 1, the function declaration for `hFcnsInExternalEML`is green because the simulation executes that function at least once. `fcn` calls `hFcnsInExternalEML` 11 times during simulation.



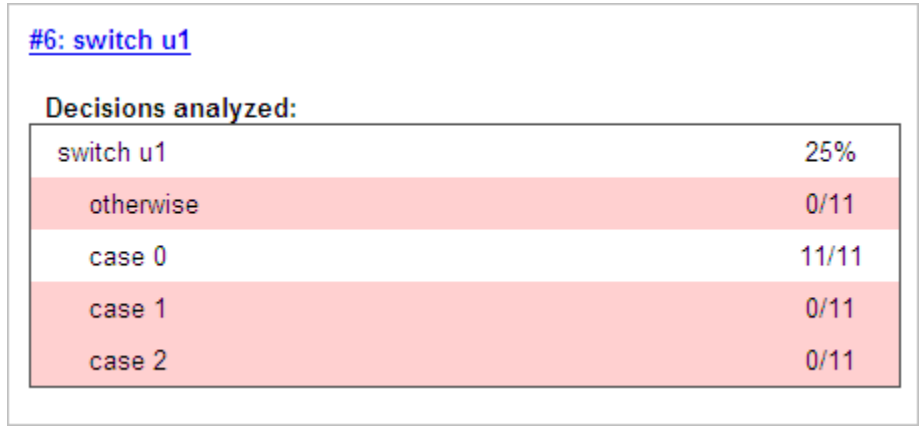
Line 4, `sldv.assume(u1 > u2)`, achieves 0% coverage because `u1 > u2` never evaluates to true.



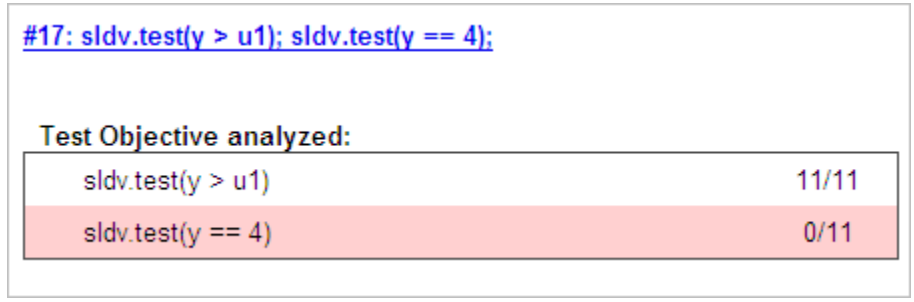
- Line 5, `sldv.condition(u1 == 0)`, achieves 100% coverage because `u1 == 0` evaluates to true for at least one time step.



- Line 6, `switch u1`, achieves 25% coverage because only one of the four outcomes in the `switch` statement (`case 0`) occurs during simulation.



- Line 17, `sldv.test(y > u1); sldv.test (y == 4)` achieves 50% coverage. The first `sldv.test` call achieves 100% coverage, but the second `sldv.test` call achieves 0% coverage.



For more information about coverage for MATLAB functions, see “Model Coverage for MATLAB Functions” on page 5-29.

For more information about coverage for Simulink Design Verifier functions, see “Objectives and Constraints Coverage” on page 1-7.

Cyclomatic Complexity

You can specify that the model coverage report include cyclomatic complexity numbers in two locations in the report:

- The Summary section contains the cyclomatic complexity numbers for each object in the model hierarchy. For a subsystem or Stateflow chart, that number includes the cyclomatic complexity numbers for all their descendants.

Summary

Model Hierarchy/Complexity:

1. fuelsys	**78**
2. . . . engine gas dynamics	5
3. . . . . Mixing & Combustion	1
4. . . . . Throttle & Manifold	4
5. . . . . Throttle	2
6. . . . fuel rate controller	72
7. . . . . Airflow calculation	1
8. . . . . Fuel Calculation	11
9. . . . . Switchable Compensation	7
10. . . . . LOW Mode	**2**
11. . . . . RICH Mode	**2**
12. . . . . Sensor correction and Fault Redundancy	9
13. . . . . MAP Estimate	**2**
14. . . . . Speed Estimate	**2**
15. . . . . Throttle Estimate	**2**
16. . . . . control logic	**51**
17. . . . . SF: control logic	50
18. . . . . SF: Fail	12
19. . . . . SF: Multi	6
20. . . . . SF: Fueling_Mode	19
21. . . . . SF: Fuel_Disabled	4
22. . . . . SF: Running	10
23. . . . . SF: Low_Emissions	4
24. . . . . SF: O2	5
25. . . . . SF: Pressure	5
26. . . . . SF: Speed	4
27. . . . . SF: Throttle	5

- The Details sections for each object list the cyclomatic complexity numbers for all individual objects.

6. SubSystem block "Throttle & Manifold"

Justify or Exclude

Parent: sldemo_fuelsys/Engine Gas Dynamics
Child Systems: Intake Manifold, Throttle

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	10
Decision	NA	73% (8/11) decision outcomes
Execution	NA	100% (13/13) objective outcomes
Relational Boundary	NA	50% (3/6) objective outcomes
Saturation on integer overflow	NA	50% (8/16) objective outcomes

Decisions Analyzed

The Decisions analyzed table lists possible outcomes for a decision and the number of times that an outcome occurred in each test simulation. Outcomes that did not occur are in red highlighted table rows.

The following graphic shows the Decisions analyzed table for the Saturate block in the Throttle & Manifold subsystem of the Engine Gas Dynamics subsystem in the `sldemo_fuelsys` example model.

Saturate block "Limit to Positive"

Justify or Exclude

Parent: sldemo_fuelsys/Engine Gas Dynamics/Throttle & Manifold

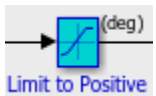
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	2
Decision	50% (2/4) decision outcomes
Execution	100% (1/1) objective outcomes
Relational Boundary	25% (1/4) objective outcomes

Decisions analyzed

input > lower limit	50%
false	0/204508
true	204508/204508
input >= upper limit	50%
false	204508/204508
true	0/204508

To display and highlight the block in question, click the block name at the top of the section containing the block's Decisions analyzed table.



Conditions Analyzed

The Conditions analyzed table lists the number of occurrences of true and false conditions on each input port of the corresponding block.

Conditions analyzed

Description	True	False
input port 1	199521	480
input port 2	200001	0

MCDC Analysis

The MCDC analysis table lists the MCDC input condition cases represented by the corresponding block and the extent to which the reported test cases cover the condition cases.

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition:	True Out	False Out
expression for output		
input port 1	TT	FT
input port 2	TT	(TF)

Each row of the MCDC analysis table represents a condition case for a particular input to the block. A condition case for input n of a block is a combination of input values. Input n is called the *deciding input* of the condition case. Changing the value of input n alone changes the value of the block's output.

The MCDC analysis table shows a condition case expression to represent a condition case. A condition case expression is a character string where:

- The position of a character in the string corresponds to the input port number.
- The character at the position represents the value of the input. (`T` means `true`; `F` means `false`).
- A boldface character corresponds to the value of the deciding input.

For example, `F`**`T`**`F` represents a condition case for a three-input block where the second input is the deciding input.

The *Decision/Condition* column specifies the deciding input for an input condition case. The *True Out* column specifies the deciding input value that causes the block to output a `true` value for a condition case. The *True Out* entry uses a condition case expression, for example, **`F`**`F`, to express the values of all the inputs to the block, with the value of the deciding variable in bold.

Parentheses around the expression indicate that the specified combination of inputs did not occur during the first (or only) test case included in this report. In other words, the test case did not cover the corresponding condition case. The *False Out* column specifies the deciding input value that causes the block to output a false value and whether the value actually occurred during the first (or only) test case included in the report.

Some model elements achieve less MCDC coverage depending on the MCDC definition used during analysis. For more information on how the MCDC definition used during analysis affects the coverage results, see “Modified Condition and Decision Coverage (MCDC) Definitions in Simulink Coverage” on page 5-4.

If you select **Treat Simulink Logic blocks as short-circuited** in the **Coverage** pane of the Configuration Parameters dialog box, MCDC coverage analysis does not verify whether short-circuited inputs actually occur. The MCDC analysis table uses an `x` in a condition expression (for example, `TFxxx`) to indicate short-circuited inputs that were not analyzed by the tool.

If you disable this feature and Logic blocks are not short-circuited while collecting model coverage, you might not be able to achieve 100% coverage for that block.

Select the **Treat Simulink Logic blocks as short-circuited** option for where you want the MCDC coverage analysis to approximate the degree of coverage that your test cases achieve for the generated code (most high-level languages short-circuit logic expressions).

Cumulative Coverage

After you record successive coverage results, you can “Access, Manage, and Accumulate Coverage Results” on page 3-10 from within the Coverage Results Explorer. By default, the results of each simulation are saved and recorded cumulatively in the report.

If you select **Show cumulative progress report** in the Coverage Results Settings, the results located in the right-most area in all tables of the cumulative coverage report reflect the running total value. The report is organized so that you can easily compare the additional coverage from the most recent run with the coverage from all prior runs in the session.

A cumulative coverage report contains information about:

- Current Run — The coverage results of the simulation just completed.
- Delta — Percentage of coverage added to the cumulative coverage achieved with the simulation just completed. If the previous simulation's cumulative coverage and the current coverage are nonzero, the delta may be 0 if the new coverage does not add to the cumulative coverage.
- Cumulative — The total coverage collected for the model up to, and including, the simulation just completed.

After running three test cases for the `slvnvdemo_autopilot_test_harness` model, the Summary report shows how much additional coverage the third test case achieved and the cumulative coverage achieved for the first two test cases.

Summary

Model Hierarchy/Complexity:		Current Run			Delta			Cumulative		
		Decision	Condition	MCDC	Decision	Condition	MCDC	Decision	Condition	MCDC
1. slvnvdemo_autopilot_test_harness	31	38%	41%	17%	8%	6%	0%	51%	41%	17%
2. . . . Logic	25	34%	38%	17%	9%	8%	0%	47%	38%	17%
3. . . . . SF: Logic	24	34%	38%	17%	9%	8%	0%	47%	38%	17%
4. . . . . SF: Altitude	11	64%	67%	33%	21%	17%	0%	93%	67%	33%
5. . . . . SF: Active	4	38%	NA	NA	13%	NA	NA	88%	NA	NA
6. . . . . SF: GS	13	11%	8%	0%	0%	0%	0%	11%	8%	0%
7. . . . . SF: Active	6	0%	NA	NA	0%	NA	NA	0%	NA	NA
8. . . . . SF: Coupled	3	0%	NA	NA	0%	NA	NA	0%	NA	NA
9. . . . Verify Outputs	5	60%	50%	NA	0%	0%	NA	80%	50%	NA
10. . . . . Subsystem1	1	0%	NA	NA	0%	NA	NA	100%	NA	NA
11. . . . . Capture time	1	0%	NA	NA	0%	NA	NA	100%	NA	NA
12. . . . . Subsystem2	1	100%	NA	NA	0%	NA	NA	100%	NA	NA
13. . . . . Capture time	1	100%	NA	NA	0%	NA	NA	100%	NA	NA
14. . . . . Subsystem3	1	0%	NA	NA	0%	NA	NA	0%	NA	NA
15. . . . . Capture time	1	0%	NA	NA	0%	NA	NA	0%	NA	NA
16. . . . . Verification	2	100%	50%	NA	0%	0%	NA	100%	50%	NA

The *Decisions analyzed* table for cumulative coverage contains three columns of data about decision outcomes that represent the current run, the delta since the last run, and the cumulative data, respectively.

Decisions analyzed:

Transition trigger expression	100%	50%	100%
false	1097/1098	1097/1097	1097/1100
true	1/1098	0/1097	3/1100

The Conditions analyzed table uses column headers *#n T* and *#n F* to indicate results for individual test cases. The table uses *Tot T* and *Tot F* for the cumulative results. You can identify the true and false conditions on each input port of the corresponding block for each test case.

Conditions analyzed:

Description:	#1 T	#1 F	#2 T	#2 F	Tot T	Tot F
Condition 1, "alt_ctrl"	1	1097	0	1097	3	1097
Condition 2, "wow"	0	1	0	0	0	3
Condition 3, "in(GS.Active.Coupled)"	0	1	0	0	0	3

The MCDC analysis *#n True Out* and *#n False Out* columns show the condition cases for each test case. The *Total Out T* and *Total Out F* column show the cumulative results.

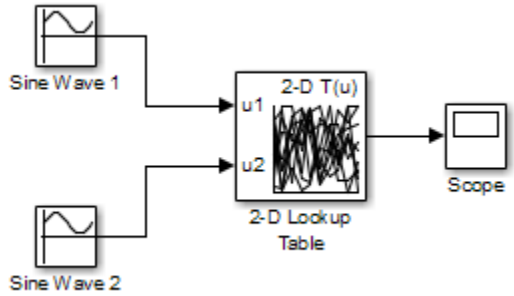
MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	#1 True Out	#1 False Out	#2 True Out	#2 False Out	Total Out T	Total Out F
Transition trigger expression						
Condition 1, "alt_ctrl"	TFF	Fxx	(TFF)	Fxx	TFF	Fxx
Condition 2, "wow"	TFF	(TTx)	(TFF)	(TTx)	TFF	(TTx)
Condition 3, "in(GS.Active.Coupled)"	TFF	(TFT)	(TFF)	(TFT)	TFF	(TFT)

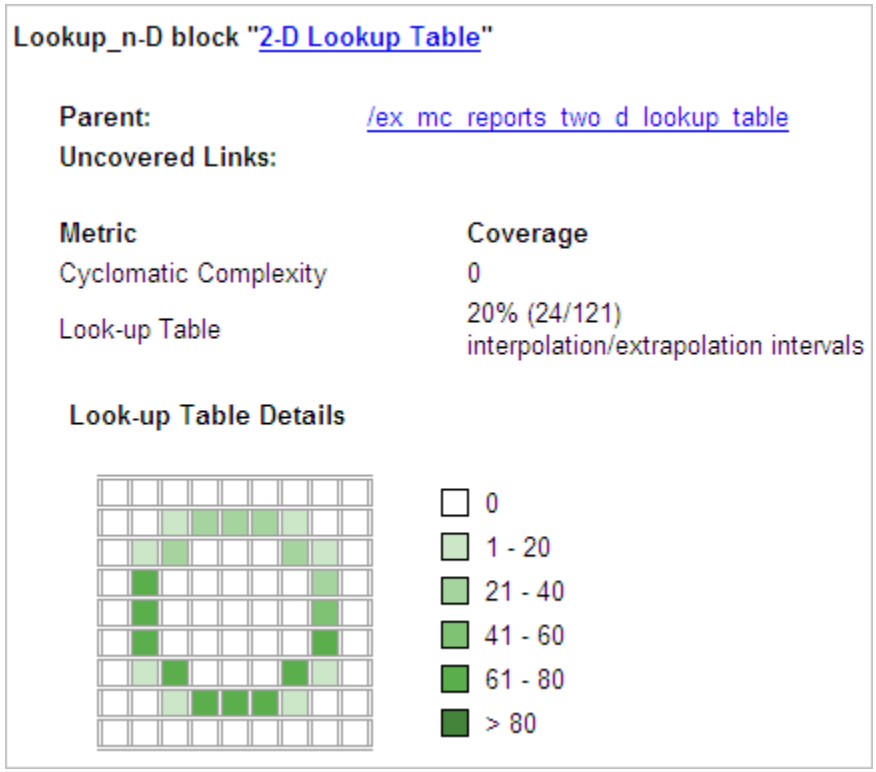
Note You can calculate cumulative coverage for reusable subsystems and Stateflow constructs at the command line. For more information, see “” on page 8-7.

N-Dimensional Lookup Table

The following interactive chart summarizes the extent to which elements of a lookup table are accessed. In this example, two Sine Wave blocks generate *x* and *y* indices that access a 2-D Lookup Table block of 10-by-10 elements filled with random values.



In this model, the lookup table indices are 1, 2,..., 10 in each direction. The Sine Wave 2 block is out of phase with the Sine Wave 1 block by **pi**/2 radians. This generates *x* and *y* numbers for the edge of a circle, which you see when you examine the resulting Lookup Table coverage.

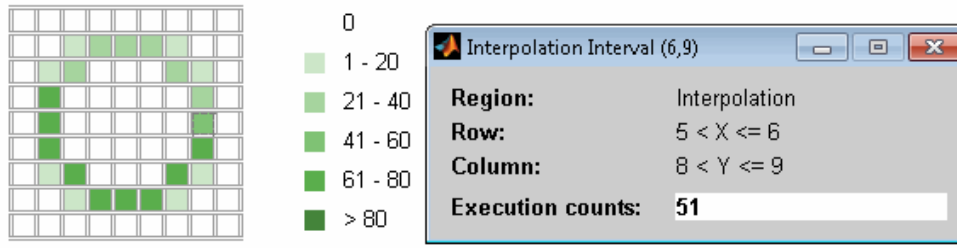


The report contains a two-dimensional table representing the elements of the lookup table. The element indices are represented by the cell border grid lines, which number 10 in each dimension. Areas where the lookup table interpolates between table values are represented by the cell areas. Areas of extrapolation left of element 1 and right of element 10 are represented by cells at the edge of the table, which have no outside border.

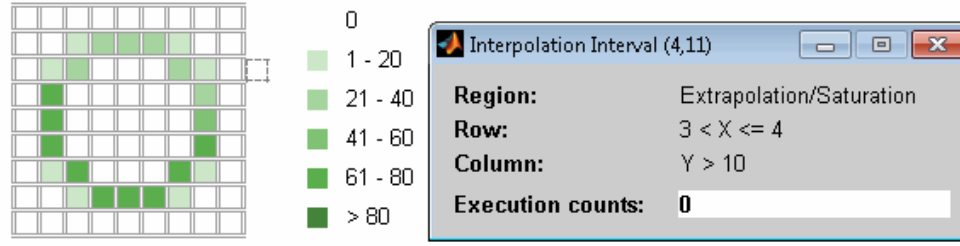
The number of values interpolated (or extrapolated) for each cell (*execution counts*) during testing is represented by a shade of green assigned to the cell. Each of six levels of green shading and the range of execution counts represented are displayed on one side of the table.

If you click an individual table cell, you see a dialog box that displays the index location of the cell and the exact number of execution counts generated for it during testing. The

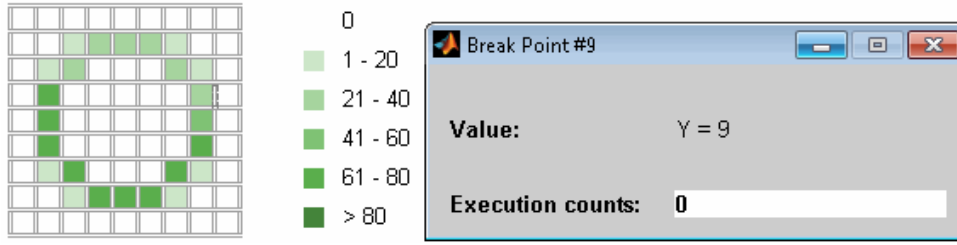
following example shows the contents of a color-shaded cell on the right edge of the circle.



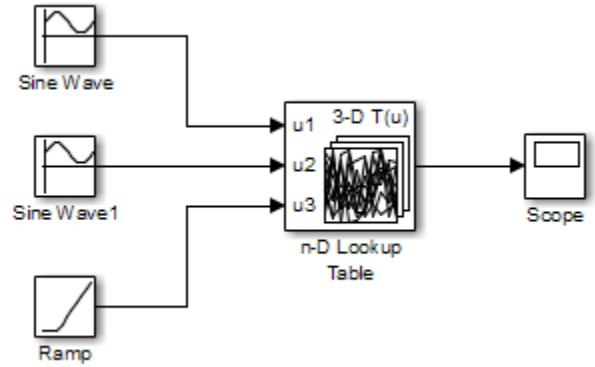
The selected cell is outlined in red. You can also click the extrapolation cells on the edge of the table.



A bold grid line indicates that at least one block input equal to its exact index value occurred during the simulation. Click the border to display the exact number of hits for that index value.



The following example model uses an n-D Lookup Table block of 10-by-10-by-5 elements filled with random values.



Both the *x* and *y* table axes have the indices 1, 2,..., 10. The *z* axis has the indices 10, 20,..., 50. Lookup table values are accessed with *x* and *y* indices that the two Sine Wave blocks generated, in the preceding example, and a *z* index that a Ramp block generates.

After simulation, you see the following lookup table report.

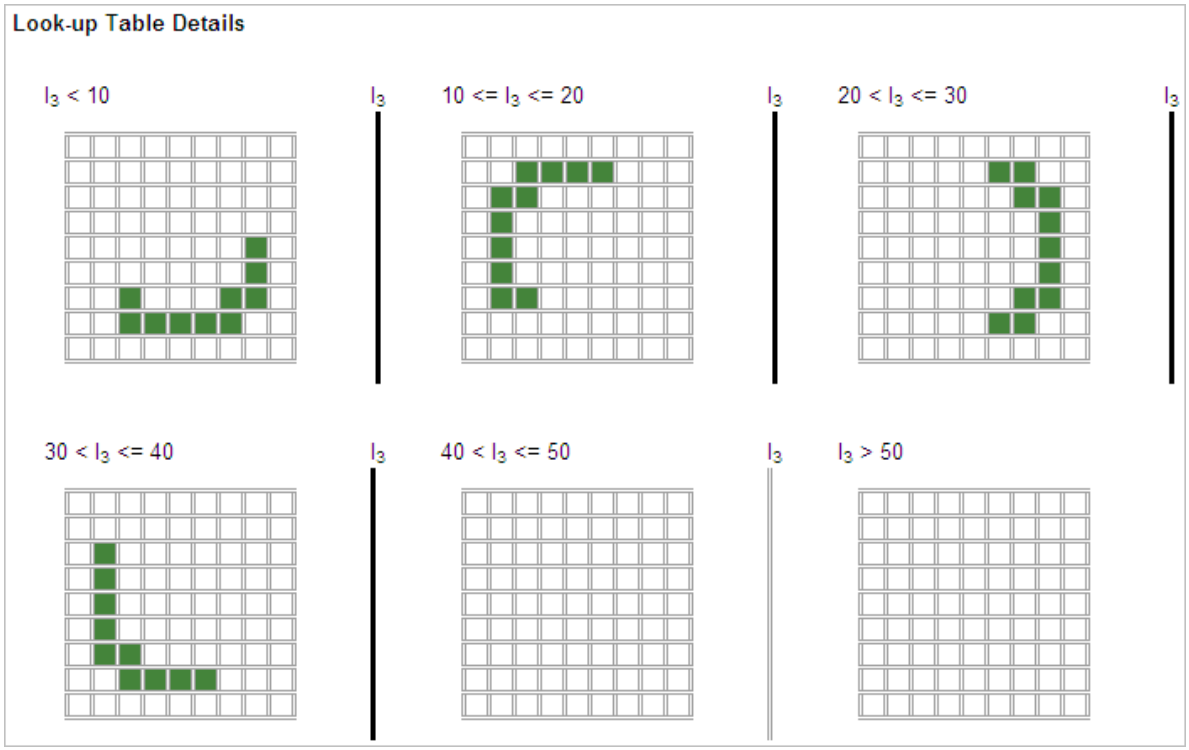
Lookup_n-D block "n-D Lookup Table"

Parent: /ex_mc_reports_three_d_lookup_table
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	0
Look-up Table	6% (42/726) interpolation/extrapolation intervals

Table map was not generated due to the table size.
Force Map Generation.

Instead of a two-dimensional table, the link `Force Map Generation` displays the following tables:



Lookup table coverage for a three-dimensional lookup table block is reported as a set of two-dimensional tables.

The vertical bars represent the exact *z* index values: 10, 20, 30, 40, 50. If a vertical bar is bold, this indicates that at least one block input was equal to the exact index value it represents during the simulation. Click a bar to get a coverage report for the exact index value that bar represents.

You can report lookup table coverage for lookup tables of any dimension. Coverage for four-dimensional tables is reported as sets of three-dimensional sets, like those in the preceding example. Five-dimensional tables are reported as sets of sets of three-dimensional sets, and so on.

Block Reduction

All model coverage reports indicate the status of the Simulink **Block reduction** parameter at the beginning of the report. In the following example, you set **Force block reduction off**.

Simulation Optimization Options

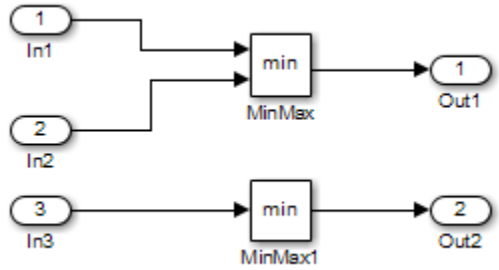
Default parameter behavior	tunable
Block reduction	forced off
Conditional branch optimization	on

In the next example, you enabled the Simulink **Block reduction** parameter, and you did not set **Force block reduction off**.

Simulation Optimization Options

Default parameter behavior	tunable
Block reduction	on
Conditional branch optimization	on

Consider the following model where the simulation does not execute the MinMax1 block because there is only one input — `In3`.



If you set **Force block reduction off**, the report contains no coverage data for this block because the minimum input to the MinMax1 block is always 1.

If you do not set **Force block reduction off**, the report contains no coverage data for reduced blocks.

Relational Boundary

On the “Coverage Pane” on page 3-2 of the Configuration Parameters dialog box, if you select the **Relational Boundary** coverage metric, the software creates a Relational Boundary table in the model coverage report for each model object that is supported for this coverage. The table applies to the explicit or implicit relational operation involved in the model object. For more information, see:

- “Relational Boundary Coverage” on page 1-8.
- The **Relational Boundary** column in “Model Objects That Receive Coverage” on page 2-2.

The tables below show the relational boundary coverage report for the relation `input1 <= input2`. The appearance of the tables depend on the operand data type.

- “Integers” on page 6-38
- “Fixed point” on page 6-39
- “Floating point” on page 6-40

Integers

If both operands are integers (or if one operand is an integer and the other a Boolean), the table appears as follows.

Relational Boundary

input1 - input2	33%
-1	0/51
0	51/51
+1	0/51

For a relational operation such as *operand_1* `<=` *operand_2*:

- The first row states the two operands in the form *operand_1* `-` *operand_2*.
- The second row states the number of times during the simulation that *operand_1* `-` *operand_2* is equal to -1.
- The third row states the number of times during the simulation that *operand_1* is equal to *operand_2*.
- The fourth row states the number of times during the simulation that *operand_1* `-` *operand_2* is equal to 1.

Fixed point

If one of the operands has fixed-point type and the other operand is either a fixed point or an integer, the table appears as follows. LSB represents the value of the least significant bit. For more information, see “Precision” (Fixed-Point Designer). If the two operands have different precision, the smaller value of precision is used.

Relational Boundary

input1 - input2	33%
-LSB	51/51
0	0/51
+LSB	0/51

For a relational operation such as *operand_1* `<=` *operand_2*:

- The first row states the two operands in the form *operand_1* `-` *operand_2*.
- The second row states the number of times during the simulation that *operand_1* `-` *operand_2* is equal to `-LSB`.
- The third row states the number of times during the simulation that *operand_1* is equal to *operand_2*.
- The fourth row states the number of times during the simulation that *operand_1* `-` *operand_2* is equal to `LSB`.

Floating point

If one of the operands has floating-point type, the table appears as follows. `tol` represents a value computed using the input values and a tolerance that you specify. If you do not specify a tolerance, the default values are used. For more information, see “Relational Boundary Coverage” on page 1-8.

Relational Boundary

input1 - input2	50%
[-tol..0]	51/51
(0..tol]	0/51

For a relational operation such as *operand_1* <= *operand_2*:

- The first row states the two operands in the form *operand_1* - *operand_2*.
- The second row states the number of times during the simulation that *operand_1* - *operand_2* has values in the range `[-tol..0]`.
- The third row states the number of times during the simulation that *operand_1* - *operand_2* has values in the range `(0..tol]` during the simulation.

The appearance of this table changes according to the relational operator in the block. Depending on the relational operator, the value of *operand_1* - *operand_2* equal to 0 is either:

- Excluded from relational boundary coverage.
- Included in the region above the relational boundary.
- Included in the region below the relational boundary.

Relational Operator	Report Format	Explanation
==	`[-tol..0)`	0 is excluded.
	`(0..tol]`	
!=	`[-tol..0)`	0 is excluded.
	`(0..tol]`	
<=	`[-tol..0]`	0 is included in the region below the relational boundary.
	`(0..tol]`	
<	`[-tol..0)`	0 is included in the region above the relational boundary.
	`[0..tol]`	
>=	`[-tol..0)`	0 is included in the region above the relational boundary.
	`[0..tol]`	
>	`[-tol..0]`	0 is included in the region below the relational boundary.
	`(0..tol]`	

0 is included below the relational boundary for <= but above the relational boundary for <. This rule is consistent with decision coverage. For instance:

- For the relation `input1 <= input2`, the decision is true if `input1` is less than or equal to `input2`. < and = are grouped together. Therefore, 0 lies in the region below the relational boundary.
- For the relation `input1 < input2`, the decision is true only if `input1` is less than `input2`. > and = are grouped together. Therefore, 0 lies in the region above the relational boundary.

Saturate on Integer Overflow Analysis

On the “Coverage Pane” on page 3-2 of the Configuration Parameters dialog box, if you select the **Saturate on integer overflow** coverage metric, the software creates a Saturation on Overflow analyzed table in the model coverage report. The software creates the table for each block with the **Saturate on integer overflow** parameter selected.

The Saturation on Overflow analyzed table lists the number of times a block saturates on integer overflow, indicating a true decision. If the block does not saturate on integer overflow, the table indicates a false decision. Outcomes that do not occur are in red highlighted table rows.

The following graphic shows the Saturation on Overflow analyzed table for the MinMax block in the Mixing & Combustion subsystem of the Engine Gas Dynamics subsystem in the `sldemo_fuelsys` example model.

MinMax block "MinMax"

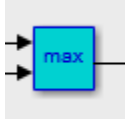
Parent: sldemo_fuelsys/Engine Gas Dynamics/Mixing & Combustion
Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	0
Saturation on Overflow	50% (1/2) objective outcomes

Saturation on Overflow analyzed:

Saturate on integer overflow	50%
false	204508/204508
true	0/204508

To display and highlight the block in question, click the block name at the top of the section containing the block's Saturation on Overflow analyzed table.



Signal Range Analysis

If you select the **Signal Range** coverage metric, the software creates a Signal Range Analysis section at the bottom of the model coverage report. This section lists the maximum and minimum signal values for each output signal in the model measured during simulation.

Access the Signal Range Analysis report quickly with the *Signal Ranges* link in the nonscrolling region at the top of the model coverage report, as shown below in the `sldemo_fuelsys` example model report.

Signal Ranges

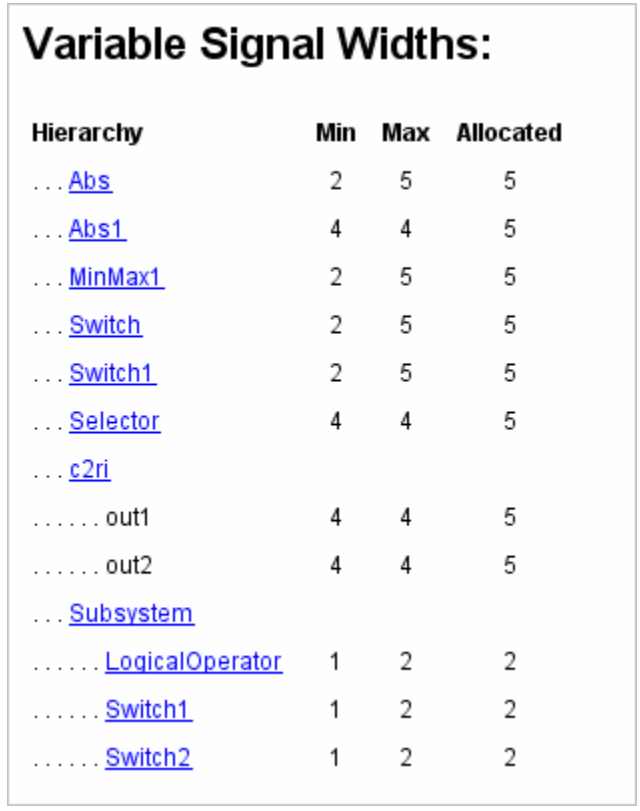
Hierarchy	Min	Max
sldemo_fuelsys		
. . . Engine_Speed_Selector	300	300
. . . MAP_Selector	0.405559	0.889674
. . . O2_Voltage_Selector	0.456832	1
. . . Throttle_Angle_Selector	10	20
. . . Constant2	0	0
. . . Constant3	12	12
. . . Constant4	0	0
. . . Constant5	0	0
. . . EGO Fault Switch	1	1
. . . Engine Speed	300	300
. . . Engine Speed Fault Switch	1	1
. . . MAP Fault Switch	1	1
. . . Throttle Angle Fault Switch	1	1
. . . Engine Gas Dynamics		
. . . . . Mixing & Combustion		

Each block is reported in hierarchical fashion; child blocks appear directly under parent blocks. Each block name in the *Signal Ranges* report is a link. For example, select the `EGO sensor` link to display this block highlighted in its native diagram.

Signal Size Coverage for Variable-Dimension Signals

If you select **Signal Size**, the software creates a Variable Signal Widths section after the Signal Ranges data in the model coverage report. This section lists the maximum and minimum signal sizes for all output ports in the model that have variable-size signals. It also lists the memory that Simulink allocated for that signal, as measured during simulation. This list does *not* include signals whose size does not vary during simulation.

The following example shows the Variable Signal Widths section in a coverage report. In this example, the Abs block signal size varied from 2 to 5, with an allocation of 5.



Variable Signal Widths:

Hierarchy	Min	Max	Allocated
. . . Abs	2	5	5
. . . Abs1	4	4	5
. . . MinMax1	2	5	5
. . . Switch	2	5	5
. . . Switch1	2	5	5
. . . Selector	4	4	5
. . . c2ri			
. . . . . out1	4	4	5
. . . . . out2	4	4	5
. . . Subsystem			
. . . . . LogicalOperator	1	2	2
. . . . . Switch1	1	2	2
. . . . . Switch2	1	2	2

Each block is reported in hierarchical fashion; child blocks appear directly under parent blocks. Each block name in the Variable Signal Widths list is a link. Clicking on the link

highlights the corresponding block in the Simulink Editor. After the analysis, the variable-size signals have a wider line design.

Simulink Design Verifier Coverage

If you select **Objectives and Constraints**, the analysis collects coverage data for all Simulink Design Verifier blocks in your model.

For an example of how this works, open the `sldvdemo_debounce_testobjblks` model.

This model contains two Test Objective blocks:

- The True block defines a property that the signal have a value of 2.
- The Edge block, inside the Masked Objective subsystem, describes the property where the output of the AND block in the Masked Objective subsystem changes from 2 to 1.

The Simulink Design Verifier software analyzes this model and produces a harness model that contains test cases that achieve certain test objectives. To see if the original model achieves those objectives, simulate the harness model and collect model coverage data. The model coverage tool analyzes any decision points or values within an interval that you specify in the Test Objective block.

In this example, the coverage report shows that you achieved 100% coverage of the True block because the signal value was 2 at least once. The signal value was 2 in 6 out of 14 time steps.

Design Verifier Test Objective block "True"

Justify or Exclude

Parent: /sldvdemo_debounce_testobjblks

Uncovered Links:

Metric	Coverage
Test Objective	0% (0/1) objective outcomes

Test Objective analyzed

2	0/1001

The input signal to the Edge block achieved a value of `True` once out of 14 time steps.

Design Verifier Test Objective block "Edge"

Justify or Exclude

Parent: sldvdemo_debounce_testobjblks/Masked Objective

Uncovered Links:

Metric	Coverage
Test Objective	0% (0/1) objective outcomes

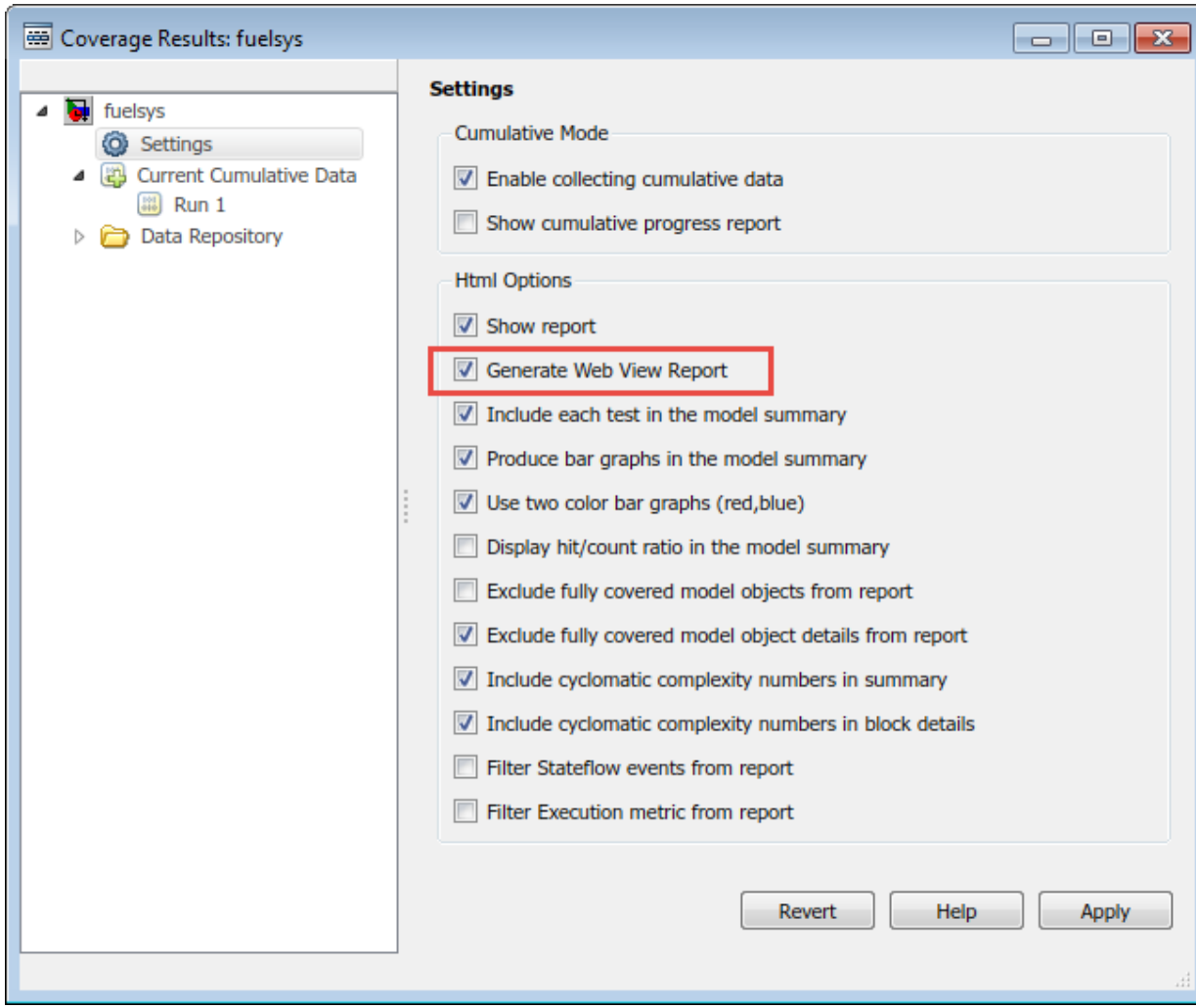
Test Objective analyzed

T	0/1001

Export Model Coverage Web View

You can export a Model Coverage Web View for your model. A Web View is an interactive rendition of a model that you can view in a Web browser. A Model Coverage Web View includes model coverage highlighting and analysis information from the Coverage Display Window, as described in “View Coverage Results in a Model” on page 5-12.

Use the Results Explorer to generate a Model Coverage Web View. After you record coverage, select **Analysis > Coverage > Open Results Explorer**. In the Results Explorer, open the **Settings**, select **Generate Web View Report**, and click **Apply**.



Next, select the Current Cumulative Data click **Generate report**.

When you generate a coverage report for your model with these settings enabled, the software generates a Model Coverage Web View that you can open in a browser. To see model coverage information for a block in a Model Coverage Web View, click that block. The model coverage data appears in the **Informer** pane, below the model.

For more information, see “Web Views” (Simulink Report Generator).

7

Excluding Model Objects from Coverage

- “Coverage Filtering” on page 7-2
- “Coverage Filter Rules and Files” on page 7-4
- “Model Objects to Filter from Coverage” on page 7-6
- “Create, Edit, and View Coverage Filter Rules” on page 7-7
- “Coverage Filter Viewer” on page 7-13

Coverage Filtering

In this section...
“When to Use Coverage Filtering” on page 7-2 “What Is Coverage Filtering?” on page 7-2

When to Use Coverage Filtering

Use coverage filtering to facilitate a bottom-up approach to recording model coverage. If you have a large model, there can be design elements that intentionally do not record 100% coverage. You can also have several design elements that you require to record 100% coverage but that do not achieve 100% coverage. You can temporarily or permanently eliminate these elements from coverage recording to focus on a subset of objects for testing and modification.

You can then iterate more efficiently—focus on a small issue, fix it, and then move on to resolve the next small issue. Before recording coverage for the entire model, you can resolve missing coverage issues within individual parts of the model.

What Is Coverage Filtering?

Coverage filtering enables you to exclude certain model objects from model coverage reporting after you simulate your Simulink model. You specify which objects you want to filter from coverage recording. There are two modes of filtering, Excluded and Justified.

Excluded objects do not contribute to coverage reports. After you specify the objects to exclude when you simulate your model, the coverage report does not record coverage for those objects.

Justified objects do contribute to coverage reports. After you specify the objects to justify when you simulate your model, the coverage report considers these blocks as achieving 100% coverage, and they appear light blue in the “Coverage Summary” on page 6-12.

Summary

Model Hierarchy/Complexity	Test 1							
	D1			C1		MCDC		Execution
1. slvnvdemo_covfilt	29	52%		40%		50%		13%
2. . . . Mode Logic	13	86%		75%		50%		NA
3. . . . . SF: Mode Logic	12	86%		75%		50%		NA
4. . . . . SF: Clipped	6	100%		NA		NA		NA
5. . . . . SF: Full	2	25%		NA		NA		NA

In the “Details” on page 6-14 section of the coverage report, justified objects show their coverage outcomes as ((covered outcomes + justified outcomes)/possible decisions).

4. State "Clipped"

Justified (Remove this rule)
Justification rationale: Justification rationale
Parent: slvnvdemo_covfilt/Mode Logic
Uncovered Links:

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	2	6
Decision (D1)	100% ((2+2)/4) decision outcomes	100% ((5+7)/12) decision outcomes

To filter objects, see “Create, Edit, and View Coverage Filter Rules” on page 7-7 and “Creating and Using Coverage Filters”.

Coverage Filter Rules and Files

In this section...
“What Is a Coverage Filter Rule?” on page 7-4
“What Is a Coverage Filter File?” on page 7-4

What Is a Coverage Filter Rule?

A coverage filter rule specifies a model object, a set of objects, or lines of code that you want to exclude from coverage recording or that you want to justify for coverage.

Each coverage filter rule includes the following fields:

- **Name**—Name or path of the object to filter from coverage
- **Type**—Whether a specific object is filtered or all objects of a given type are filtered
- **Mode**—Whether the object to be filtered is `Excluded` or `Justified`

  Coverage reports do not include `Excluded` blocks. The coverage reports assume that `Justified` blocks receive full coverage, but show that they are distinct from other covered blocks in the coverage report.
- **Rationale**—An optional description that describes why this object is filtered from coverage

What Is a Coverage Filter File?

A coverage filter file is a set of coverage filter rules. Each rule specifies one or more objects or lines of code to exclude from coverage recording.

To apply the coverage filter rules after coverage recording, you create coverage filter rules or load an existing coverage filter file. After you create the coverage filter rules, the specified objects or lines of code are excluded from coverage when you generate a report. You can reuse a coverage filter file for several Simulink models. However, a model can have only one attached coverage filter file.

When you make changes to the coverage filter rules after you record coverage, you can update the coverage report without needing to resimulate your model. After you make changes, click **Apply** and then **Generate Report** in the Coverage Filter Viewer to update the report.

If you use the default file name for the active model, and the coverage filter file exists on the MATLAB path, you see the coverage filter rules each time that you open the model. To save your current coverage filter rules to a file, click **Save filter**. To load an existing coverage filter file, click **Load filter**

Model Objects to Filter from Coverage

In your model, the objects that you can filter from coverage recording are:

- Simulink blocks that receive coverage, including MATLAB Function blocks
- Subsystems and their contents. When you exclude a subsystem from coverage recording, none of the objects inside the subsystem record coverage.
- Individual library-linked blocks or charts
- All reference blocks linked to a library
- Stateflow charts, subcharts, states, transitions, and events

For a complete list of model objects that receive coverage, see “Model Objects That Receive Coverage” on page 2-2.

Create, Edit, and View Coverage Filter Rules

In this section...
“Create and Edit Coverage Filter Rules” on page 7-7
“Save Coverage Filter to File” on page 7-10
“Load Coverage Filter File” on page 7-11
“Update the Report with the Current Filter Settings” on page 7-11
“View Coverage Filter Rules in Your Model” on page 7-11
“View Coverage Filter Rules in Your Model” on page 7-12

Create and Edit Coverage Filter Rules

- “Create a Coverage Filter Rule” on page 7-7
- “Select the Filtering Mode” on page 7-8
- “Add Rationale to a Coverage Filter Rule” on page 7-8
- “Justify Dead Logic from Simulink Design Verifier Dead Logic Analysis” on page 7-9

Create a Coverage Filter Rule

To create a coverage filter rule:

1. In the **Coverage** pane of the Configuration Parameters dialog box, enable model coverage.
2. To record coverage results, simulate the model.
3. Create a new filter rule in one of two ways:
   - In the model window, right-click a model object and select **Coverage** > **Exclude**.
   - In the Details section of the Coverage Report, click **Justify or Exclude** for a model object.

Depending on which option you select, the **Type** field in the “Coverage Filter Viewer” on page 7-13 is set for the coverage filter rule you selected. You cannot override the value in the **Type** field.

Select the Filtering Mode

When you create a filtering rule, the default filtering mode is `Excluded`. Excluded objects do not appear in the coverage reports. You can also set the filtering mode to `Justified`. Justified blocks appear as achieving 100% coverage.

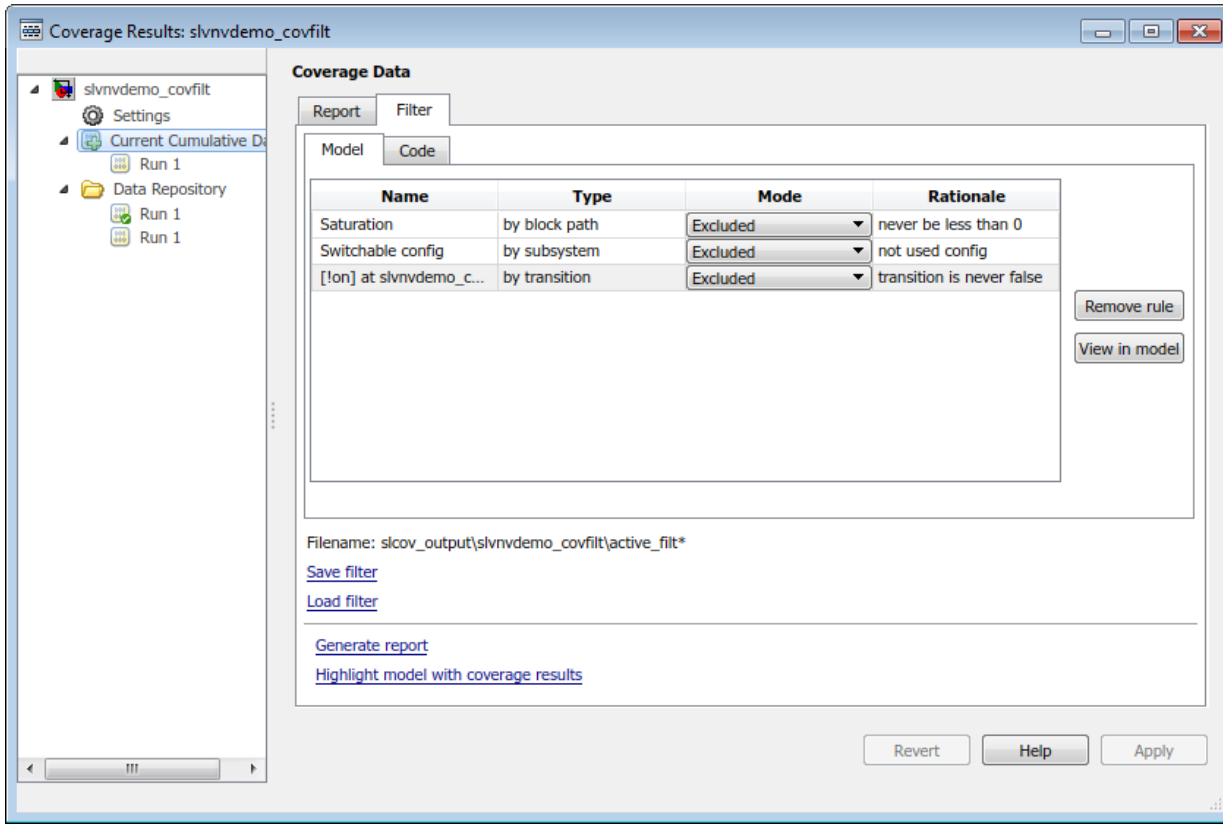
For more information, see “Coverage Filtering” on page 7-2.

Add Rationale to a Coverage Filter Rule

Optionally, you can add text that describes why you exclude that object or objects from coverage recording. This information can be useful to others who review the coverage for your model. When you add a coverage filter rule, the Coverage Filter Viewer opens. To add the rationale:

1. Double-click the Rationale field for the rule.
2. Delete the existing text.
3. Add the rationale for excluding this object.

The following graphic shows examples of text in the **Rationale** field.



Note The **Rationale** field and **Mode** field are the only coverage filter rule fields that you can edit in the Coverage Filter Viewer.

After you add a new coverage filter rule or edit an existing coverage filter rule, click **Apply** to enable the **Generate report** and **Highlight model with coverage results** links.

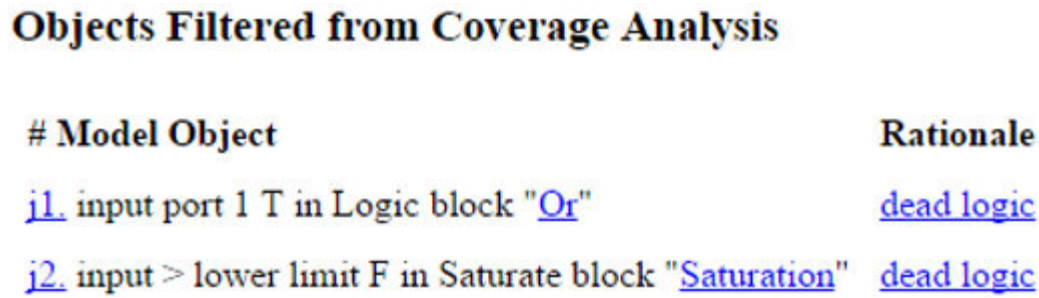
Justify Dead Logic from Simulink Design Verifier Dead Logic Analysis

You can create justification rules in the Coverage Results Explorer using the dead logic detected during a Simulink Design Verifier Dead Logic Analysis.

1. Open the Coverage Results Explorer from the Simulink menu. Select **Analysis > Coverage > Open Results Explorer**.
2. Click **Current Cumulative Data** to access the coverage results for the current simulation and navigate to the **Filter** tab.
3. Click **Make justification filter rules for dead logic (using Simulink Design Verifier)**.

   Simulink Design Verifier runs the Dead Logic Analysis and populates the list of filters.
4. Click **Generate report**.

   The justified rules from the previous step are shown in the **Objects Filtered from Coverage Analysis** section at the beginning of the report. To navigate to the rules' corresponding items in the **Details** section of the report, use the hyperlinks in the rule descriptions. Clicking the hyperlinks in the **Rationale** column navigates to the Coverage Results Explorer.


Objects Filtered from Coverage Analysis

#	Model Object	Rationale
j1.	input port 1 T in Logic block "Or"	dead logic
j2.	input > lower limit F in Saturate block "Saturation"	dead logic

   You can add justification rules for elements that do not receive coverage to the filter by clicking in the **Details** section of the report.

Save Coverage Filter to File

After you define the coverage filter rules, save the rules to a file so that you can reuse them with this model or with other models. By default, coverage filter files are named `<model_name>_covfilter.cvf`.

In the Current Cumulative Data section of the Coverage Filter Viewer:

1. Click **Save filter**.
2. Specify a file name and folder for the filter file and click **Save**.

If you make multiple changes to the coverage filter rules, apply the changes to the coverage filter file each time.

Load Coverage Filter File

After you save a coverage filter file, you can load the coverage filter file for other models.

In the Current Cumulative Data section of the Coverage Filter Viewer:

1. Click **Load filter**.
2. Navigate to the filter file and click **Open**.

You can have only one coverage filter file attached to a model at a time. If you attach a different coverage filter file, the newly attached file replaces the previously attached file.

Two or more models can have the same coverage filter file attached. If a model has an attached filter file that contains coverage filter rules for specific objects in a different model, those rules are ignored during coverage recording.

Update the Report with the Current Filter Settings

If you change the filtering settings or add filters after you simulate the model, you can update the coverage report and model highlighting without resimulating the model. After you have simulated the model, in the Current Cumulative Data section of the Coverage Filter Viewer:

1. **Apply** or **Revert** any changes you have made.
2. Click **Generate Report**.

View Coverage Filter Rules in Your Model

Whenever you define a coverage filter rule or remove an existing coverage filter rule, the Coverage Filter Viewer opens. This dialog box lists the coverage filter rules for your model. For more information, see “Coverage Filter Viewer” on page 7-13.

The Coverage Filter Viewer is available in the Current Cumulative Data section of the **Coverage Results** viewer. Alternatively, you can right-click anywhere in the model window and select **Coverage > Open Filter Viewer**

If you are inside a subsystem, you can view any coverage filter rule attached to the subsystem. To open the Coverage Filter Viewer, right-click any object inside the subsystem and select **Coverage > Show filter parent**.

View Coverage Filter Rules in Your Model

Whenever you define a coverage filter rule or remove an existing coverage filter rule, the Coverage Filter Viewer opens. This dialog box lists all the coverage filter rules for your model. For more information, see “Coverage Filter Viewer” on page 7-13.

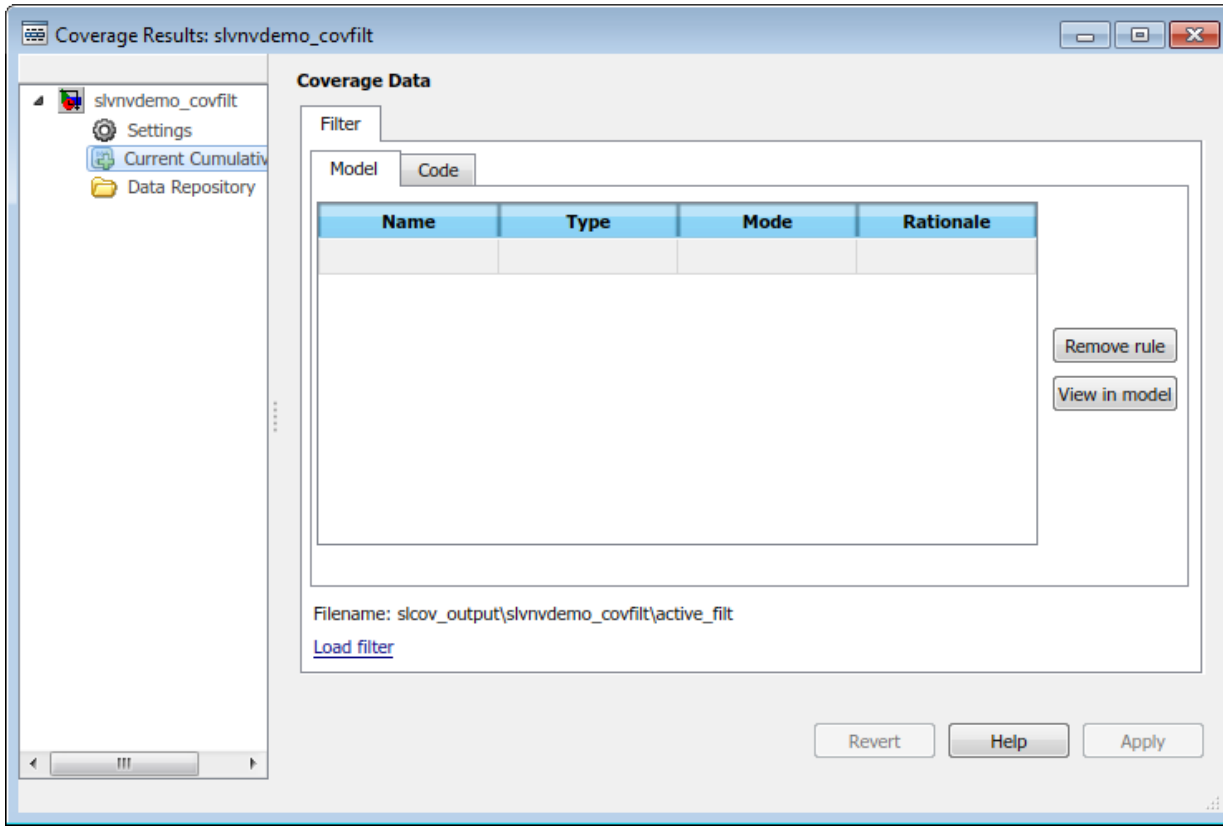
To open the Coverage Filter Viewer, right-click anywhere in the model window and select **Coverage > Open Filter Viewer**.

If you are inside a subsystem, you can view any coverage filter rule attached to the subsystem. To open the Coverage Filter Viewer, right-click any object inside the subsystem and select **Coverage > Show filter parent**.

Coverage Filter Viewer

In the Coverage Filter Viewer, you can:

- Review and manage the coverage filter rules for your Simulink model.
- Load or save coverage filter files for your model.
- Navigate to the model to create additional coverage filter rules.



To	Action
Navigate to a model object associated with a rule.	1 Select the rule. 2 Click **View in model**.

To	Action
Delete a rule.	1 Select the rule. 2 Click **Remove rule**.
Save the current rules to a file.	1 Click **Save filter**. 2 Specify a file name and folder for the filter file and click **Save**.
Load an existing coverage filter file.	1 Click **Load filter**. 2 Navigate to the filter file and click **Open**.
Update the current coverage report with the current filtering rules.	1 **Apply** or **Revert** any changes you have made. 2 Click **Generate Report**.

Automating Model Coverage Tasks

- “Commands for Automating Model Coverage Tasks” on page 8-2
- “Create Tests with cvtest” on page 8-3
- “Run Tests with cvsim” on page 8-5
- “Retrieve Coverage Details from Results” on page 8-7
- “” on page 8-7
- “Create HTML Reports with cvhtml” on page 8-8
- “Save Test Runs to File with cvsave” on page 8-9
- “Load Stored Coverage Test Results with cvload” on page 8-10
- “Use Coverage Commands in a Script” on page 8-11

Commands for Automating Model Coverage Tasks

Using model coverage commands lets you automate the entire model coverage process with MATLAB scripts. You can use model coverage commands to set up model coverage tests, execute them in simulation, and store and report the results.

Create Tests with cvtest

The `cvtest` command creates a test specification object. Once you create the object, you simulate it with the `cvsim` command.

The call to `cvtest` has the following default syntax:

```
cvto = cvtest(root)
```

`root` is the name of, or a handle to, a Simulink model or a subsystem of a model. `cvto` is a handle to the resulting test specification object. Only the specified model or subsystem and its descendants are subject to model coverage.

To create a test object with a specified label (used for reporting results):

```
cvto = cvtest(root, label)
```

To create a test with a setup command:

```
cvto = cvtest(root, label, setupcmd)
```

You execute the setup command in the base MATLAB workspace, just prior to running the instrumented simulation. Use this command for loading data prior to a test.

The returned `cvtest` object, `cvto`, has the following structure.

Field	Description
`id`	Read-only internal data-dictionary ID
`modelcov`	Read-only internal data-dictionary ID
`rootPath`	Name of the system or subsystem for analysis
`label`	String for reporting results
`setupCmd`	Command executed prior to simulation
`settings.condition`	Set to 1 for condition coverage
`settings.decision`	Set to 1 for decision coverage
`settings.designverifier`	Set to 1 for coverage for Simulink Design Verifier blocks.
`settings.mcdc`	Set to 1 for MCDC coverage

Field	Description
`settings.overflowsaturation`	Set to `1` for saturate on integer overflow coverage
`settings.sigrange`	Set to 1 for signal range coverage
`settings.sigsize`	Set to `1` for signal size coverage.
`settings.tableExec`	Set to 1 for lookup table coverage
`modelRefSettings.enable`	String specifying one of the following values: • `Off` — Disables coverage for all referenced models • `all` — Enables coverage for all referenced models • `filtered` — Enables coverage for only referenced models not listed in the `excludedModels` subfield
`modelRefSettings.excludeTopModel`	Set to 1 for excluding coverage for the top model
`modelRefSettings.excludedModels`	String specifying a comma-separated list of referenced models for which coverage is disabled when `modelRefSettings.enable` specifies `filtered`
`emlSettings.enableExternal`	Set to `1` to enable coverage for external program files called by MATLAB functions in your model
`sfcnSettings.enableSfcn`	Set to `1` to enable coverage for C/C++ S-Function blocks in your model.
`options.forceBlockReduction`	Set to `1` to override the Simulink **Block reduction** parameter if it is enabled.

Run Tests with cvsim

Use the `cvsim` command to simulate a test specification object.

The call to `cvsim` has the following default syntax:

```
cvdo = cvsim(cvto)
```

This command executes the `cvtest` object `cvto` by simulating the corresponding model. `cvsim` returns the coverage results in the `cvdata` object `cvdo`. When recording coverage for multiple models in a hierarchy, `cvsim` returns its results in a `cv.cvdatagroup` object.

You can also control the simulation in a `cvsim` command by setting model parameters for the Simulink `sim` command to apply during simulation:

- The following command executes the test object `cvto` and simulates the model using the default model parameters. The `cvsim` function returns the coverage results in the `cvdata` object `cvdo` and returns the simulation outputs in a `Simulink.SimulationOutput` object `simOut`:

  ```
  [cvdo,simOut] = cvsim(cvto)
  ```

- The following commands create a structure, `paramStruct`, that specifies the model parameters to use during the simulation. The first command specifies that the simulation collect decision, condition, and MCDC coverage for this model.

  ```
  paramStruct.CovMetricSettings  = 'dcm';
  paramStruct.SimulationMode     = 'rapid';
  paramStruct.AbsTol          = '1e-5';
  paramStruct.SaveState       = 'on';
  paramStruct.StateSaveName   = 'xoutNew';
  paramStruct.SaveOutput      = 'on';
  paramStruct.OutputSaveName = 'youtNew';
  ```

  Note For a complete list of model parameters, see “Model Parameters” (Simulink).

  The following `cvsim` command executes the test object `cvto` and simulates the model using the model parameter values specified in `paramStruct`:

  ```
  [cvdo,simOut] = cvsim(cvto,paramStruct);
  ```

You can also execute multiple test objects with the `cvsim` command. The following command executes a set of coverage test objects, `cvto1, cvto2, ...` using the default simulation parameters. `cvsim` returns the coverage results in a set of `cvdata` objects, `cvdo1, cvdo2, ...` and returns the simulation outputs in `simOut`.

```
[cvdo1, cvdo2, ..., simOut] = cvsim(cvto1, cvto2, ...)
```

Retrieve Coverage Details from Results

Simulink Coverage provides commands that allow you to retrieve specific coverage information from the `cvtest` object after you have simulated your model and recorded coverage. Use these commands to retrieve the specified coverage information for a block, subsystem, or Stateflow chart in your model or for the model itself:

- `complexityinfo` — Cyclomatic complexity coverage
- `conditioninfo` — Condition coverage
- `decisioninfo` — Decision coverage
- `mcdcinfo` — Modified condition/decision (MCDC) coverage
- `overflowsaturationinfo` — Saturate on integer overflow coverage
- `relationalboundaryinfo` — Relational boundary coverage
- `sigrangeinfo` — Signal range coverage
- `sigsizeinfo` — Signal size coverage
- `tableinfo` — Lookup Table block coverage
- `getCoverageinfo` — Coverage for Simulink Design Verifier blocks

The basic syntax of these functions is:

```
coverage = <coverage_type_prefix>info(cvdata_object, ...
    object, ignore_descendants)
```

- `coverage` — Multipart vector containing the retrieved coverage results for `object`
- `cvdata_object` — `cvdata` object that you create when you call `cvsim`
- `object` — Handle to a model or object in the model
- `ignore_descendants` — Flag to ignore coverage results in subsystems, referenced models, and Stateflow charts

???

Create HTML Reports with cvhtml

Once you run a test in simulation with `cvsim`, results are saved to `cv.cvdatagroup` or `cvdata` objects in the base MATLAB workspace. Use the `cvhtml` command to create an HTML report of these objects.

The following command creates an HTML report of the coverage results in the `cvdata` object `cvdo`. The results are written to the file `file` in the current MATLAB folder.

```
cvhtml(file, cvdo)
```

The following command creates a combined report of several `cvdata` objects:

```
cvhtml(file, cvdo1, cvdo2, ...)
```

The results from each object are displayed in a separate column of the HTML report. Each `cvdata` object must correspond to the same root model or subsystem, or the function produces errors.

Save Test Runs to File with cvsave

Once you run a test with `cvsim`, save its coverage tests and results to a file with the `cvsave` function:

```
cvsave(filename, model)
```

Save all the tests and results related to `model` in the text file `filename.cvt`:

```
cvsave(filename, cvto1, cvto2, ...)
```

Save the tests in the text file `filename.cvt`. Information about the referenced models is also saved.

You can save specified `cvdata` objects to file. The following example saves the tests, test results, and referenced models' structure in `cvdata` objects to the text file `filename.cvt`:

```
cvsave(filename, cvdo1, cvdo2, ...)
```

Load Stored Coverage Test Results with cvload

The `cvload` command loads into memory the coverage tests and results stored in a file by the `cvsave` command. The following example loads the tests and data stored in the text file `filename.cvt`:

```
[cvtos, cvdos] = cvload(filename)
```

The `cvtest` objects that are loaded are returned in `cvtos`, a cell array of `cvtest` objects. The `cvdata` objects that are loaded are returned in `cvdos`, a cell array of `cvdata` objects. `cvdos` has the same size as `cvtos`, but can contain empty elements if a particular test has no results.

In the following example, if `restoretotal` is 1, the cumulative results from prior runs are restored:

```
[cvtos, cvdos] = cvload(filename, restoretotal)
```

If `restoretotal` is unspecified or 0, the model's cumulative results are cleared.

cvload Special Considerations

When using the `cvload` command, be aware of the following considerations:

- When a model with the same name exists in the coverage database, only the compatible results are loaded from the file. They reference the existing model to prevent duplication.
- When the Simulink models referenced in the file are open but do not exist in the coverage database, the coverage tool resolves the links to the models that are already open.
- When you are loading several files that reference the same model, only the results that are consistent with the earlier files are loaded.

Use Coverage Commands in a Script

The following script demonstrates some common model coverage commands.

This script:

- Creates two data files to load before simulation.
- Creates two `cvtest` objects, `testObj1` and `testObj2`, and simulates them using the default model parameters. Each `cvtest` object uses the `setupCmd` property to load a data file before simulation.
- Enables decision, condition, and MCDC coverage.
- Retrieves the decision coverage results for the Adjustable Rate Limited subsystem.
- Uses `cvhtml` to display the coverage results for the two tests and the cumulative coverage.
- Compute cumulative coverage with the + operator and save the results

```
mdl = 'slvnvdemo_ratelim_harness';
mdl_subsys = 'slvnvdemo_ratelim_harness/Adjustable Rate Limiter';

open_system(mdl);
open_system(mdl_subsys);

t_gain = (0:0.02:2.0)'; u_gain = sin(2*pi*t_gain);
t_pos = [0;2]; u_pos = [1;1]; t_neg = [0;2]; u_neg = [-1;-1];
save('within_lim.mat','t_gain','u_gain','t_pos','u_pos', ...
    't_neg', 'u_neg');
t_gain = [0;2]; u_gain = [0;4]; t_pos = [0;1;1;2];
u_pos = [1;1;5;5]*0.02; t_neg = [0;2]; u_neg = [0;0];
save('rising_gain.mat','t_gain','u_gain','t_pos','u_pos', ...
    't_neg', 'u_neg');

testObj1               = cvtest(mdl_subsys);
testObj1.label         = 'Gain within slew limits';
testObj1.setupCmd      = 'load(''within_lim.mat'');';
testObj1.settings.mcdc = 1;
testObj1.settings.condition = 1;
testObj1.settings.decision = 1;

testObj2          = cvtest(mdl_subsys);
testObj2.label    = 'Rising gain that temporarily exceeds slew limit';
testObj2.setupCmd = 'load(''rising_gain.mat'');';
testObj2.settings.mcdc = 1;
testObj2.settings.condition = 1;
testObj2.settings.decision = 1;

[dataObj1,simOut1] = cvsim(testObj1);
decision_cov1 = decisioninfo(dataObj1,mdl_subsys);
percent_cov1 = 100 * decision_cov1(1) / decision_cov1(2)
cc_cov2 = complexityinfo(dataObj1, mdl_subsys);

[dataObj2,simOut2] = cvsim(testObj2,[0 2]);
decision_cov2 = decisioninfo(dataObj2,mdl_subsys);
```

```
percent_cov2 = 100 * decision_cov2(1) / decision_cov2(2)
cc_cov2 = complexityinfo(dataObj1, mdl_subsys);


cvhtml('ratelim_report',dataObj1,dataObj2);
cumulative = dataObj1+dataObj2;

cvsave('ratelim_testdata',cumulative);

close_system('slvnvdemo_ratelim_harness',0);
```

Component Verification

- “Component Verification” on page 9-2
- “Verify Generated Code for a Component” on page 9-7

Component Verification

In this section...
“Simulink Coverage Tools for Component Verification” on page 9-2
“Workflow for Component Verification” on page 9-3
“Verify a Component Independently of the Container Model” on page 9-4
“Verify a Model Block in the Context of the Container Model” on page 9-5

Using component verification, you can test a design component in your model with one of these approaches:

- System analysis. Within the context of the model that contains the component, you use systematic simulation of closed-loop controllers to verify components within a control system model. You can then test the control algorithms with your model.
- Component analysis. As standalone components, for a high level of confidence in the component algorithm, verify the component in isolation from the rest of the system.

  Verifying standalone components provides several advantages:

  - You can use the analysis to focus on portions of the design that you cannot test because of the physical limitations of the system being controlled.
  - For open-loop simulations, you can test the plant model without feedback control.
  - You can use this approach when the model is not yet available or when you need to simulate a control system model in accelerated mode for performance reasons.

Simulink Coverage Tools for Component Verification

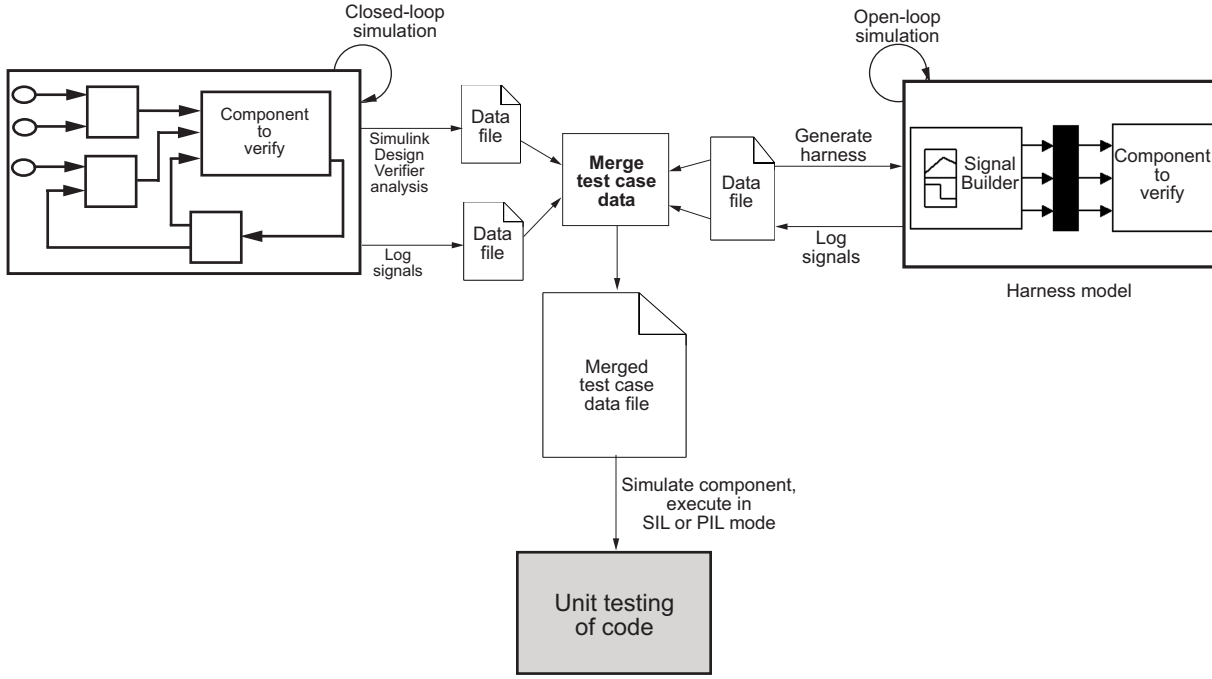
By isolating a component to verify and by using tools that the Simulink Coverage software provides, you create test cases to expand the scope of the testing for large models. You can:

- Achieve 100% model coverage — If certain model components do not record 100% coverage, the top-level model cannot achieve 100% coverage. By verifying these components individually, you can create test cases that fully specify the component interface, allowing the component to record 100% coverage.
- Debug the component — To verify that each model component satisfies the specified design requirements, you can create test cases that verify that specific components perform as they were designed to perform.

- Test the robustness of the component — To verify that a component handles unexpected inputs and calculations properly, you can create test cases that generate data. Then, test the error-handling capabilities in the component.

Workflow for Component Verification

This graphic illustrates two approaches for component verification.



1 Choose your approach for component verification:

- For closed-loop simulations, verify a component within the context of its container model by logging the signals to that component and storing them in a data file. If those signals do not constitute a complete test suite, generate a harness model and add or modify the test cases in the Signal Builder.
- For open-loop simulations, verify a component independently of the container model by extracting the component from its container model and creating a harness model for the extracted component. Add or modify test cases in the Signal Builder and log the signals to the component in the harness model.

2. Prepare component for verification.
3. Create and log test cases. You can also merge the test case data into a single data file.

   The data file contains the test case data for simulating the component. If you cannot achieve the expected results with a certain set of test cases, add new test cases or modify existing test cases in the data file. Merge the test cases into a single data file.

   Continue adding or modifying test cases until you achieve a test suite that satisfies your analysis goals.
4. Execute the test cases in software-in-the-loop or processor-in-the-loop mode.
5. After you have a complete test suite, you can:

   - Simulate the model and execute the test cases to:
     - Record coverage.
     - Record output values to make sure that you get the expected results.
   - Invoke the Code Generation Verification (CGV) API to execute the generated code for the model that contains the component in simulation, software-in-the-loop (SIL), or processor-in-the-loop (PIL) mode.

   Note To execute a model in different modes of execution, you use the CGV API to verify the numerical equivalence of results. See “Programmatic Code Generation Verification” (Embedded Coder).

Verify a Component Independently of the Container Model

Use component analysis to verify:

- Model blocks
- Atomic subsystems
- Stateflow atomic subcharts

1. Depending on the type of component, take one of the following actions:

   - Model blocks — Open the referenced model.
   - Atomic subsystems — Extract the contents of the subsystem into its own Simulink model.

- Atomic subcharts — Extract the contents of the Stateflow atomic subchart into its own Simulink model.

2. Create a harness model for:
   - The referenced model
   - The extracted model that contains the contents of the atomic subsystem or atomic subchart
3. Add or modify test cases in the Signal Builder of the harness model.
4. Log the input signals from the Signal Builder to the test unit.
5. Repeat steps 3 and 4 until you are satisfied with the test suite.
6. Merge the test case data into a single file.
7. Depending on your goals, take one of these actions:
   - Execute the test cases to:
     - Record coverage.
     - Record output values and make sure that they equal the expected values.
   - Invoke the Code Generation Verification (CGV) API to execute the test cases in software-in-the-loop (SIL) or processor-in-the-loop (PIL) mode on the generated code for the model that contains the component.

If the test cases do not achieve the expected results, repeat steps 3 through 5.

Verify a Model Block in the Context of the Container Model

Use system analysis to:

- Verify a Model block in the context of the block's container model.
- Analyze a closed-loop controller.

1. Log the input signals to the component by simulating the container model or analyze the model by using the Simulink Design Verifier software.
2. If you want to add test cases to your test suite or modify existing test cases, create a harness model with the logged signals.
3. Add or modify test cases in the Signal Builder in the harness model.
4. Log the input signals from the Signal Builder to the test unit.

5. Repeat steps 3 and 4 until you are satisfied with the test suite.
6. Merge the test case data into a single file.
7. Depending on your goals, do one of the following:
   - Execute the test cases to:
     - Record coverage.
     - Record output values and make sure that they equal the expected values.
   - Invoke the Code Generation Verification (CGV) API to execute the test cases in software-in-the-loop (SIL) or processor-in-the-loop (PIL) mode on the generated code for the model.

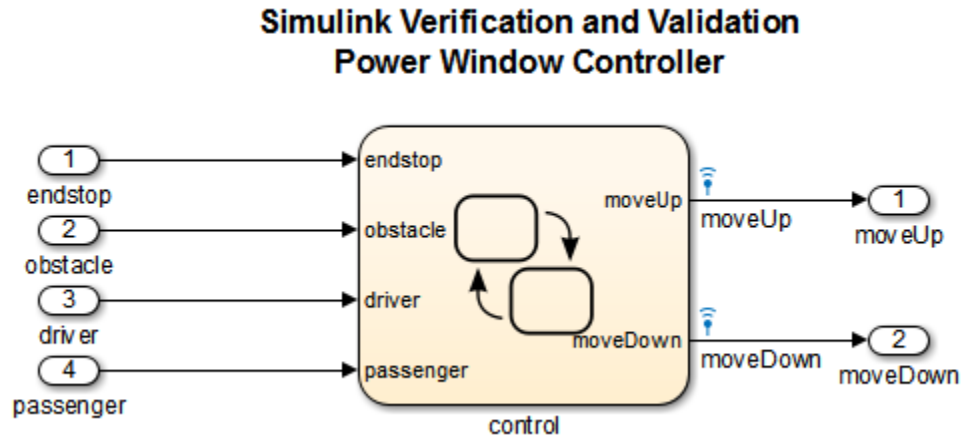
If the test cases do not achieve the expected results, repeat steps 3 through 5.

Verify Generated Code for a Component

This example shows how to verify generated code for a model component. You use component verification functions to create test cases and measure coverage for a referenced model. In addition, you execute the referenced model in both simulation mode and software-in-the-loop (SIL) or processor-in-the-loop (PIL) mode using the Code Generation Verification (CGV) API and then compare the results.

The component you verify is a Model block named `control`, which is part of the `power_window_control_system` subsystem in the `slvnvdemo_powerwindow` model.

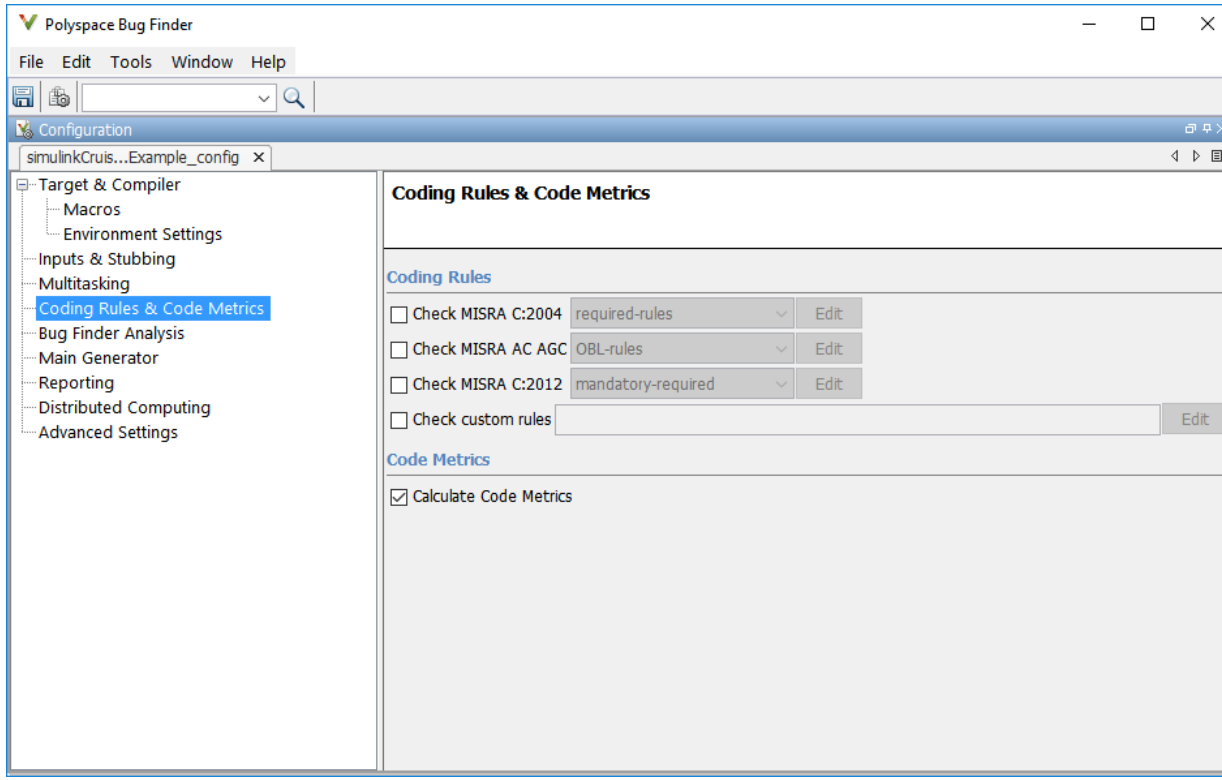
The Model block references the `slvnvdemo_powerwindow_controller` model.



The referenced model implements controller logic with a Stateflow® chart.

3 After the code is generated, right-click Compute target speed and select **Polyspace** > **Options**.

4 Click the **Configure** (Polyspace Bug Finder) button. This option allows you to choose more advanced Polyspace analysis options in the Polyspace configuration window.



5 On the same pane, select **Calculate Code Metrics**. This option turns on code metric calculations for your generated code.

6 Save and close the Polyspace configuration window.

7 From your model, right-click Compute target speed and select **Polyspace > Verify Code Generated For > Selected Subsystem**.

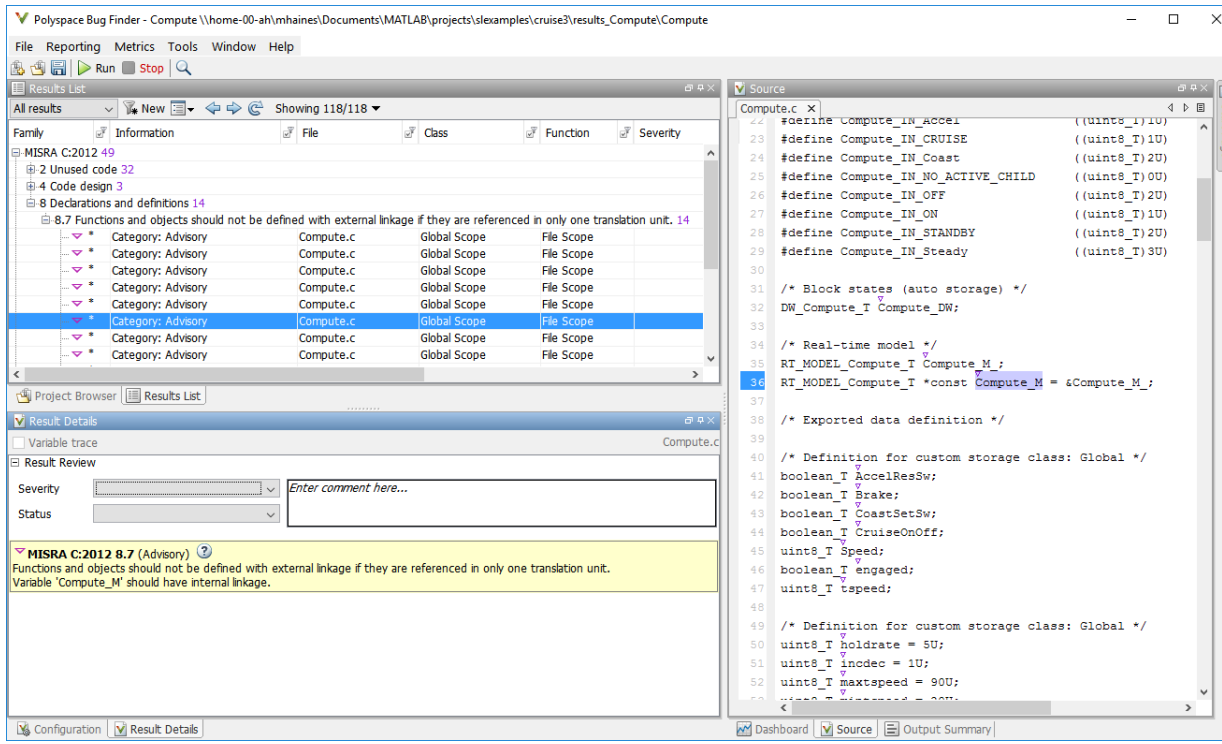
Polyspace Bug Finder analyzes the generated code for a subset of MISRA checks and defect checks. You can see the progress of the analysis in the MATLAB Command Window. Once the analysis is finished, the Polyspace environment opens.

Review Results

After you run a Polyspace analysis of your generated code, the Polyspace environment shows you the results of the static code analysis. There are 50 MISRA C:2012 coding rule violations in your generated code.

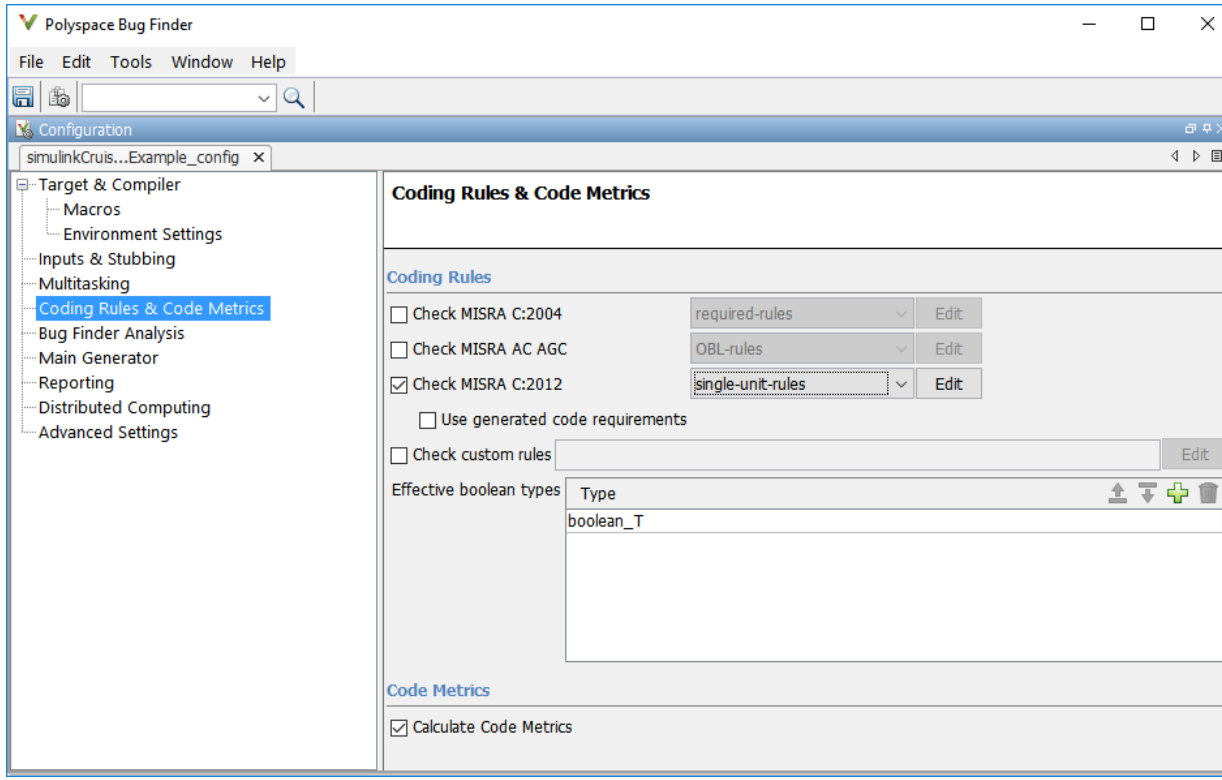
1. Expand the tree for rule 8.7 and click through the different results.

   Rule 8.7 states that functions and objects should not be global if the function or object is local. As you click through the 8.7 violations, you can see that these results refer to variables that other components also use, such as `CruiseOnOff`. You can annotate your code or your model to justify every result. But, because this model is a unit in a larger program, you can also change the configuration of the analysis to check only a subset of MISRA rules.



2. In your model, right-click Compute target speed and select **Polyspace** > **Options**.
3. Set the **Settings from** (Polyspace Bug Finder) option to `Project configuration`. This option allows you to choose a subset of MISRA rules in the Polyspace configuration.
4. Click the **Configure** button.
5. In the Polyspace Configuration window, on the **Coding Rules & Code Metrics** pane, select the check box **Check MISRA C:2012** and from the drop-down list, select

`single-unit-rules`. Now, Polyspace checks only the MISRA C:2012 rules that are applicable to a single unit.



6 Save and close the Polyspace configuration window.

7 Rerun the analysis with the new configuration.

When the Polyspace environment reopens, there are no MISRA results, only code metric results. The rules Polyspace showed previously were found because the model was analyzed by itself. When you limited the rules Polyspace checked to the single-unit subset, no violations were found.

When this model is integrated with its parent model, you can add the rest of the MISRA C:2012 rules.

Generate Report

To demonstrate compliance with MISRA C:2012 and report on your generated code metrics, you must export your results. This section shows you how to generate a report after the analysis. If you want to generate a report every time you run an analysis, see `Generate report`.

1. If they are not open already, open your results in the Polyspace environment.
2. From the toolbar, select **Reporting** > **Run Report**.
3. Select **BugFinderSummary** as your report type.
4. Click **Run Report**.

   The report is saved in the same folder as your results.
5. To open the report, select **Reporting** > **Open Report**.

See Also

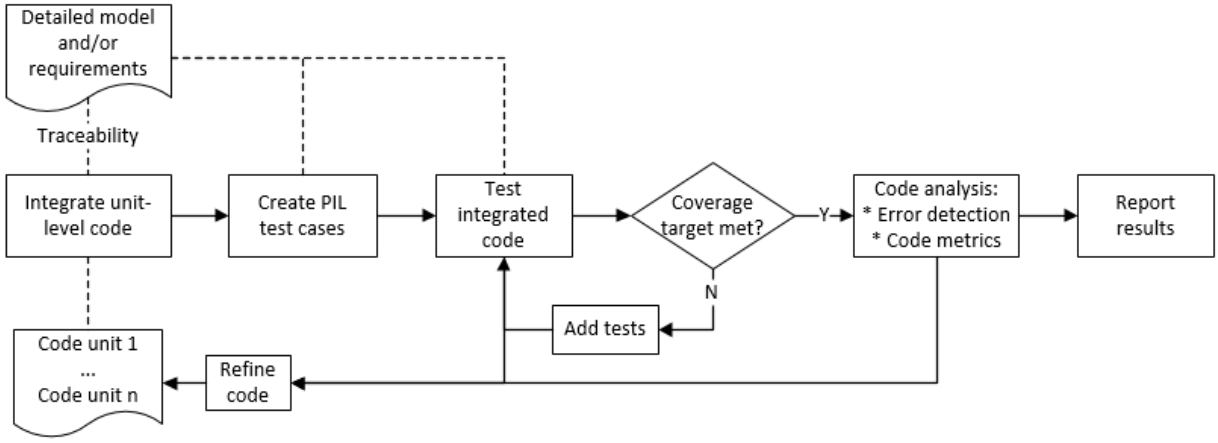
Related Examples

- “Analyze Generated Code Using Polyspace Bug Finder” (Polyspace Bug Finder)
- “Test Two Simulations for Equivalence” (Simulink Test)
- “Export Test Results and Generate Reports” (Simulink Test)

Module Verification and Testing Processor-in-the-Loop

Module Verification and Testing Processor-in-the-Loop Overview

Module verification involves testing and analyzing code at a system level, integrating generated code from your model, handwritten code, and legacy code. Module verification often includes generating code that executes on a target object, rather than the desktop environment. Analyze the code to resolve errors and evaluate key metrics. Test the integrated system using new requirements-based tests and system-level tests from your model. Collect coverage on these tests and add tests to meet coverage targets.



See Also

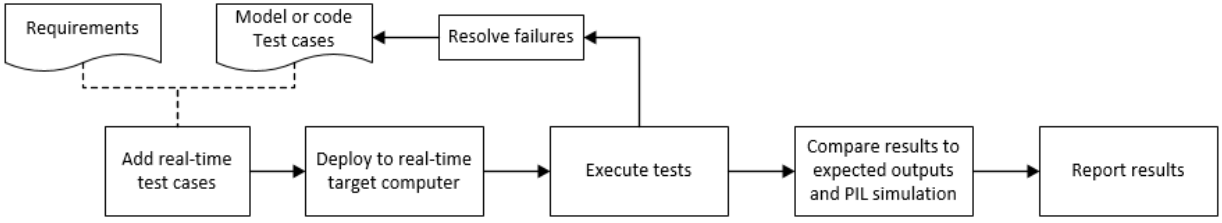
Related Examples

- “Test Two Simulations for Equivalence” (Simulink Test)
- “Analyze Generated Code Using Polyspace Bug Finder” (Polyspace Bug Finder)

Test a Model in Real Time

Real-Time Testing and Testing Production Models Overview

Real-time testing assesses the system while including the effects of timers, physical signals, and target hardware. Sweep through parameter values on the target, verify system operation during execution, and verify expected results in the desktop environment. Systems that have been verified on target hardware often exist in a change-controlled state. You can test these systems without modifying them by using isolated simulation and analysis environments.



See Also

Related Examples

- “Create and Run Real-Time Application from Simulink Model” (Simulink Real-Time)
- “Test Models in Real Time” (Simulink Test)
- “Run-Time Assessments” (Simulink Test)