



# **HiLiTE**

## **Test Automation Suite**

# **User Guide**

**Honeywell Laboratories Document No.: AES-R04-004**

**Version 7.8**

**(for use with HiLiTE Versions 7.8 and higher)**

**January 28, 2011**

**Honeywell**

Honeywell International, Inc.  
Aerospace Advanced Technology  
1985 Douglas Drive  
Golden Valley, MN 55422

— **Honeywell Confidential** —

HONEYWELL CONFIDENTIAL: This copyrighted work and all information are the property of Honeywell International Inc., contain trade secrets and may not, in whole or in part, be used, duplicated, or disclosed for any purpose without prior written permission of Honeywell International Inc.

Copyright © 2005-2011 Honeywell International Inc. All Rights Reserved.

# Contents

<b>Contents</b>	<b>i</b>
<b>Illustrations</b>	<b>ix</b>
<b>Tables</b>	<b>xiii</b>
<b>Glossary</b>	<b>xv</b>
<b>Acronyms and Abbreviations</b>	<b>xviii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 DO-178B Objectives Met by Using HiLiTE	2
1.2 Installation and Processing Requirements	2
1.3 Getting More Help and Reporting Problems	2
1.4 Document Conventions	3
1.5 References	3
<b>2 Getting Started with HiLiTE</b>	<b>5</b>
2.1 Using the HiLiTE Tutorial	5
2.2 Running HiLiTE	5
To use HiLiTE Shell:	5
To run HiLiTE from a Windows command prompt:	6
2.3 HiLiTE Input and Output	6
Required input:	6
Optional input:	7
Output files:	7
HiLiTE configuration data:	7
2.4 Generating Tests for a Specific Test Harness	8
<b>3 A Brief HiLiTE Tutorial</b>	<b>9</b>
3.1 How HiLiTE Works	10
3.2 Step 1: Create a Model	10
3.3 Step 2: Create a HiLiTE Command File	11
3.4 Step 3: Create a Range File	11
3.5 Step 4: Run HiLiTE to Generate Tests	12
Advanced Users	15
3.6 Step 5: Examine the HiLiTE Output	15
3.6.1 Examine the Status Report	16
3.6.2 Examine the Test Vector Report and Test Vectors	18
3.6.3 Examine the XML file of HiLiTE execution results	21
<b>4 Creating HiLiTE Command Files</b>	<b>23</b>
4.1 Command File Format	23
Reading a command file	24
Path formatting in examples	24
4.2 Basic HiLiTE Command-File Elements	25

4.3	Advanced HiLiTE Command-File Elements and Options .....	30
4.3.1	Command OptionSets and Options .....	30
	Default Values .....	30
	Multiple Instances of an OptionSet.....	30
4.3.2	Attribute Specification in Command-File Elements .....	31
4.3.3	Command-File Elements to Control Test Generation and Test Vector Output Location .....	31
4.3.4	Miscellaneous Advanced HiLiTE Command-File Elements .....	48
<b>5</b>	<b>Generating Test Cases.....</b>	<b>51</b>
5.1	HiLiTE Command File Processing .....	52
5.2	Test Generation Command Files .....	53
5.3	Specifying Test Vector Output Formats .....	54
5.4	Generating Tests .....	55
5.4.1	Generated Tests and Test Reports.....	57
5.4.2	Browsing Test Reports– Test Case Templates and Generated Vectors .....	60
5.4.3	Using Observation Points for Measuring Expected Output .....	61
	Specifying use of observation points .....	62
	Specifying port variable names.....	62
5.5	Examples of Test Generation.....	63
5.5.1	Validated Localizer .....	63
5.5.2	Memory Switch Patterns.....	67
5.5.3	State Flow Charts.....	68
5.6	Specifying Tolerance in Measurement of Expected Output Values .....	69
5.6.1	Default Tolerances used in Expected Output Measurement .....	69
5.6.2	Need for Specifying a Higher Tolerance than Default.....	69
5.6.3	Guidelines for Specifying Tolerance to Override The Default.....	71
5.6.4	Floating Point Precision Issues for Specific Blocks .....	71
	Test vector failures for blocks convFloatToU32 and convU32ToFloat due to precision.....	71
<b>6</b>	<b>Generating Stateflow Test Cases.....</b>	<b>73</b>
6.1	Stateflow Test Case Categories .....	73
6.1.1	State Reachability .....	74
6.1.2	Transition Reachability.....	74
6.1.3	Action Coverage .....	74
6.1.4	Condition Coverage .....	74
6.1.5	HiLiTE Limitations for Stateflow Testing.....	75
6.2	Testing a Stateflow Chart as an “Isolated Block” within the Model .....	75
6.2.1	Script to Isolate a Large Stateflow Chart into a Separate Model.....	77
	Using the script .....	77
	Error handling in isolateStateflow.m script .....	77
6.3	User Supplied Test Vectors (USV) to Complete Test Generation.....	78
	Steps to creating a USV .....	78
6.3.1	Debug Vectors .....	79
6.3.2	Writer Report .....	79

6.4	Using the StateflowOptions OptionSet.....	81
6.5	Using the Stateflow Example.....	83
6.5.1	The RiggedUnrigged Stateflow Model.....	84
6.5.2	Generating Test Cases for the RiggedUnrigged model.....	85
<b>7</b>	<b>Improving Test Generation Coverage Using Additional Observation and Stimulation Points..</b>	<b>91</b>
7.1	Using Secondary Stimuli .....	91
7.1.1	Definitions.....	92
7.1.2	Approach.....	92
7.1.3	Secondary Stimulus Usage Examples.....	92
7.2	Using HiLiTE's Suggestion Capability .....	93
7.2.1	Examining HiLiTE's Suggestions .....	94
7.2.2	Experimenting with Placement of Observation and Stimulation Points .....	95
	SignalBoundaries OptionSet Summary .....	96
<b>8</b>	<b>Using and Interpreting HiLiTE Output for Status and Errors .....</b>	<b>97</b>
8.1	HiLiTE Log in Console and Log File.....	97
8.1.1	User Review and Disposition of Errors and Warnings in HiLiTE Log .....	99
	Log Error Messages that Require User Review and Action .....	99
	Spurious/Redundant Log Error Messages that should be Ignored by User .....	100
8.1.2	Control of Logging by HiLiTE.....	101
	Example Modifications of Logger Levels .....	101
8.2	Model Summary Report.....	102
8.2.1	Model Summary Elements.....	102
8.2.1.1	SimulinkTestGeneration.....	103
8.2.1.2	StateflowTestCreation.....	103
8.2.1.3	Model Defects .....	104
8.3	Model Status Report .....	105
8.3.1	Status Report: Header .....	106
8.3.2	Status Report: User Specification of Data Types and Normal Ranges .....	106
8.3.3	Status Report: Errors in Propagation of Data Type and Ranges Thru the Model.....	107
	Spurious Error Messages for Propagation of Data Types and Ranges .....	108
8.3.4	Status Report: Test Generation Summary.....	108
8.3.5	Status Report: Stateflow Charts Test Generation Summary .....	108
	Stateflow Chart Requirements Coverage.....	108
8.3.6	Status Report: Test Generation for Block Requirements.....	109
	Potential Testability Limitations.....	109
8.3.6.1	Block Requirements Categories .....	110
8.3.6.2	Block Requirements Coverage Reporting.....	111
8.3.7	Status Report: Robustness Coverage of Model Inputs.....	112
8.3.8	Status Report: Stateflow Errors.....	112
<b>9</b>	<b>Model Analysis.....</b>	<b>115</b>
9.1	Creating HiLiTE Command Files for Model Analysis.....	115
	Successful completion of Model Analysis.....	116

9.2	Data Type and Range Propagation Analysis.....	117
	HiLiTE's Usage and Reporting of Range Propagation Results .....	117
9.2.1	User Assistance needed for HiLiTE Data Type and Range Propagation.....	118
	Specifying Feedback Loop Data Type.....	118
	Using the ExtensiveModelAnalysis Option.....	119
	Specifying Precise Ranges at the Outputs of StateFlow Charts.....	119
9.2.2	Special, Optional Range Checks .....	119
9.3	Detection of Model Design and Testability Defects .....	120
9.3.1	Categories of Model Defects.....	120
9.3.2	Types of Model Defects Currently Reported by HiLiTE.....	121
	9.3.2.1 Checks for Violations of Design Review Guidelines.....	123
9.3.3	Model Defect Reporting and Dispositions .....	125
	HiLiTE Reporting of Model Testability Defects .....	125
	User Dispositions of Defects and Potential Over Reporting of Defects by HiLiTE.....	125
9.3.4	Model Defect Examples.....	125
9.4	Generating Output Range Files.....	126
<b>10</b>	<b>HiLiTE Template Manager .....</b>	<b>128</b>
	Starting the Template Manager.....	128
	Closing the Template Manager .....	128
10.1	Viewing/Editing Information in the Template Manager.....	129
	What is a Block Type?.....	129
10.1.1	Viewing a template for a Block Type:.....	129
10.1.2	Viewing Block Type Details .....	130
	About UniversalDataType Specification .....	134
10.1.3	Viewing Block Requirements.....	134
10.1.4	Viewing Template Details .....	135
10.1.5	Viewing Test Case Details.....	136
10.1.6	Viewing Test Case Steps .....	138
10.1.7	Mapping Test Cases to Block Requirements .....	140
10.2	Adding and Changing Block Requirements and Templates .....	141
10.2.1	Adding/Changing Block Requirements .....	141
10.2.2	Adding/Changing Templates .....	142
10.2.3	Supporting Alternate Template Families .....	144
	Running HiLiTE with a particular template family:.....	144
	Capturing block requirements specific to a template family: .....	144
10.2.4	Creating Groups and Conditions.....	145
10.2.5	Modifying Templates.....	145
10.2.6	Adding Test Cases .....	146
10.2.7	Deleting Test Cases .....	146
10.2.8	Changing Test Cases.....	146
10.2.9	Adding Steps to Test Cases .....	147
10.2.10	Deleting Steps in a Test Case .....	148
10.2.11	Reordering Steps in a Test Case .....	148

10.2.12 Changing Group Relations (assigning test cases to groups).....	148
10.3 Tips for Creating Test Cases.....	149
10.4 Application Menus.....	149
10.4.1 File Menu.....	149
Open and Save As Dialog Box .....	149
Export and Import Dialog Box .....	150
10.4.2 Edit Menu .....	151
Settings .....	151
Revisions .....	151
10.4.3 Tools Menu.....	152
10.4.4 Help Menu .....	152
10.5 Managing Configuration of Changes to Blocks and Templates .....	152
10.5.1 Using Modified Blocks and Templates Data when running HiLiTE.....	153
10.5.2 Reviewing User-Modified Block and Template Data .....	153
10.6 Template Formula Language .....	154
10.6.1 Infix Operators.....	154
10.6.2 Predefined Constants .....	155
10.6.3 Math and Logic Functions .....	155
10.6.4 Keyword References.....	156
Port Objects and Values.....	156
Block Object and Values .....	159
Model Object and Values.....	160
10.6.5 Test Case Fields Reference.....	160
10.6.6 Definitions and Explanations of Specific Constructs .....	161
10.6.6.1 MDVD.....	161
<b>11 Using the HiLiTE Test Harness .....</b>	<b>162</b>
Reference Example.....	163
11.1 Generating Model Code.....	163
11.2 Generating Test Vectors and Harness Interface Code .....	164
11.3 Compiling and Linking Model and Harness Code.....	164
11.3.1 Compiling Harness Code.....	164
11.3.1.1 A makeinclude File for Target-Platform-Specific Definitions.....	165
11.3.2 Compiling Model Code .....	165
11.3.2.1 Separating Model Code from Harness Code.....	166
11.3.3 Linking all Object Code to Create Test Harness Executable.....	166
11.3.4 Creating Model-Specific Makefiles for the Compile and Link Steps.....	166
11.4 Running the Test Harness Executable .....	166
Usage .....	167
Options .....	167
11.5 Tolerance on Measurement of Output Values .....	168
11.5.1 Overriding the Built-In Default Tolerance .....	168
<b>12 Using the TPI Converter.....</b>	<b>170</b>
Files required to use the tool.....	170

Starting the TPI Converter .....	171
Closing the TPI Converter .....	171
12.1 Inputs .....	171
TPI generated with Test Points ON Option .....	171
TPI generated with Test Points OFF Option.....	171
CSV file	171
12.2 OUTPUT.....	171
Final TPI	171
Log.csv	172
12.3 Using the TPI Converter .....	172
12.3.1 Command Line Usage .....	172
12.3.2 Using the TPI Converter Interface.....	172
Creating TPI Files.....	173
Field descriptions .....	173
12.3.3 Example .....	175
Tolerance:	182
12.4 INI file.....	182
<b>Appendix A. Troubleshooting.....</b>	<b>186</b>
Incorrect or Missing Parameter Elements and Attributes .....	186
When Starting a Command file, HiLiTE Installs.....	188
Running Multiple Instances of HiLiTE/Appender Error .....	189
HiLiTE.log is Missing.....	189
Duplicate State Machine and Model Names .....	189
HiLiTE cannot start MATLAB.....	189
Can't Access MATLABEngine.dll .....	190
If dependencies are missing, you may not have installed all the HiLiTE prerequisites. ....	190
Initialization Errors .....	190
Problems after installing a new version of MATLAB .....	191
Missing Libraries .....	191
Garbled output or MATLAB crashes.....	191
FAQ	191
<b>Appendix B. Creating Range and Data Type Files.....</b>	<b>194</b>
Data Types .....	194
Creating Range and Data Type Files .....	195
Naming Variables in Symbol Files.....	199
Including Unique Identifiers in the Range File.....	200
<b>Appendix C. Reporting Problems and Enhancement Requests .....</b>	<b>202</b>
<b>Appendix D. Test Generation Objectives .....</b>	<b>204</b>
Automated Model-Based Test Generation.....	206
HiLiTE Test Generation.....	206
DO-178B Objectives Satisfied using HiLiTE.....	207



<b>Appendix E. HiLiTE Support for Block Libraries and MATLAB Versions .....</b>	<b>208</b>
<b>Appendix F. Generating Test Vector Output for Component Test Platform (CTP) .....</b>	<b>218</b>
Correctness Criteria.....	218
OptionSet Elements for Output Generation to CTP.....	219
Anchor Tracing in TPI files .....	226
<b>Appendix G. Block Attributes Lists.....</b>	<b>228</b>
<b>Appendix H. Command File Elements for Diagnostics and Development.....</b>	<b>234</b>
<b>Appendix I. Blocks with HiLiTE Built-In (Dynamic) Templates.....</b>	<b>240</b>
Look-Up Tables .....	242
1-D Look-Up Table.....	242
Data Point test cases: .....	242
Interpolation test cases:.....	242
Beyond Boundary Test Cases .....	243
Clipping test cases: .....	243
Extrapolation test cases:.....	244
Look-Up Table (2-D) .....	245
Data Point test cases: .....	245
Interpolation test cases:.....	245
Beyond Boundary Test Cases .....	245
Clipping test cases: .....	245
Extrapolation test cases:.....	247
Test Case Templates .....	247
Look-Up Table (3-D) .....	249
Data Point test cases: .....	249
Interpolation test cases:.....	249
Beyond Boundary Test Cases .....	249
Clipping test cases: .....	249
Extrapolation test cases:.....	251
Look-Up Table Pre-Index .....	253
Data Point test cases: .....	253
Interpolation test cases:.....	253
Beyond Boundary Test Cases .....	253
Clipping test cases: .....	253
Test Case Templates .....	253
Look-Up Table N-D (HW)–1D.....	254
Data Point test cases: .....	255
Test Case Templates: .....	255
Look-Up Table N-D (HW)-2D .....	255
Data Point test cases: .....	256
Look-Up Table N-D (HW)-3D .....	256
Vector Selector (HW) .....	257
Test Case Templates .....	257
Matrix Selector (HW) .....	258
Enable Subsystem (Nested Model) .....	261

Initial Condition .....	261
Reset/Held .....	261
Tests for Block: Subsys_Enabled .....	262
Test Case Templates .....	262
Inport (HW) .....	263
Bus Selector (HW) .....	264
Selecting signals from input.....	264
Generated Test cases:.....	264
Test Point (HW) .....	265
Generated Test cases for Boolean Test Points .....	265
Generated Test cases for numeric Test Points .....	266
Stimulation Blocks .....	267
Stimulate Boolean (HW) .....	267
Generated test cases .....	267
Stimulate Numeric (HW).....	268
Generated test cases .....	269
Vector Assignment (HW) .....	269
Generated Test Cases.....	271
<b>Appendix J. Generating Test Vector Output for VectorCast Platform.....</b>	<b>272</b>
Testcase Specific files .....	273
CSCI Specific Files.....	273
OptionSet Elements for Output Generation to HTVC .....	273
<b>INDEX .....</b>	<b>276</b>

## Illustrations

Figure 1 - Command Prompt window with manually entered HiLiTE command.....	6
Figure 2 - Simplified view of HiLiTE processing .....	10
Figure 3 - Simulink model StrangeSum.mdl .....	10
Figure 4 - HiLiTE command file for model StrangeSum.mdl .....	11
Figure 5 - Range file for model StrangeSum.mdl .....	12
Figure 6 - HiLiTE Shell Test Selection page.....	13
Figure 7 - HiLiTE Shell Output page and StrangeSum MATLAB model.....	14
Figure 8 - Completed HiLiTE Console Log for StrangeSum command file .....	14
Figure 9 - HiLiTE command execution .....	15
Figure 10 - HiLiTE-generated output files for model StrangeSum.mdl .....	16
Figure 11 - StrangeSum test status report .....	17
Figure 12 - StrangeSum test report (partial) .....	18
Figure 13 - Contents of StrangeSum_Vectors.csv .....	20
Figure 14 - Contents of StrangeSum_Summary.xml .....	21
Figure 15 - Command file to generate testcases .....	24
Figure 16 - Option sets and option key-value pairs in <Command> and <Model> elements.....	30
Figure 17 - How HiLiTE generates test cases.....	52
Figure 18 - Simple test generation command file .....	53
Figure 19 - Test generation command file with column specification.....	53
Figure 20 - Test generation command file specifying test case format .....	54
Figure 21 - Test generation command file .....	55
Figure 22 - Command execution information for BasicMath.hilite test generation .....	55
Figure 23 - Simulink model for BasicMath .....	56
Figure 24 - Input details showing Data Type and Normal Operating Range.....	56
Figure 25 - Test report for BasicMath .....	58
Figure 26 - Status report for BasicMath.mdl .....	59
Figure 27 - Template and test case for min block.....	60
Figure 28 - Legend in the <i>model_Report.htm</i> .....	61
Figure 29 - Forward propagation difficulty with time-dependent blocks in series .....	61
Figure 30 - Validated localizer model.....	64
Figure 31 - Template and test vectors for lag filter block.....	65
Figure 32 - Template and test cases for block rawValidSwitch1_Memory .....	66

Figure 33 - Model including switch delay .....	67
Figure 34 - Test report for memory switch .....	68
Figure 35 - Example model with series of blocks introducing high numerical error.....	70
Figure 36 - CSV file input to HiLiTE to specify the tolerance for an output measurement .....	70
Figure 37 - “Conv Float to U32 (HW)” and “Conv U32 to Float (HW)” blocks simulated in MATLAB by giving constant input giving same outputs for different inputs.....	71
Figure 38 - An example of test case failure for after block”AfterconvFloatToU32” .....	72
Figure 39 - Portion of Status report showing reachability for Stateflow test cases .....	73
Figure 40 - An example Stateflow model to be tested in isolation .....	76
Figure 41 - Partial view of the isolated TPI file showing the inputs and outputs .....	76
Figure 42 - The isolated model created by the isolateStateflow.m script .....	77
Figure 43 - Difficult scenario for the Script; it will not be able to resolve Input port 1 of the chart. ....	78
Figure 44 - The error reported by the script in MATLAB command prompt indicating the port number of the chart.....	78
Figure 45 - incomplete model created by the script with input unattached .....	78
Figure 46 - An example Stateflow model in which HiLiTE cannot fire the transition from B to C.....	80
Figure 47 - Hand-modified vector so HiLiTE can reach state C (and progress to D).....	80
Figure 48 - RiggedUnrigged model for Stateflow example.....	84
Figure 49 - RiggedUnrigged Stateflow example model path in HiLiTE installation directory .....	85
Figure 50 - RiggedUnrigged model execution complete status .....	85
Figure 51 - Location of results for RiggedUnrigged model.....	86
Figure 52 - Partial view of RiggedUnrigged_StatusReport.html.....	87
Figure 53 - Partial view of test generation report RiggedUnrigged_Report.htm.....	88
Figure 54 - Input and output port values.....	89
Figure 55 - Generated test vectors .....	89
Figure 56 - Time dependent blocks in series with limits on input – difficulty in model-level test generation.....	91
Figure 57 - Internal Construction of Stimulate Numeric Block.....	93
Figure 58 - Use of Stimulation Block to Ease Test Generation of Downstream Block leadLag.....	93
Figure 59 - HiLiTE Shell Output page showing successful test generation .....	98
Figure 60 - Example command window showing errors .....	99
Figure 61 - Model summary for BasicMath example .....	102
Figure 62 - Model summary for RiggedUnRigged.mdl showing StateflowTestGen status .....	104
Figure 63 - Partial view of Summary for a model with design defects at Simulink Level .....	105
Figure 64 - Status report header.....	106
Figure 65 - Status report all data types and normal ranges specified.....	107

Figure 66 - Status report example: missing data types and normal ranges and error with range computation for Stateflow chart output ports .....	107
Figure 67 - Status report example: spurious range propagation error should be ignored .....	108
Figure 68 - Status report example: Test Generation Summary .....	108
Figure 69 - Status report example: Stateflow charts test generation summary .....	109
Figure 70 - Status Report: Test Generation for Blocks' Requirements - Layout .....	109
Figure 71- Status Report: Coverage Reporting of Different Types of Block Requirements .....	111
Figure 72 - Status Report: where no requirements specified are reported .....	112
Figure 73 - Status Report: Reporting of Robustness Coverage of Model Inputs.....	112
Figure 74 - Example Stateflow chart .....	113
Figure 75 - Portion of writer report.....	113
Figure 76 - HiLiTE command file for model analysis.....	116
Figure 77 - RangePropExample.mdl.....	116
Figure 78. Output Range File for the example Model .....	117
Figure 79. Indication of overflow upstream in the model from a Boolean output.....	118
Figure 80. Overflow error indication in the model status report.....	118
Figure 81 - Simulink Range Analysis Example: Detecting Divide-by-Zero Possibility .....	125
Figure 82 - Example of Untestable Condition (mutually exclusive conditions).....	126
Figure 83 - Template Manager Main Window .....	129
Figure 84 - Block Details window for Block Type TimerFixed.....	131
Figure 85 - Block Details Window with <i>More</i> Information Displayed .....	131
Figure 86. Unverisal DataType Sets Hierarchy .....	134
Figure 87 - Requirements Table within Block Details Window.....	135
Figure 88 - Manage Template Dialog.....	135
Figure 89 - Test Case Details Dialog.....	137
Figure 90 - Selected Test Case Details and Step Table .....	138
Figure 91 - Selected Test Case Details with min and max values for Inputs.....	138
Figure 92 - Input dialog box .....	139
Figure 93 - Step Details Dialog.....	140
Figure 94 - Mapping a test case to requirements .....	141
Figure 95 - Checkboxes to make Template Manager read only .....	141
Figure 96 - Editing block test requirements.....	142
Figure 97 - Template Details Dialog.....	143
Figure 98 - Templates drop-down.....	144
Figure 99 - Selecting TemplateFamily test case requirement.....	144

Figure 100 - New Test Case Window .....	146
Figure 101 - Dialog Box for Open and Save As .....	150
Figure 102 - Dialog Box for Export and Import .....	150
Figure 103 - Settings dialog box .....	151
Figure 104 - Template Family Revisions dialog .....	152
Figure 105 - User specification of blocks and templates data in HiLiTE Command File .....	153
Figure 106 - Model Status Report showing use of User-Specified Blocks and Templates Files.....	153
Figure 107 - HiLiTE Test Harness usage and flow of code files.....	162
Figure 108 - Specifying tolerance in test vector file (Modelname_Vectors.csv).....	168
Figure 109 - GUI for TPI Converter .....	173
Figure 110 - TPI Files Tab.....	175
Figure 111 - Simulink Model.....	176
Figure 112 - Test Points in the generated code. ....	176
Figure 113 - Basic Range File Containing Only Normal Operating Ranges .....	197
Figure 114 - Example File Containing Range and Data Type Information .....	198
Figure 115. Examples of Vector Ranges .....	199
Figure 116 - Honeywell Aerospace MBD Process for DO-178B using HAM & HiLiTE Tools .....	205
Figure 117 - DO-178B Based Process .....	205
Figure 118 - Example Command Element Containing CTP and Source OptionSets .....	220
Figure 119 - Anchors specified in the Revision block of the model.....	226
Figure 120 - All Anchors specified in revision block appear in AT5 Source in TPI file.....	226
Figure 121 - Anchor appearing in "Requirements" line for the blocks in the TPI file for which supported anchors are specified in Revision block.....	227
Figure 122 - MATLAB Model to VectorCAST executable Test Case Conversion Process .....	273

## Tables

Table 1. Basic HiLiTE command file elements .....	25
Table 2. Advanced HiLiTE command file elements for test generation and output location.....	32
Table 3. Miscellaneous advanced HiLiTE command file elements.....	48
Table 4. Fields in Block Details.....	132
Table 5. Fields of Port Type .....	133
Table 6. Fields in Manage Template.....	135
Table 7. Fields in Test Case Details.....	137
Table 8. Selected Test Case Fields and Step Table.....	139
Table 9. Fields in Step Details .....	140
Table 10. Math and Logic Functions .....	155
Table 11. Port Objects and Values Keywords .....	158
Table 12. Block Objects and Values Keywords.....	159
Table 13. Keys and values available for the <i>HTH</i> OptionSet .....	164
Table 14. Matching of test points in TPI_ON and TPI_OFF files.....	177
Table 15. An excerpt from the CSV files containing only model outputs and model level test points ....	178
Table 16. An excerpt from the CSV file containing the moved columns .....	178
Table 17. Configuration File Errors Reported in the Command Window and HiLiTE.log.....	186
Table 18. HAM Library Blocks Supported by HiLiTE .....	209
Table 19. CCC Library Blocks Supported by HiLiTE.....	212
Table 20. Boeing Library Blocks Supported by HiLiTE .....	213
Table 21. Key Values Available for the General OptionSet.....	220
Table 22. Keys and Values Available for CTP OptionSet.....	221
Table 23. Keys and Values Available for the Source OptionSet .....	225
Table 24. Keys and Values Available for UseExplicitAT3Anchors OptionSet .....	225
Table 25. HAM block attributes .....	229
Table 26. Boeing block attributes .....	230
Table 27. CCC block attributes.....	232
Table 28. Common block attributes.....	233
Table 29. Command file elements used for development and diagnostics .....	235
Table 30. HAM blocks with HiLiTE Built-In (dynamic) templates.....	241
Table 31. Dynamically-generated test cases for Look-Up Table 1D-Clip, fully interpolated .....	243
Table 32. Dynamically-generated test cases for Look-Up Table 1D-Extrapolate, fully interpolated.....	244

Table 33. Dynamically generated test cases for Look-Up Table 2-D (HW) with clipping, fully interpolated .....	246
Table 34. Dynamically generated test cases for Look-Up Table 2-D (HW) with extrapoliation, fully interpolated .....	248
Table 35. Dynamically generated test cases for Look-Up Table 3D (HW), with clipping, fully interpolated .....	250
Table 36. Dynamically generated test cases for Look-Up Table 3D-Extrapolate .....	252
Table 37. Dynamically generated test cases for Look-Up Table preIndex1 .....	254
Table 38. Dynamically generated test cases for Look-Up Table N-D (HW)-1D .....	255
Table 39. Dynamically generated test cases for Look-Up Table N-D (HW)-2D .....	256
Table 40. Dynamically generated test cases for Vector Selector (HW).....	258
Table 41. Dynamically generated test cases for Matrix Selector (HW).....	260
Table 42. Dynamically generated test cases for Enable Subsystem (Nested Model) .....	262
Table 43. Dynamically generated test cases for Inport (HW).....	263
Table 44. Dynamically generated test cases for a Bus Selector block.....	265
Table 45. Dynamically generated test cases for a Test Point Boolean block.....	266
Table 46. Dynamically generated test cases for a Test Point Numeric block.....	267
Table 47. Dynamically generated test cases for a Stimulate Boolean block.....	268
Table 48. Dynamically generated test cases for a Stimulate Numeric Block .....	269
Table 49. Dynamically generated test cases for Vector Assignment (HW) block.....	271
Table 50. Key Values Available for the HTVC OptionSet.....	274



# Glossary

**Backward propagation**

Backward projection of a subject block's inputs through prior blocks. The process of setting up input(s) to a prior block in such a way as to produce a desired value at the subject block's input.

**Block instance**

A specific occurrence of a library block in a Simulink® model. A model can include multiple instances of a library block.

**Block requirements**

Low-level requirements for each function block to be supported by the *test case template*.

**Block type**

*see* library block.

**Chart**

*see* Stateflow chart.

**Command file**

Instructions provided to HiLiTE-TG when it is run from the command line. It contains the name of the MATLAB Simulink model, the symbol data-type/range files, and the name of the directory where the HiLiTE-TG output files are placed.

**Forward propagation**

Forward projection of a subject block's output through a subsequent block. The process of setting up input(s) to a subsequent block in such a way as to correctly determine the value of a subject block's output by measuring the output of a subsequent block.

**Generated test case**

A time sequence of model input values to be injected and model output values expected that exercise a specific *Test Case Template* for a specific block instance in a specific model. For each block instance in a model, HiLiTE-TG will generate a set of *test cases* corresponding to the *test case templates* associated with that block's type. *See also* Test case.

**HiLiTE-TG supported library block**

A library block for which HiLiTE-TG can generate tests for instances of that block type and for blocks adjacent to them in a model. These also include some non functional blocks (e.g., sub system, bus selector bus creator, mux, demux etc.) for which HiLiTE may not need to generate any test case, but must account for its connectivity functionality. The block types that can be used in a model are defined in the Honeywell Autocode Manager (HAM) block library and other application-domain specific block libraries such as the Common Commercial Controller (CCC) block library. All HiLiTE-TG supported library blocks are identified in the appropriate section of this document.

**Library block**

A Simulink standard library block or a library block (masked subsystem) created by a user. A library block provides specific functionality supported by MATLAB®/Simulink®. Also called a block type.

**Low-Level Software Requirements (LLSR)**

The low-level requirements represented by the contents of a model. These are typically the functional requirements of the blocks in a model, which are specified in a formal requirements document.

**Low-Level Test Requirements (LLTR)**

The test requirements captured in test cases that correspond to the testing of the LLSRs. All the LLTR's of a block taken together must verify all the LLSR's of that block. The reason LLTR's are

distinguished from LLSR's is that the LLTR's are for testing purposes only and do not represent a specification of the requirements of models or blocks.

### **MATLAB® Simulink®/Stateflow model**

A block diagram built in the MATLAB Simulink/Stateflow tool using one or more Library Blocks and/or Stateflow charts.

### **Operating range**

Each signal in the model has a “normal operating range” that denotes the typical range of values of that signal during normal operation. The normal range is significant only for numeric signals; for booleans it is always [0, 1]. The normal range comprises a minimum and a maximum value. For model inputs, the user specifies the normal range to HiLiTE either in the model or in a separate range file. For all intermediate signals and model outputs, the normal range is computed by HiLiTE using range propagation.

### **Signal name**

The label applied to block output values. For model outputs that are implied “test points” at the output of blocks (as in HAM blocks), the signal name uses the format "*blockname\_TP*". For intermediate signals (displayed without any special background color), the signal name uses the format "*blockname.OI*", where the rightmost digit indicates the index of the block's output port that this signal denotes.

### **Stateflow chart**

A top-level entity in the Stateflow part of the model that denotes a state machine consisting of inputs, outputs, and states. A Simulink/Stateflow model can contain one or more *chart*. The following terms describe Stateflow charts.

**State:** A state entity within a chart that describes some operational mode. A state describes a mode of an event-driven system.

**Children:** The set of states that are contained hierarchically inside another state.

**Initial node:** The starting point of execution for a state. When a state that has children becomes active, the transitions from the initial node are evaluated and exactly one is taken.

**Transition:** A Stateflow transition that begins either at a state or an Initial Node and ends at a State. A transition describes the conditions that support and actions that occur when a system changes operational modes.

### **Symbols**

Variable identifiers for inputs, outputs, states, and named constants in a MATLAB model.

### **Symbol file**

ASCII-formatted files for implementing symbol variables in the model's code. These files describe data types (e.g. int32, bool, uint16, float32) and may also contain range information for normal operating range and maximum allowable range values for certain symbols. HiLiTE-TG can read multiple symbol files in various ASCII text formats.

### **Test case**

A time sequence test vectors that exercise a specific requirement in a model. For a Simulink model, a generated test case corresponds to a *Test Case Template* for a specific block instance in a model.

### **Test case template**

A description of desired behavior for a library block given as a time sequence of inputs and expected outputs. Note that test case templates may include hard-coded values or formulae and may involve multiple steps over time. Test templates also cover both normal use and robustness requirements. The HiLiTE.mdb (Access database) includes test case templates for each HiLiTE-TG supported block

type; these templates represent the reviewed requirements for normal range and robustness. You can edit these templates and add new ones as needed.

**Test vector**

A set of signal values to be applied as model inputs and corresponding output values expected as model outputs. A test vector is displayed as a row in a table of the test report that HiLiTE generates for each successful run. Each test vector can have one or more time steps. The indicated model input values must be applied for the given number of time steps and the model output expected value must be measured at the end of the last time step in the vector.

**Time step**

A time step is one step (frame) of execution of the model. For a model with a rate of 10 Hz, there are 10 time steps per second, each of .1 second duration.

**Test Generation Status Report**

This report is generated by HiLiTE–TG that displays the list of blocks in the model and identifies the low-level requirements of the block for which a test case could not be generated by HiLiTE–TG. It also displays the Requirement % coverage achieved for both Simulink and Statechart requirements, lists the test generation directives (includes Stateflow Test Generation options used) which were used in the HiLiTE command file, and displays any errors with respect to user specification of data types and ranges.

**Test Generation Summary & Guidance Reports**

These are reports in XML and HTML format intended solely for the guidance of the user in using HiLiTE (not used as part of any DO-178B related activity or process).

## Acronyms and Abbreviations

<b>ANSI</b>	American National Standards Institute, Inc.
<b>CCC</b>	Common Commercial Controller
<b>CM</b>	configuration management
<b>HAM</b>	Honeywell Autocode Manager
<b>HiLiTE</b>	Honeywell Integrated Lifecycle Tools and Environment
<b>HiLiTE-TAS</b>	HiLiTE Test Automation Suite (verification tool)
<b>HiLiTE-TG</b>	HiLiTE Test Generator (a component of HiLiTE-TAS)
<b>HiLiTE-TH</b>	HiLiTE Test Harness (a component of HiLiTE-TAS)
<b>LLSR</b>	Low-Level Software Requirements
<b>LLTR</b>	Low-Level Test Requirements (corresponding to LLSRs)
<b>MC/DC</b>	Modified Condition/Decision Coverage
<b>MDVD</b>	Minimum Distinct Value Difference
<b>PSAC</b>	Plan for Software Aspects of Certification
<b>RTW/EC</b>	MATLAB Real-Time Workshop Embedded Coder
<b>SUT</b>	software under test
<b>SCM</b>	Software Configuration Management
<b>TOR</b>	Tool Operational Requirements
<b>TQP</b>	Tool Qualification Plan
<b>TVR</b>	Tool Verification Records

# 1 Introduction

This document describes the version of HiLiTE Test Automation Suite for use with MATLAB® Simulink®/Stateflow® models and Honeywell Autocode Manager (HAM).

The Honeywell Integrated Lifecycle Tools and Environment–Test Automation Suite (HiLiTE–TAS) is a verification tool that generates test cases from a design model and verifies that the model’s executable object code conforms to the high- and low-level requirements embedded in the model. HiLiTE–TAS is used independently from the source code generation and source code review tools of the MATLAB Simulink/Stateflow modeling environment.

HiLiTE-TAS imports design diagrams (as MATLAB® Simulink/Stateflow® models), generates test cases against the low-level requirements specified in the design diagram, and applies and verifies the test cases on the executable object code that implements the model. HiLiTE–TAS consists of two components: HiLiTE Test Generator (HiLiTE-TG) and HiLiTE Test Harness (HiLiTE-TH).

HiLiTE-TG imports models (design diagrams) from MATLAB Simulink/Stateflow and generates test cases based on the functional requirements of the function blocks in the model and their interconnection. The output of HiLiTE-TG includes test cases in two different file formats that test low-level requirements. A test harness can apply these test cases to the executable object code of the model and compare the expected output values in the test cases to actual output values observed from the object code as pass-fail criteria for each test case. HiLiTE–TG supports the use of two test harnesses that can be used independently from each other.

HiLiTE-TH interfaces with the executable object code that implements the model. It applies model input values contained in the test cases to the object code and executes the model function for each time step. The model outputs for each time step are measured and compared to the expected output listed in the test case. HiLiTE-TH generates a report giving the pass/fail status of each test case (a test case fails if the measured output is significantly different from expected output).

### Use HiLiTE to:

- Catch problems that can escape manual reviews and simple analysis within the MATLAB® environment.
- Perform multiple-model analysis.
- Achieve structural coverage of object code.
- Verify that object code meets reviewed requirements embedded in MATLAB models.

### HiLiTE provides several benefits. It:

- Reduces software verification costs and time to meet DO-178B certification and DFSS/quality objectives (historically up to 30% savings in total cost).
- Improves model portability and code reusability.
- Reduces cost and time required for software and system integration.

## 1.1 DO-178B Objectives Met by Using HiLiTE

HiLiTE automates the manual effort of test case creation to satisfy DO-178B as outlined for verification objectives A-6.3, A-6.4, and A-7.4.

DO-178B Verification Objectives Covered by HiLiTE	
A-6.3	Executable Object code complies with low-level requirements. (HiLiTE-TH)
A-6.4	Executable Object code is robust with low-level requirements. (HiLiTE-TH)
A-7.4	Test coverage of low-level requirements is achieved. (HiLiTE-TH)

**Note:** For specific information on the usage context (MATLAB Simulink/Stateflow versions, HAM and other block libraries, test vector output formats) of a qualified version of HiLiTE and the DO-178B credit that can be claimed, please refer to the Tool Qualification Accomplishments Summary (TQAS) and the Tool Operational Requirements (TOR) of the specific, qualified version of HiLiTE being used.

## 1.2 Installation and Processing Requirements

For details about HiLiTE installation procedures, refer to the current *HiLiTE Installation Guide*. Following is a summary of HiLiTE processing requirements.

The HiLiTE tool is written in C# and J# using the Microsoft Windows .NET runtime environment. It is designed for use in conjunction with the MATLAB/Simulink tool, operating on a computer using Windows XP or Windows 2000 and Microsoft .NET framework version 2.0 or newer.

HiLiTE uses the following software:

- MATLAB Simulink, release 14 SP3 or newer
- Microsoft .NET redistributable, version 2.0 or newer
- Microsoft Visual J# redistributable, version 2.0
- Perl (to support HiLiTE Test Harness and scripts) available from <http://www.activestate.com/Products/ActivePerl/>

## 1.3 Getting More Help and Reporting Problems

If you experience any technical problems or have difficulty using HiLiTE, send an e-mail to: [HiLiTE.Support@honeywell.com](mailto:HiLiTE.Support@honeywell.com). Someone from the HiLiTE Support Team will contact you.

For reporting bugs and enhancement requests, please refer to *Appendix C: Reporting Problems and Enhancement Requests*.

## 1.4 Document Conventions

<element name>	Names enclosed in angle brackets are XML tags that define what you want HiLiTE to process.
change in type face	Code and command file examples are displayed in <b>Lucida Console</b> typeface. Within examples and lists, elements, attributes, and user response is emphasized with either <b>boldface</b> or <i>italic</i> type.
file paths	<b>%HILITE_HOME%</b> indicates the installation path for HiLiTE. (e.g. if you accepted the default installation path, %HILITE_HOME% is c:\program files\hilite)
• (dot or period)	indicates the current working directory. This convention is used in command files to indicate that HiLiTE will find the described item (model, template, etc.) in the same folder as the HiLiTE configuration file.

## 1.5 References

These documents will also be helpful for your understanding of HiLiTE and certain aspects of test generation:

- [1] *An Approach to Improving HiLiTE's Auto-Test Generation Using Secondary Stimuli and Observation Points*. White paper. Honeywell Laboratories Document No.: AES-09-003. July 31, 2009.
- [2] *B787 FCE Component Test Platform User Guide*, Doc no. 787-450430-0106, Revision ?, Honeywell International Inc.
- [3] D. Bhatt, S. Hickman, K. Schloegel, and D. Oglesby. "An Approach and Tool for Test Generation from Model-Based Functional Requirements," Proc. of 1st International Workshop on Aerospace Software Engineering, 2007.
- [4] D. Bhatt; Hall, B.; Dajani-Brown, S.; Hickman, S.; Paulitsch, M.; "Model-based development and the implications to design assurance and certification," Proceeding of the Digital Avionics Systems Conference, 2005, Volume 2, 30 Oct.-3 Nov. 2005 Pages:13 pp. Vol. 2. (Available at: <http://w3.cas.honeywell.com/swtech/mbd/hilite/documentation/mbd-paper-bhatt-dasc05-official.pdf> )
- [5] Flight Controls Electronics (FCE) Control Law Low-Level Function Definitions, Cage Code 81205, DWG No. S641Z001, Sheet 06, Rev B2, Boeing.
- [6] *HiLiTE Installation Guide*, supplied with HiLiTE Installation, Honeywell International Inc.
- [7] *HiLiTE Shell User Guide*, supplied with HiLiTE Installation, Honeywell International Inc.
- [8] *HiLiTE Test Vector File Format*, Honeywell International Inc., Honeywell Laboratories Doc. No. AES-R05-009, Version 1.5.
- [9] *Logic Symbol Definitions for the Common Commercial Controller*, Document No. 31-18014, August 29, 2008, Honeywell Aerospace, Honeywell International Inc.
- [10] *MATLAB, Simulink User Guide*, [MathWorks, Inc.]
- [11] Radio Technical Commission for Aeronautics, Inc. *Software Considerations in Airborne Systems and Equipment Certification*, 1992, RTCA/DO-178B.

- [12] *Software Approval Guidelines*, FAA Order No. 8110.49.
- [13] *Systems Requirements Specification for the Honeywell Autocode Manager Block Library*, Version 5, SPEC. NO. PS7027827-003, CAGE CODE 55939, REV A., Honeywell International Inc.
- [14] *Guidelines for Placement of Stimulation Blocks and Observation Points in MATLAB Simulink Models for use with HiLiTE*, Honeywell Laboratories Doc. No. AES-R09-002, Version 1.0.
- [15] *Tool Operational Requirements (TOR) for Honeywell Integrated Lifecycle Tools & Environment Test Automation Suite (HiLiTE-TAS) v7*. Honeywell Laboratories Doc. No. AES-R10-001, Revision 1.4.



## 2 Getting Started with HiLiTE

Once HiLiTE and other supporting programs are installed, obtain or create suitable input for HiLiTE, including processing instructions that are given in a command file. Basic input includes MATLAB Simulink or Stateflow models and the symbol files that identify their input and output. To create a command file that gives HiLiTE directions for generating code and test cases, follow instructions in Section 4.

### 2.1 Using the HiLiTE Tutorial

To see HiLiTE “in action” before reading the details, step through the tutorial in Section 3, A Brief HiLiTE Tutorial. The tutorial lets you try basic HiLiTE usage with example files that are included in the HiLiTE installation.

### 2.2 Running HiLiTE

The HiLiTE command file contains all instructions that identify whether HiLiTE should generate code or test cases and the data to be used in processing. To actually start HiLiTE, you run a simple command as described in the following steps. To learn how to create a command file, refer to the Section 4, Creating HiLiTE Command Files. For detailed information about HiLiTE output, refer to *Section 5, Generating Test Cases*.

HiLiTE is installed with several examples that you can run to see how it works and whose command files you can examine, or even use as templates for your own projects.

You can use the included HiLiTE Shell tool to select models or command files. When you are proficient with HiLiTE, you may want to run it from a Windows command prompt.

#### To use HiLiTE Shell:

HiLiTE Shell is a tool included in the HiLiTE installation to simplify usage. It can be used to create and execute simple HiLiTE command files for selected models or to execute existing command files. The tool is available from the **Start** menu (select **Programs→HiLiTE→HiLiTE Shell**). For detailed instructions,

refer to the *HiLiTE Shell User Guide*, also available from the Start menu (Programs→HiLiTE→Documentation→HiLiTE Shell User Guide).

### To run HiLiTE from a Windows command prompt:

1. Open a Command Prompt window (or the shell of your choosing).
2. Navigate to the directory that holds your command file (e.g. `cd \project files\Hi Li TE\`).
3. Enter the HiLiTE command qualified with the name of the command file you are using. E.g.:

```
cmd-prompt> hilite strangesum.hilite
```

HiLiTE displays processing information in the console log. If MATLAB is not already running, HiLiTE starts MATLAB and displays the model from which it will generate code or test procedures. Figure 1 is an example of invoking HiLiTE from command prompt window.

```

C:\HiLiTE\Examples>cd C:\HiLiTE\Examples\HiLiTE-TG-HAM\MathExamples
C:\HiLiTE\Examples\HiLiTE-TG-HAM\MathExamples>Hilite strangesum.hilite
HiLiTE Version 7.6.2.0 - Honeywell Confidential - Copyright 2005-2010 Honeywell International Inc.
This copyrighted work and all information and expression are the property of Honeywell International Inc., contain trade secrets and may not, in whole or in part, be licensed, used, duplicated, closed for any purpose without prior written permission of Honeywell International Inc. All rights reserved.
Note: The following log contains specific reports of errors in the inputs to HiLiTE (command file, model, range files).
The log also contains diagnostic information to help in future enhancements to HiLiTE; which should be ignored by the user.
Processing HiLiTE Command File: strangesum.hilite ...

##### Initialization of data types, default ranges and max allowed ranges:
All real data types in the model will be interpreted normally (32 or 64 bits); as per default
Target Data Type definitions used: HAMRTW; as per default
Maximum Allowed Range for 32-bit floats is r[-3.402823E+38:3.402823E+38]; as per default
Default Ranges for Data Types:
float - r[-1000:1000] as per default; double - r[-1000:1000] as per default; int - i[-100:100] as per default; unsigned int - i[0:100] as per default;
Reading ranges/data-types for model inputs and other signals from file: C:\HiLiTE\Examples\HiLiTE-TG-HAM\MathExamples\StrangeSum_Ranges.csv
Loading HiLiTE BlockTypes from HiLiTE Standard XML file: C:\HiLiTE\Data\HiliteBlocks.xml
Loading Test Case Templates from HiLiTE Standard XML file: C:\HiLiTE\Data\HiliteTemplates.xml
##### Starting Matlab please wait .....
MatlabEngine <DLL> version 5.1
Using MATLAB Version 7.1.0.246 (R14) Service Pack 3

```

Figure 1 - Command Prompt window with manually entered HiLiTE command

When HiLiTE console log indicates that processing is complete, you can close the command prompt window or run HiLiTE against another command file. Successful generation will create code or test cases and supporting files in the output directories specified in the command file.

HiLiTE also keeps a copy of the console log in a file named *HiLiTE.log* in the HiLiTE install directory (%HiLiTE\_HOME%\Build\bin\).

## 2.3 HiLiTE Input and Output

The following lists define the files that HiLiTE uses as input and the files it creates as output.

### Required input:

**HiLiTE Command File:** Instructions for HiLiTE to process. This is the file you select in HiLiTE Shell or name on the command line when you run HiLiTE. It instructs HiLiTE to generate test cases and contains the names of the models and supporting files, the attributes of the data that you want to process (for example, the columns from which to extract data), and the name of the output directory. Section 4, Creating Command Files includes examples of command files and detailed instructions for creating them. Typically, this file has a .hilite extension.

**MATLAB models:** Block diagrams built in MATLAB that include one or more block instances. To generate code you can use either Stateflow or Simulink models. For test generation, this version of HiLiTE can process only MATLAB/Simulink or Stateflow models. It generates test cases or code files for each block in the model (except model inputs and outputs). For help creating MATLAB models, refer to the MATLAB documentation and online help files.

### Optional input:

HAM-based models may already include range values for inputs, outputs, and constants. If a model does NOT include this information, you will need to include a *range file* that defines this information. Even when the model includes the information, you may want to use a range file to ease generating tests for different ranges.

**Range files** are ASCII files in which columns are delimited by commas, spaces, or tabs. Range files contain the values of normal operating ranges (opmin, opmax) with corresponding symbol names. A project may use several range files to cover all the symbols. To generate tests from a model, be sure you include references to all the range files needed to cover all the referenced symbols. For more information about creating range files, refer to Appendix B: Creating Range and Data Type Files.

Note that if you specify the normal range values for each input in the HAM Input Port block, you do not need to create a range file. Information in the range file will override ranges in the model; this may be desirable if you wish to test different ranges, as you can easily change the range file without disturbing the model.

### Output files:

**Test Vector Files:** HiLiTE generates test vectors to be used with your chosen test harness—.csv (*ModelName\_Vectors.csv*) format for HiLiTE Test Harness (HTH) or .tpi for Component Test Platform (CTP).

**Test VectorReport:** This is an HTML version of the vector file for you to study. The report is named for the model being tested (*ModelName\_Report.htm*). For more information about test generation output, refer to Section 5, Generating Test Cases.

**Model Analysis and Status Report:** Status reports (*ModelName\_StatusReport.htm*) describe any issues encountered in analyzing the model and list which test cases were generated or could not be generated for each function block in the model. Test cases cannot be generated for function blocks that are not supported or if HiLiTE-TG cannot determine values. You will need to manually create the cases that HiLiTE cannot generate. For more information about test generation output, refer to Section 8, Using and Interpreting HiLiTE Output. For information about model analysis, refer to Section 9, Model Analysis.

**HiLiTE Execution (Model Analysis and Test Generation) Summary:** The file *modelname\_Summary.xml* summarizes the HiLiTE results, showing the models used, success of the Simulink and Stateflow test generation, requirements coverage achieved, results of model analysis including lists of model design defects found by HiLiTE, and other error status.

**HiLiTE Log:** HiLiTE logs command execution status and certain errors encountered for each run in a console log and replicates the same information in a log file (HiLiTE.log) stored in the %HILITE\_HOME% \Build\bin directory. Please see Section 8.1 for more information on the HiLiTE log.

### HiLiTE configuration data:

HiLiTE uses data from installed files to process your models and command file. You may be able to change some of these files or may receive occasional updates that enhance HiLiTE's processing.

**HiLiTEBlocks.xml and HiLiTETemplates.xml:** These files define the block and port types from which HiLiTE can generate code or test cases and the templates that it uses for generation. The database is supplied with the HiLiTE installation and resides in the %HILITE\_HOME% Data directory. You can update or add to the templates as needed using the HiLiTE TemplateManager. The TemplateManager is

installed with HiLiTE and the documentation is provided in Section 10. You can find it at Start→Programs→HiLiTE→HiLiTE TemplateManager. The application includes online help as well.

**For user information only – no changes are permitted**

**Mapping files:** The XML files that map the block types from the MATLAB model to the internal block types that HiLiTE uses. These files are supplied with the HiLiTE installation and reside in the %HILITE\_HOME%\BlockLibraries\ directory.

**Keyword files:** The XML files that define options specified in the command file. These files are supplied with the HiLiTE installation and reside in the %HILITE\_HOME%\Data directory.

## **2.4 Generating Tests for a Specific Test Harness**

HiLiTE can output test vectors to use in different test harnesses. Currently, it will always output vectors for the HiLiTE test harness. You can specify that you also want output to use with CTP or SATGT. For more information about specifying the output test vector format for a test harness, refer to the OptionSet information in Section 4.3.1, Command OptionSets and Options.

## 3 A Brief HiLiTE Tutorial

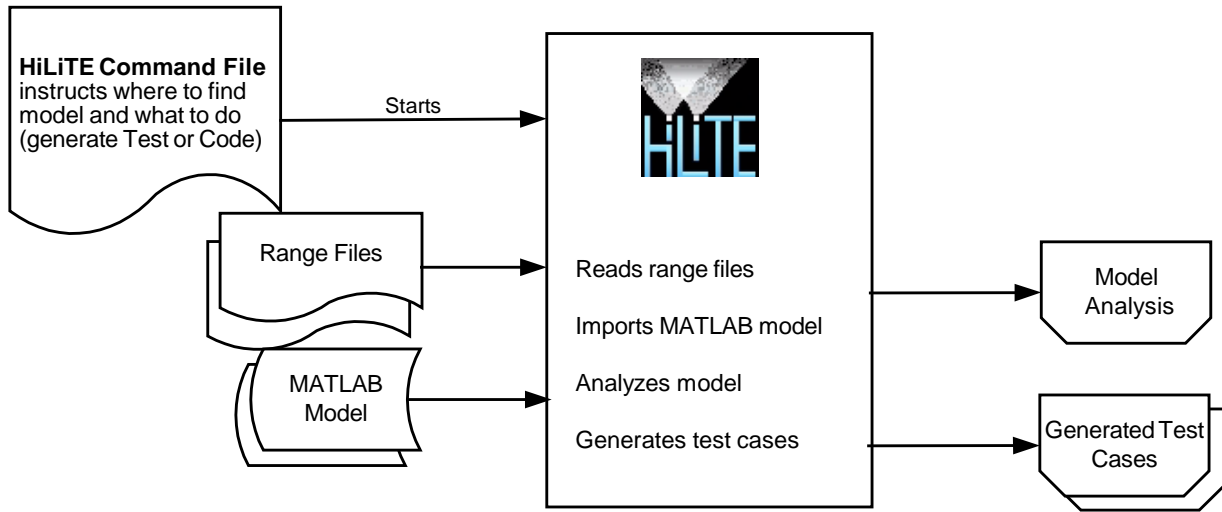
This section steps you through five basic steps of generating tests with HiLiTE to give you a quick, hands-on overview of how to use the tool. You can create new files for each step or simply use the examples included in your installation.

1. Create a MATLAB Simulink model using a block library.
2. Create a HiLiTE command file.
3. Create a range file for a math-based model.
4. Run HiLiTE to generate tests.
5. Examine the HiLiTE output that includes test vectors.

You can run the example directly from the installation folder given below (if you have write permission), or you can copy the entire folder to your work area and run it from there.

### 3.1 How HiLiTE Works

Figure 2 illustrates how HiLiTE processes data to generate code and test cases.

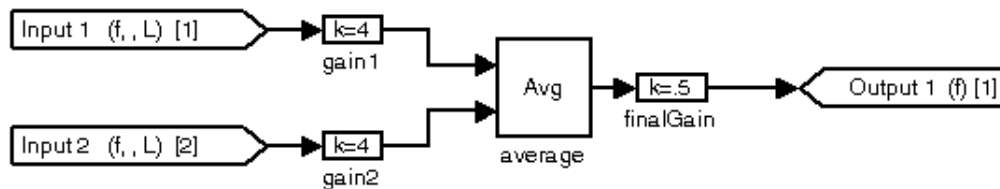


**Figure 2 - Simplified view of HiLiTE processing**

The user supplies the command file and the documents shown on the left side of the illustration. The command file refers to the model and symbol files and includes instructions for how HiLiTE should process them (whether to generate tests or code, the order to read symbol files, and so on). The central box summarizes how HiLiTE processes the information to generate the test cases.

### 3.2 Step 1: Create a Model

The HiLiTE installation comes with several example models. For this tutorial, you may want to examine the model *StrangeSum.mdl* found in: `%HILITE_HOME%\Examples\HiLiTE-TG-HAM\MathExamples\`. (`%HILITE_HOME%` is the folder where you installed HiLiTE (e.g. `C:\HiLiTE`))



**Figure 3 - Simulink model StrangeSum.mdl**

The data type of each input port block is set to double/float in the model, as is the convention for HAM models. HiLiTE needs the value of this data type to generate tests.

### 3.3 Step 2: Create a HiLiTE Command File

HiLiTE command files contain the parameters HiLiTE uses to run a model and generate tests. The files use XML formatting; they can be created and read in any text editor. *StrangeSum.hilite* is shown in Figure 4.

```
<?xml version="1.0"?>
<HiLiTEParameters>
  <ProjectPath>.</ProjectPath>

  <!--Specify the directory to store reports -->
  <TestOutput root="c:\hilite\TestingOutput">
    <Category name="Reports">StrangeSumOutput</Category>
    <Category name="Vectors">StrangeSumOutput</Category>
  </TestOutput>

  <!--Specify which HiLiTE extension (block library support) to use-->
  <Extension>HAM</Extension>

  <!-- describe range file format -->
  <ColumnSpec specName="OpRangeOnly" delimiters="," commentChars="#"
    titleLines="1">
    <Column name="symbol" col="0"/>
    <Column name="opmin" col="1"/>
    <Column name="opmax" col="2"/>
  </ColumnSpec>

  <!-- specify name of range file -->
  <SymbolFile fName="StrangeSum_Ranges.csv" specName="OpRangeOnly" />

  <!-- specify operation (generate test; models to test)-->
  <Command name="TestGen">
    <Model name="StrangeSum" />
  </Command>
</HiLiTEParameters>
```

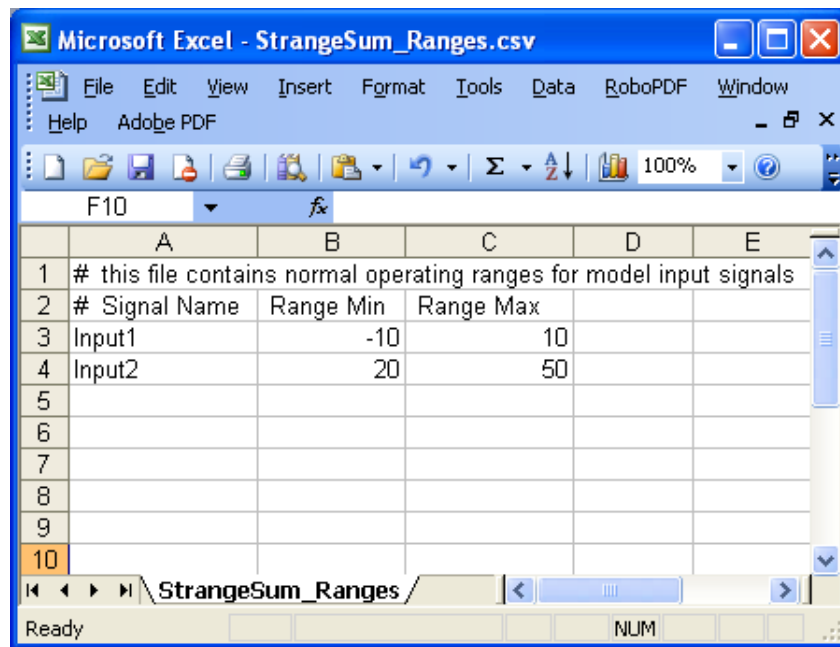
**Figure 4 - HiLiTE command file for model StrangeSum.mdl**

This example is fairly typical of simple HiLiTE command files. It begins with information about where the model is (project path) and where to put the generated output (test output). The next three areas (Extension, ColumnSpec, SymbolFile) describe where HiLiTE will find required input data and how it is formatted. The last part of the file is the actual command (TestGen) along with the model it uses.

### 3.4 Step 3: Create a Range File

Range files specify the normal operating ranges for the input ports of a mathematical model. This information is essential, since many tests are generated based upon boundary conditions of the normal range at the inputs and outputs of each block. HiLiTE automatically propagates user-specified ranges at the model inputs through the blocks in the model to compute the normal range at the inputs and outputs of each block.

Range files are typically created in a comma-separated-value (CSV) format using Microsoft Excel. They contain a row for each input port (signal) of the model. Figure 5 shows *StrangeSum\_Ranges.csv*, the range file for the StrangeSum model.



	A	B	C	D	E
1	# this file contains normal operating ranges for model input signals				
2	# Signal Name	Range Min	Range Max		
3	Input1	-10	10		
4	Input2	20	50		
5					
6					
7					
8					
9					
10					

**Figure 5 - Range file for model StrangeSum.mdl**

Note that if you specify the normal range values for each input in the HAM Input Port block, you do not need to create a range file. Information in the range file will override ranges in the model; this may be desirable if you wish to test different ranges, as you can easily change the range file without disturbing the model.

### 3.5 Step 4: Run HiLiTE to Generate Tests

1. Launch HiLiTE Shell from from the **Start** menu (select **Programs→HiLiTE→HiLiTE Shell**).

When you launch HiLiTE Shell, it displays a list along the left side of the Test Selection page. The list shows the names of the MATLAB models or HiLiTE command files contained in the last folder (and its subfolders) that you used. Figure 6 shows the list of HiLiTE Command files in the installed Examples folder.



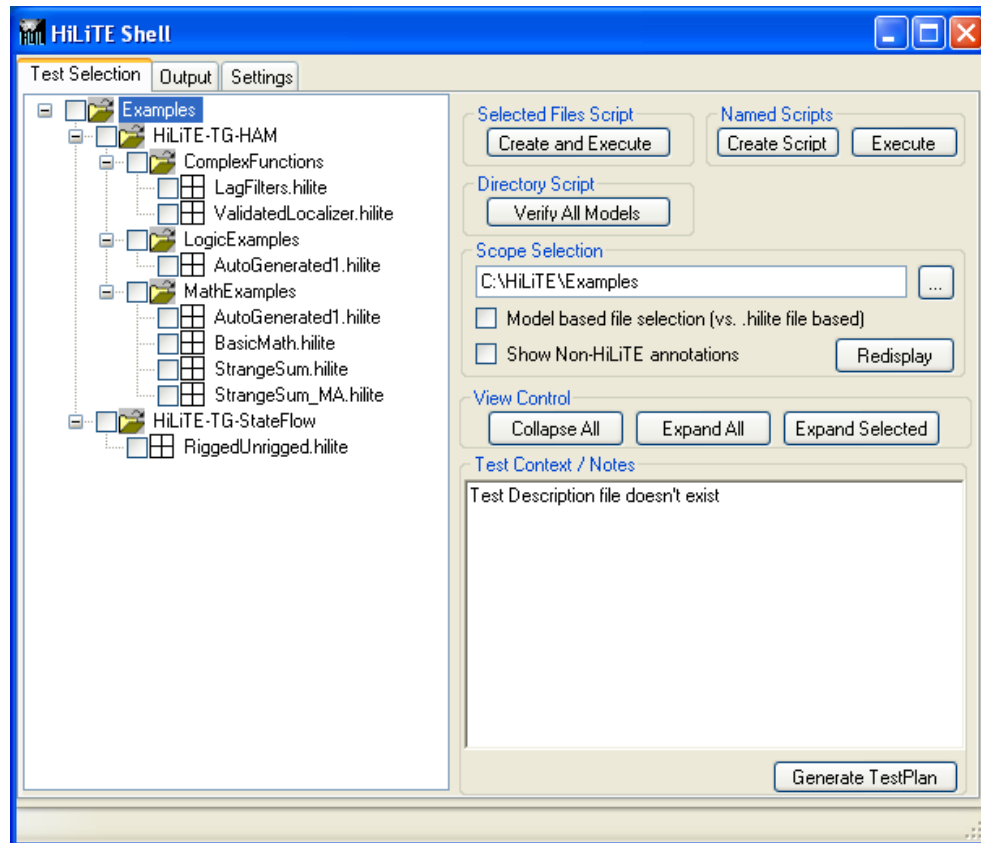



Figure 6 - HiLiTE Shell Test Selection page

To select the files to use:

2. Under Scope Selection, click the  button. HiLiTE Shell displays the Browse for Folders dialog.
3. Browse to the folder containing *StrangeSum.hilite* (If you accepted all defaults when you installed HiLiTE, this will be C:\HiLiTE\Examples\HiLiTE-TG-HAM\MathExamples\ ) and click **OK**.
4. Make sure the box labeled **Model based file selection** is *not* checked.
5. In the displayed list, click the box next to **StrangeSum.hilite**.
6. Click the **Create and Execute** button at the top of the HiLiTE Shell window. HiLiTE Shell displays the Output page. As the command file is processed, the MATLAB window containing the model will appear briefly on your screen (Figure 7). The MATLAB windows close as HiLiTE completes processing. The Output page shows the console log summarizing the steps of HiLiTE processing and shows whether test generation completed successfully (Figure 8).

### 3 A Brief HiLiTE Tutorial

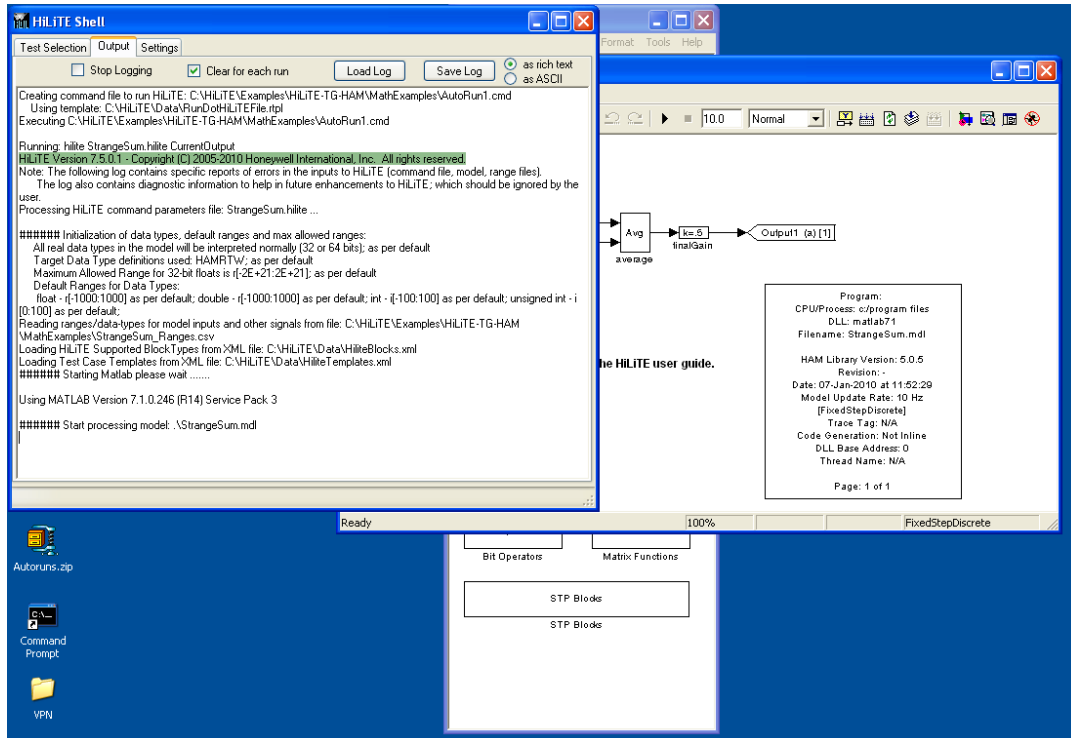


Figure 7 - HiLiTE Shell Output page and StrangeSum MATLAB model

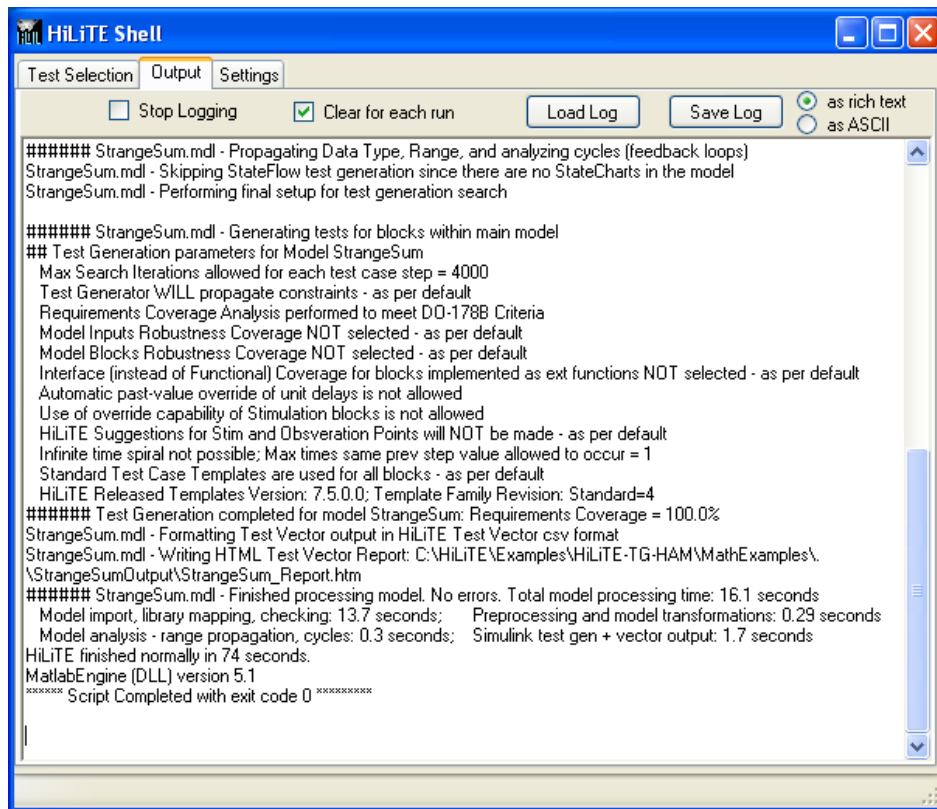


Figure 8 - Completed HiLiTE Console Log for StrangeSum command file

## Advanced Users

If you are already familiar with HiLiTE, you may prefer to use a DOS command window to run HiLiTE. In your Windows Explorer, browse for the name of the *.hilite* file.

1. Note or copy (Ctrl-C) the path to the file.
2. Start a DOS command window. (Start>Programs>Accessories>Command Prompt).
3. Change directories to the path for the *.hilite* file. (For the StrangeSum example, enter:  
`cd %HiLiTE_Home%\Examples\HiLiTE-TG-HAM\MathExamples`)
4. At the next command prompt, type `hilite filename.hilite`. (For the StrangeSum example, type: `hilite strangesum.hilite`)

The DOS command window shows HiLiTE's console log indicating progress. When HiLiTE opens your model, MATLAB will start up, opening both the model and the HAM window. When HiLiTE has completed processing the commands in *StrangeSum.hilite*, you will see the next command prompt at the bottom of the DOS command window. When all applications are open, your desktop will look something like Figure 9.

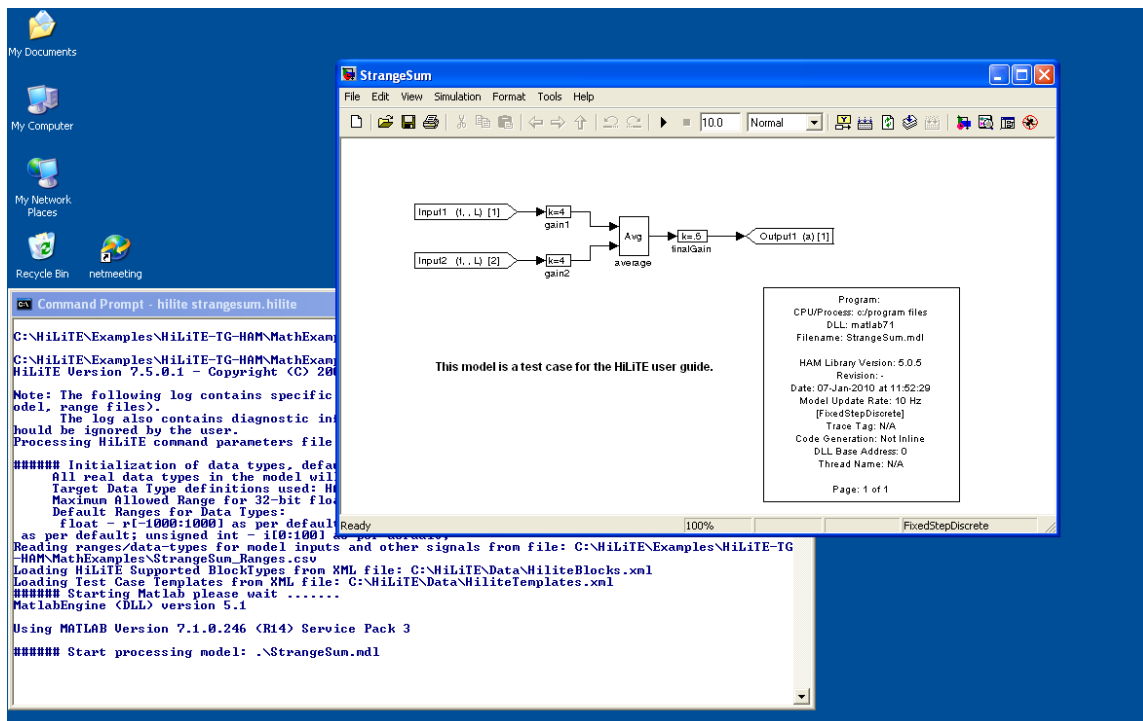
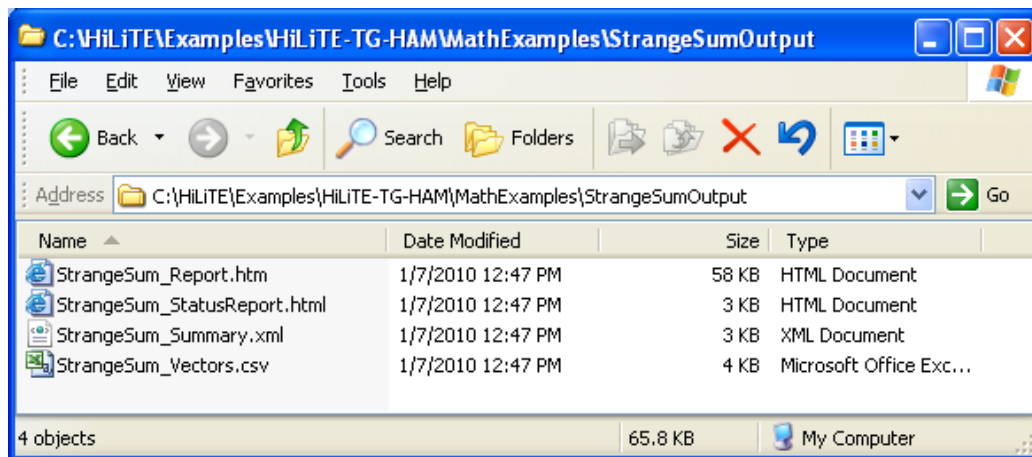


Figure 9 - HiLiTE command execution

Read the log in the DOS command window to see how HiLiTE processed the model and the commands in *StrangeSum.hilite*.

## 3.6 Step 5: Examine the HiLiTE Output

HiLiTE stores the test vectors and reports in the path specified in the command file. For *StrangeSum.hilite*, this path is *.\StrangeSumOutput*, within the folder that contains the model.



**Figure 10 - HiLiTE-generated output files for model StrangeSum.mdl**

Display this folder in the Windows Explorer. You should see four files:

1. **Strangesum\_StatusReport.html** contains the status of test generation, including any un-testable conditions in the model (design problems) and HiLiTE tool limitations.
2. **Strangesum\_Report.htm** contains block-level test case templates and generated test vectors in HTML format.
3. **Strangesum\_Vectors.csv** contains generated test vectors in machine-readable CSV format for input to a test harness. Note: this format is an example format; the actual format used in a project will be specific to a particular test harness such as CTP (.tpi) or SATGT.
4. **StrangeSum\_Summary.xml** summarizes the HiLiTE execution results, showing the models used, success of the Simulink and Stateflow test generation, requirements coverage status, success of the model analysis, and error status.

#### 3.6.1 Examine the Status Report

Figure 11 shows the status report for the StrangeSum example indicating that all block tests were generated without errors.

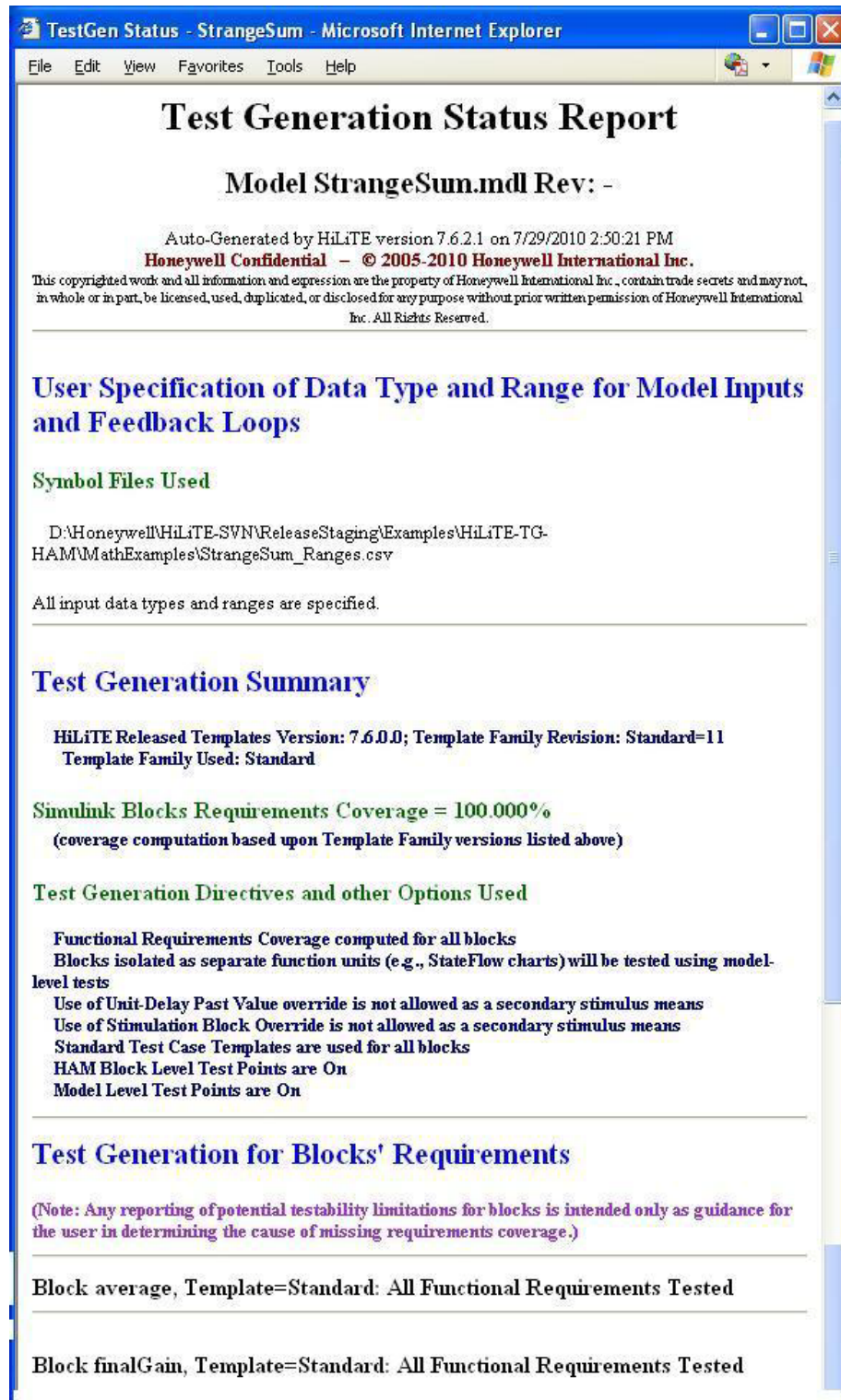


Figure 11 - StrangeSum test status report

You can display the status report by double-clicking on the file `%HILITE_HOME%\Examples\HiLiTE-TG-HAM\MathExamples\StrangesumOutput\StrangeSum_StatusReport.html`.

### 3.6.2 Examine the Test Vector Report and Test Vectors

The test report (strangesum\_Report.htm) contains tables of the test case templates and generated test vectors for each block. Figure 12 shows part of the test vector report for the StrangeSum example.

Test case templates define the input and output requirements for each test for each block. In the portion of the report shown in Figure 12, four templates require specific input values that should produce the expected output values. For example, the NearMax template requires input values of the normal range maximum for the input port (Input=120). Testing will be successful if the value at Output is 60—the normal range maximum for that port

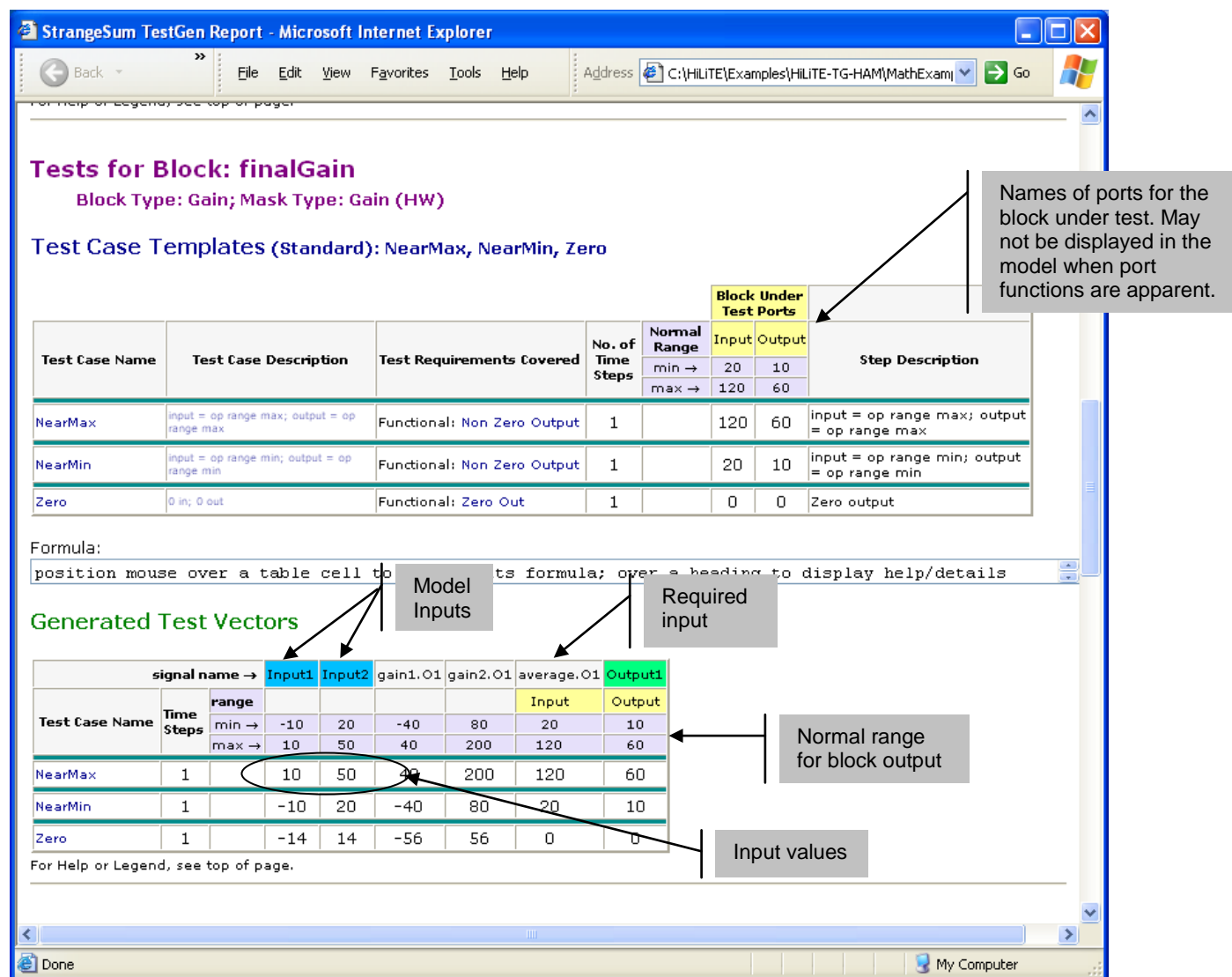


Figure 12 - StrangeSum test report (partial)

The Generated Test Vectors show the actual values input to the block under test and the value at the block's output port. The template requires that Input1 and Input2 have values of 10 and 50, but since these ports are not directly accessible, HiLiTE must determine what values to give the model inputs to affect the correct values. In the generated test vector for NearMax, notice that the block under test receives four inputs. These inputs are two model inputs (Input1 and Input2, named in blue cells at the top of the Test Vectors table) and two intermediate values named gain1.O1 and gain2.O1. The model input



values were determined by HiLiTE. The intermediate values are the output of the gain1 and gain2 blocks and also the direct input to the Average block—their values are those required by the template. The value of the block output is 60, so the test was successful.

You can see the entire report by double-clicking on the file `%HILITE_HOME%\Examples\HiLiTE-TG-HAM\MathExamples\TestGenOutput\strangesum_Report.htm`. If you have your browser set to allow active content, buttons at the top of the report let you display Help (with a glossary) and a Legend that explains the color coding. The Help and Legend are reproduced below so that you can view them side-by-side with the HTML page.

### Help:

#### Context Sensitive Help

You can get specific help by positioning the mouse over certain headings of table columns

#### Glossary

##### Test Case

A time sequence of test vectors to be exercised against the model to test high- or low- level requirements of the model. Test cases in tables are separated by thick colored lines. The color of the line (dark blue-green or violet-red--see legend) above the first vector of a test case denotes whether the model code must be initialized to run the test case.

##### Test Vector

A set of signal values to be applied as model inputs and corresponding output values expected as model outputs. Each test vector can have one or more time steps as indicated in the *Time Steps* column of the table. The indicated model input values must be applied for the given number of time steps and the model output expected value must be measured at the end of the last time step in the vector. A test vector is displayed as a row in a table. The first column (after names, description) denotes the number of time steps in the test vector. The rest of the columns denote a model input, constant, model output or an intermediate signal in the model. Intermediate signals and constants are not essential parts of a test vector; they are included only for user guidance.

##### Time Step

A time step is one step (frame) of execution of the model. For a model with a rate of 10 Hz, there are 10 time steps per second, each of .1 second duration.











##### Signal Name

In the test vectors table, signal name is displayed in the top row, with a background color of sky blue for input and spring green for output (see the legend). For user-level model inputs and outputs (inport and outport blocks), the signal name appears inside that block in the model. For model outputs that are implied "test points" at the output of blocks (as in HAM blocks), the signal name uses the format "*blockname\_TP*". For intermediate signals (displayed without any special background color), the signal name uses the format "*blockname.O1*", where the rightmost digit indicates the index of the block's output port that this signal denotes.

##### Normal Range

Each signal in the model has a "normal operating range," which denotes the typical range of values of that signal during normal operation. The normal range is significant only for numeric signals; for booleans it is always [0, 1]. The normal range comprises a minimum and a maximum value. For model inputs, the user specifies the normal range to HiLiTE either in the model or in a separate range file. For all intermediate signals and model outputs, the normal range is computed by HiLiTE using range propagation.

### Legend:

 : model input signal (inport)	 : model output signal (outport)
 : input or output port of Block Under Test	 : a constant
 : min or max value of normal operating range for this column's signal	 : extra execution step inserted for signal setup/pass-thru
 : A value in this cell was altered to fit within Max Allowable Range, review template to ensure this is not an error	
 : The formula value in this cell is either malformed or entirely outside the Max Allowable Range; correct the template	
 : a normal test case not requiring model initialization before its first step	
 : a test case that requires model initialization before its first step	
<i>number, X</i> : Don't Care - nominal value computed as necessary	

### 3 A Brief HiLiTE Tutorial

A machine-readable (csv or tpi) file contains the actual vectors that will be used by the test harness. Figure 13 shows the vector file for StrangeSum displayed in Excel.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	HiLiTE-1-1	Vector	Initialize	InterfaceC	ColumnCo	ColumnNa	Information	Tolerance					
2	Information	ModelName	Model Ver	HiLiTE Ver	GenerationDate=1/7/2010 12:46:58 PM								
3	Comment	HiLiTE Released Templates Version: 7.5.0.0; Template Family Revision: Standard=4											
4	InterfaceCo	2	1	0	0	3							
5	ColumnCo	Repetition	Input:1	Input:2	Output:1	Local_Out	Local_Out	Local_Out					
6	ColumnNa		Input1	Input2	Output1	average.O	gain1.O1	gain2.O1					
7	Comment	Normal Range Min											
8	Comment	Normal Range Max											
9	Comment	Tolerance (if specified for an output) denotes relative (fractional) tolerance otherwise default relative tolerance of .0001 is used											
10	Tolerance												
11	Comment	Tests for b average MaskType Average (HW)											
12	Comment	Test Case: AllMax [1] Block: average MaskType Average (HW) Requireme Functional - SumOperation (1 of 2)											
13	Vector	1	10	50	X	120	X	X					
14	Comment	Test Case: AllMin [2] Block: average MaskType Average (HW) Requireme Functional - SumOperation (1 of 2)											
15	Vector	1	-10	20	X	20	X	X					
16	Comment	Test Case: MaxOut [1] Block: average MaskType Average (HW) Requireme Functional - SumOperation (1 of 2)											
17	Vector	1	10	0	X	20	X	X					
18	Comment	Test Case: MaxOut [2] Block: average MaskType Average (HW) Requireme Functional - SumOperation (2 of 2)											
19	Vector	1	0	50	X	100	X	X					
20	Comment	Test Case: MinOut [1] Block: average MaskType Average (HW) Requireme Functional - SumOperation (1 of 2)											
21	Vector	1	-10	0	X	-20	X	X					
22	Comment	Test Case: MinOut [2] Block: average MaskType Average (HW) Requireme Functional - SumOperation (2 of 2)											
23	Vector	1	0	20	X	40	X	X					
24	Comment	Tests for b finalGain MaskType Gain (HW)											
25	Comment	Test Case: NearMax Block: finalGain MaskType Gain (HW) Requireme Functional - Non Zero Output											
26	Vector	1	10	50	60	X	X	X					
27	Comment	Test Case: NearMin Block: finalGain MaskType Gain (HW) Requireme Functional - Non Zero Output											
28	Vector	1	-10	20	10	X	X	X					
29	Comment	Test Case: Zero Block: finalGain MaskType Gain (HW) Requireme Functional - Zero Out											
30	Vector	1	-14	14	0	X	X	X					
31	Comment	Tests for b gain1 MaskType Gain (HW)											
32	Comment	Test Case: NearMax Block: gain1 MaskType Gain (HW) Requireme Functional - Non Zero Output											
33	Vector	1	10	X	X	X	40	X					
34	Comment	Test Case: NearMin Block: gain1 MaskType Gain (HW) Requireme Functional - Non Zero Output											
35	Vector	1	-10	X	X	X	-40	X					
36	Comment	Test Case: Zero Block: gain1 MaskType Gain (HW) Requireme Functional - Zero Out											
37	Vector	1	0	X	X	X	0	X					
38	Comment	Tests for b gain2 MaskType Gain (HW)											
39	Comment	Test Case: NearMax Block: gain2 MaskType Gain (HW) Requireme Functional - Non Zero Output											
40	Vector	1	50	Y	Y	Y	200						

Figure 13 - Contents of StrangeSum\_Vectors.csv



### 3.6.3 Examine the XML file of HiLiTE execution results

Figure 1 shows the HiLiTE execution results in StrangeSum\_Summary.xml.

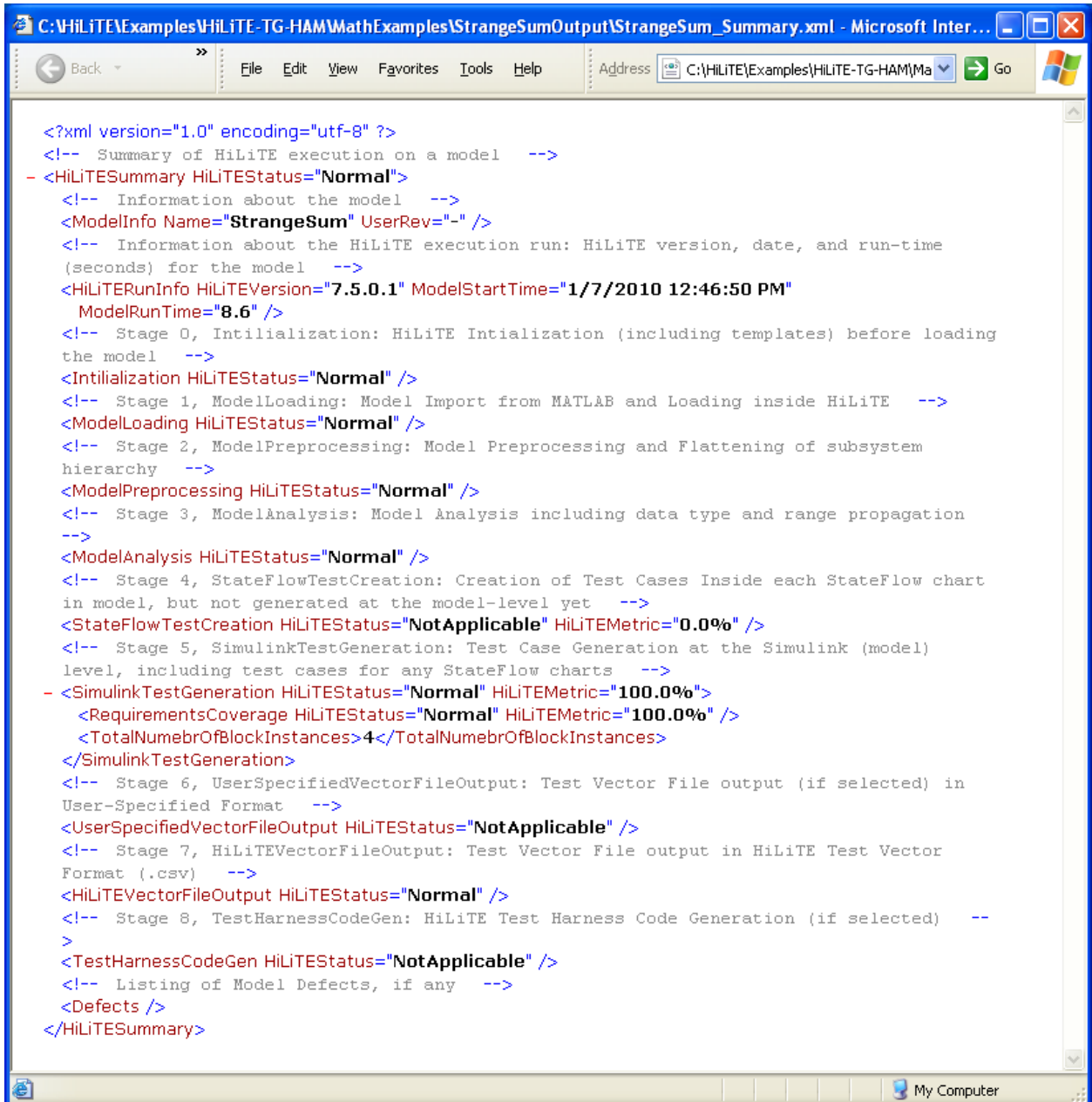


Figure 14 - Contents of StrangeSum\_Summary.xml



## 4 Creating HiLiTE Command Files

HiLiTE uses a command file of execution parameters in XML format. You can create command files from scratch or edit a sample file that comes with HiLiTE. You will probably wish to create a file for each project from which you can generate test cases for several related models. You can edit the files in any plain text or html editor (do not use Microsoft Word as it inserts hidden formatting characters).

The command file instructs HiLiTE:

- Whether to generate test cases or to perform model analysis.
- Where the input files are located.
- Where to place the generated output.

### 4.1 Command File Format

HiLiTE reads commands from a text file that is written in XML, a tagging/markup language. XML tags instruct HiLiTE how to run: what models and values to use, what information to output, and so on. Tags are written as text enclosed in angle brackets, e.g. <Command />. Figure 15 is a HiLiTE command file with comments and callouts that label *elements*, *attributes*, *keys* and *values*.

- Elements are the high-level instructions shown first after the opening bracket.
- Attributes give more information about how to process the element. They consist of a name (called a key) and a value which quantifies the key. Attributes are contained within their element's brackets.
- OptionSets are collections of Option keys and values that affect how each command is carried out. OptionSets must be named.

Figure 15 shows several basic HiLiTE elements with appropriate attributes and values. The tables later in this chapter define these and other elements.

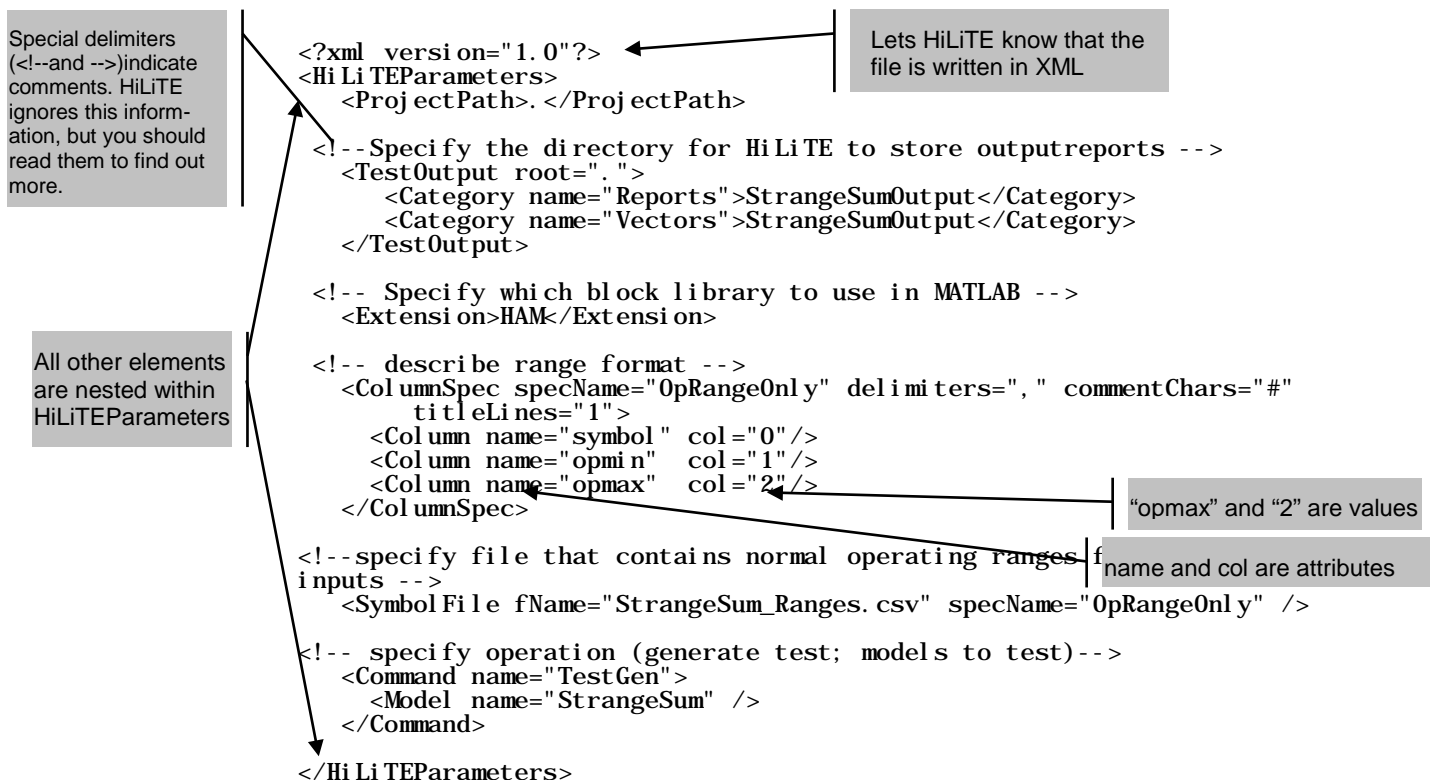


Figure 15 - Command file to generate testcases

### Reading a command file

To understand Figure 15, read the comment lines that appear above most elements in the file. Comments are indicated with tag delimiters like this: `<!--comment-->`. The top level element `<HiLiTEParameters>` encloses all other elements. These other elements contain information HiLiTE requires to find a model and its associated symbol files. The model and range files are found in `<ProjectPath>`. `<ColumnSpec>` and `<SymbolFile>` elements tell how to find relevant data in the range files. The `<Command>` element tells what to do (generate tests, in this case) and where to place the output.

The end of an instruction is indicated with a forward slash (/) either within the second of pair of tags (note `</HiLiTEParameters>` which ends the top level element) or at the end a self-terminating tag (for example, each `Column` element ends with `/>`). A tag may include one or more attributes to clarify the instruction.

**Note:** XML is case sensitive; it is important to notice and enter tag and attribute names using the correct capitalization.

Tables in the following sections explain most elements and attributes that are used in HiLiTE command files. Table 1 summarizes the basic elements and Table 2 and Table 3 describe elements that are available for more advanced use. Chapter 6 includes more command file examples and information about generating test cases. Appendix H describes command file elements that developers may find useful.

### Path formatting in examples

File paths specified in examples throughout this document follow these conventions:

- (period or dot) indicates the current working directory (see `<ProjectPath>` example in Table 1) You can use this convention in creating HiLiTE command files.

an absolute system path (e.g., starting with a drive letter) is self defining; all other paths are relative to the current working directory or the project path.

## 4.2 Basic HiLiTE Command-File Elements

Elements described in Table 1 are those you will use most often in creating command files to test models. In creating command files, note that:

- Capitalization is important; command elements other than OptionSets, attributes, and values are case sensitive. Type them exactly as shown in the table and as given in your own model and file names.
- OptionSet names and option key values are generally not case sensitive, but using mixed case spellings aids readability.
- Single and double quote marks can be used interchangeably in command elements, as long as the same mark is used in a pair.

**Table 1. Basic HiLiTE command file elements**

Element Tag	Description	Example
<b>&lt;HiLiTEParameters&gt;</b> <b>&lt;/HiLiTEParameters&gt;</b> Required	identifies the enclosed elements as instructions to HiLiTE. This is the top-level tag—all other elements fall within HiLiTEParameters. Capitalization as shown in this—and all—tags is required.	<b>&lt;HiLiTEParameters&gt;</b> <ProjectPath>./ProjectPath> <ColumnSpec specName='OpRanges' delimiters=', ' commentChars='#' titleLines='1'> <Column name='symbol' col='0'/> <Column name='opmin' col='1'/> <Column name='opmax' col='2'/> </ColumnSpec> <SymbolFile fName='ranges.csv' specName='OpRanges' /> <Command name='TestGen' > <Model name='testgenexample1' /> </Command> <b>&lt;/HiLiTEParameters&gt;</b>
<b>&lt;ProjectPath&gt;</b> <b>&lt;/ProjectPath&gt;</b> Required	specifies where the models, symbol files, and other input to be processed are located. The default path used in the example command files is the current working directory (where the command file is stored). Current working directory is represented with a dot (.); see first example to the right. Using this path designation makes the command files portable.	<b>&lt;ProjectPath&gt;./ProjectPath&gt;</b>  <b>&lt;ProjectPath&gt;C:\HiLiTETemp\SCN10b&lt;/ProjectPath&gt;</b>
<b>&lt;TestOutput&gt;</b> <b>&lt;/TestOutput&gt;</b> Optional	specifies where HiLiTE should place the test cases and test reports. If you do not use this element, HiLiTE places output into subdirectories of the directory from which you are running HiLiTE. .\Vectors will hold the test cases and .\Reports will hold the reports. TestOutput has one attribute: <i>root</i> (required)—specifies the path to which HiLiTE should send output. Use "." To specify the directory from which you are running the command file; otherwise, enter the full path.	<b>&lt;TestOutput</b> root="c:\TestingOutput"> <Category name="Reports">MyReports</Category> <Category name="Vectors">MyVectors</Category> <b>&lt;/TestOutput&gt;</b>  <b>&lt;TestOutput</b> root="." />

Element Tag	Description	Example
<b>&lt;Category /&gt;</b> Optional	<p>gives further path specification for where HiLiTE should place output. It is a subelement of TestOutput. It has one attribute:</p> <p><b>name</b> (required)—specifies paths for specific output. Use these names:</p> <p><i>Vectors</i>—specifies where HiLiTE will put the generated test cases in the format required by the test harness tools. Default path is /Vectors as a subdirectory of the current working directory.</p> <p><i>Reports</i>—specifies where HiLiTE will put the html reports describing the test cases. Default path is /Reports as a subdirectory of the current working directory.</p>	<pre>&lt;TestOutput root="."&gt;   &lt;Category name="Reports"&gt;MyReports&lt;/Category&gt;   &lt;Category name="Vectors"&gt;MyVectors&lt;/Category&gt; &lt;/TestOutput&gt;</pre>
<b>&lt;Extension&gt;</b> <b>&lt;/Extension&gt;</b> Required for most users	<p>lets you specify the types of blocks and templates that support the model under test. This element has no attributes.</p> <p>Extension keywords for supported block libraries are:</p> <p><b>HAM, Boeing, CCC, Simulink</b></p> <p>If you do not specify Extension, HiLiTE will default to <i>Simulink</i>, implying only raw Simulink block support. Users who create their own library mappings for experimentation can use the &lt;LibraryMapping&gt; element (See Appendix E)</p>	<pre>&lt;Extension&gt;HAM&lt;/Extension&gt;  &lt;Extension&gt;Boeing&lt;/Extension&gt;  &lt;Extension&gt;CCC&lt;/Extension&gt;  &lt;Extension&gt;Simulink&lt;/Extension&gt;</pre>
<b>&lt;ColumnSpec&gt;</b> <b>&lt;/ColumnSpec&gt;</b>	<p>describes the format of a range file containing range/data type information. It has the following attributes:</p> <p><b>specName</b>—the name of the column specification. This name will be used by any &lt;SymbolFile&gt; specification that uses this column format for its data.</p> <p><b>delimiters</b>—a list of the delimiters used to separate columns in the spec. Common delimiters are , (comma), \t (tab), and ' ' (space). Use these character sequences to define non printable delimiters:</p> <p>\t – TAB</p> <p>\n – NEWLINE</p> <p>\r – CARRIAGE RETURN</p> <p>\f – FORM FEED</p> <p><b>commentChars</b>—a list of chars, any of which, if found in the first column of a line, indicates that the entire line is a comment</p>	<pre>&lt;ColumnSpec specName='OpRanges'   delimiters='\t' commentChars='#' titleLines='1'&gt;   &lt;Column name='symbol' col='0'/&gt;   &lt;Column name='opmin' col='1'/&gt;   &lt;Column name='opmax' col='2'/&gt; &lt;/ColumnSpec&gt;</pre>

Element Tag	Description	Example
	<b>titleLines</b> —the number of lines at the top of the file that contain information that should be ignored.	
<b>&lt;Column /&gt;</b>	<p>is a required subelement of the &lt;ColumnSpec&gt; element. It maps a specific column name to a column number in the symbol file. It consists of two attributes:</p> <p><b>name</b>—the column name. Column names are pre-specified. Currently supported names are:</p> <ul style="list-style-type: none"> <li>'symbol' (required) – this column contains symbol names</li> <li>'opmin' (optional) – this column contains the min value the symbol may have</li> <li>'opmax' (optional) – this column contains the max value the symbol may have.</li> <li>'allowmin' (optional) – this column contains the extreme allowable min value the symbol may have</li> <li>'allowmax' (optional) – this column contains the extreme allowable max value the symbol may have.</li> <li>'lir' (optional) – this column contains the language independent representation of the data type used for this value.</li> <li>'utype' (optional) – this column contains the universal data type used for this value.</li> <li>'tolpct' (optional) – this column contains the % tolerance to be applied in the measurement of a model output or observation point denoted by a symbol. <i>tolpct</i> should be specified only if the default tolerance needs to be overridden for that variable measurement.</li> </ul> <p><b>col</b>— the number of the column containing the data. Note that this is zero-based; the first column in the file (column A in Excel) is 0.</p> <p>The Column element can take an optional subordinate element: &lt;Translation&gt; (see Table 3)</p>	<pre>&lt;ColumnSpec specName='Tolerances'   delimiters='\\t' commentChars='#' titleLines='1'&gt;   &lt;Column name='symbol' col='0'/&gt;   &lt;Column name='tolpct' col='1'/&gt; &lt;/ColumnSpec&gt;</pre>

Element Tag	Description	Example
<b>&lt;SymbolFile /&gt;</b> optional	<p>names a file from which HiLiTE retrieves system dictionary and range information. The element indicates which files to read and which &lt;ColumnSpec&gt; to use when reading them. You can include multiple SymbolFile references. SymbolFile has the following attributes:</p> <p><b>fName</b>—the name of the range file to be read. This can be a fully qualified name or a name relative to the ProjectPath.</p> <p><b>specName</b>—the name of the column specification to use when reading this file. This indicates the format of the file.</p> <p>Note: the order in which you specify symbol files is important. HiLiTE reads all symbol files at once and uses them in the order they are specified. If more than one symbol file includes the same column specifications, HiLiTE uses the values from the <i>last</i> file it reads in which that symbol attribute is defined.</p>	<pre>&lt;SymbolFile fName='filename.csv' specName='OpRanges'/&gt;</pre> <p>Note: the <i>specName</i> here reference the ColumnSpec examples given above</p> <p><b>Multiple SymbolFile declarations can be provided, potentially referencing different <i>ColumnSpecs</i>:</b></p> <pre>&lt;SymbolFile fName='InRanges.csv' specName='OpRanges' /&gt; &lt;SymbolFile fName='OutTol.csv' specName='Tolerances' /&gt;</pre>
<b>&lt;Command&gt;</b> <b>&lt;/Command&gt;</b> required	<p>specifies the command HiLiTE will execute. It can have several attributes, depending on the command specified. It has three subordinate elements: <i>Model</i>, <i>Group</i>, <i>Option Set</i>. The attributes of Command are:</p> <p><b>name</b>—required attribute that gives the name of the actual command that HiLiTE is to execute. In most cases you will use this command:</p> <p><i>TestGen</i>—Invokes the HiLiTE Test Generator.</p> <p>See Table 2 and Table 3 for more command options and attributes of TestGen.</p> <p>See <i>Appendix H</i> for advanced/diagnostic commands.</p>	<pre>&lt;Command name='TestGen' &gt;   &lt;Model name='StrangeSum' /&gt; &lt;/Command&gt;</pre>



Element Tag	Description	Example
<b>&lt;Model /&gt;</b> required	<p>is a sub-element of &lt;Command&gt; that specifies the model to which the command applies. Model tags have attributes and sub-elements.</p> <p><b>name</b>—the only required attribute. It specifies the name of the model file to be processed. The model file is assumed to be in the directory specified by ProjectPath. The .mdl extension need not be included; it will be appended automatically to the name.</p> <p>Models that do not have a HAM model revision block (that specifies execution rate) must have the rate specified in the Model element using one (and only one) of the following attributes:</p> <p><b>periodMs</b> (optional)—indicates the period (in milliseconds) of a single execution cycle for this model.</p> <p>OR</p> <p><b>rateHz</b> (optional)—gives the number of execution cycles per second; use this only if the rate is not specified in the model.</p>	<pre>&lt;Command name='TestGen' &gt;   &lt;Model name='StrangeSum' /&gt; &lt;/Command&gt;</pre> <p>Only for a model without a HAM model revision block:</p> <pre>&lt;Command name='TestGen"&gt;   &lt;Model name='YX440d' rateHz='50' /&gt; &lt;/Command&gt;</pre>

## 4.3 Advanced HiLiTE Command-File Elements and Options

HiLiTE command files can specify many more elements than described in Table 1. Table 2 and Table 3 describe elements that are useful once you are familiar with basic HiLiTE usage. These elements let you define special testing circumstances, identify specific behaviors, and control output style and location. Appendix H describes command file elements that HiLiTE Team developers use for diagnostics and special circumstances. elements, command file

### 4.3.1 Command OptionSets and Options

You can specify command options to control HiLiTE operation, such as how to handle special testing circumstances. Options are grouped in OptionSets that are specified within command file `<Command>` and `<Model>` elements, described in Table 1.

Multiple OptionSets can be specified `<Command>` and `<Model>` elements at any level of the XML hierarchy. Each OptionSet must have a *name* attribute (all available OptionSets names are described in the following tables) and include one or more `<Option>` subelements.

`<Option>` elements control specific aspects of HiLiTE command operation. Each Option is specified as a “key-value pair,” where the key denotes the option name. Multiple Options can be specified within an OptionSet.

```
<Command name="TestGen">
  <OptionSet name="OptionSetA">
    <Option key="optionx">val uex</Option>
    <Option key="optiony">val uey</Option>
  </OptionSet>
  <OptionSet name="OptionSetB">
    <Option key="optionh">val ueh</Option>
  </OptionSet>
  ...
  <Model name="mymodel">
    <OptionSet name="OptionSetC">
      <Option key="optiong">val ueg</Option>
    </OptionSet>
  </Model>
</Command>
```

Figure 16 – Option sets and option key-value pairs in `<Command>` and `<Model>` elements

When creating HiLiTE command files that use OptionSets, note that:

- Most OptionSet names and Option key names are case-insensitive; however, using mixed-case spelling will increase readability. Check the StatusReport to ascertain whether HiLiTE properly recognizes them.
- All OptionSet specifications must precede the *Model* elements in the command file.

### Default Values

If you do not specify particular OptionSets or Option key values, HiLiTE will use default values for the missing OptionSets and Options. Default values for the Options are listed in the following tables along with general option descriptions. While certain options are required, many of them are useful mainly for refining test generation, and not necessary for most HiLiTE usage. HiLiTE's automatic use of default values means you needn't be concerned about those options.

### Multiple Instances of an OptionSet

If an OptionSet and a particular option is specified in both the Command element and a Model element, the innermost specification of an Option takes precedence. The HiLiTE command log specifies the context from which the value of each Option is taken.

Named OptionSets can also be specified at the Command level or Model level. The same named OptionSet can be specified at more than one level with either distinct or overlapping Options. If the same Option key is specified at more than one level, the order of precedence is: Model level, then Command level.

### 4.3.2 Attribute Specification in Command-File Elements

Attributes can be used within several of the XML elements of the HiLiTE command file to fine tune HiLiTE default behavior. An attribute is specified in the following manner.

```
<ElementTagA AttributeX="ValueX" AttributeY="ValueY" />
```

For backward compatibility, certain attributes are available as alternates to specifying an Option. These are noted in Table 2.

### 4.3.3 Command-File Elements to Control Test Generation and Test Vector Output Location

Most HiLiTE command file elements, option sets, options, and attributes are described Table 2. While certain elements are required, others are useful for refining some aspects of HiLiTE operation, but not necessary for simple usage. You may want to experiment with these after you become familiar with HiLiTE.

Table 2. Advanced HiLiTE command file elements for test generation and output location

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<b>&lt;Command&gt;</b> <b>&lt;Model&gt;</b>  optional attributes for controlling certain search parameters and test generation directives	<p>You can control certain search parameters and test generation directives by specifying attributes in the Command or Model element. This is an alternative (deprecated) to specifying Options in the respective OptionSets (described in the following rows). The following attributes can be specified (note: the inner specification at Model level supersedes the specification at Command level):</p> <p>MaxRecursionLevels, MaxIterations, PropagateConstraints</p> <p><i>MaxRecursionLevels</i>: (default=200) Specifying a higher value instructs HiLiTE to use more recursion contexts in the search.</p> <p><i>MaxIterations</i>: (default=4000) Specifying a higher value instructs HiLiTE to use more points of search iteration (expansion) for a particular test case step.</p> <p>InputsRobustnessCoverage, BlocksRobustnessCoverage, InterfaceCoverageForExtFunctions</p>	<pre>&lt;Command name="TestGen" &gt;   &lt;Model name='MyModel1'     <b>PropagateConstraints='No'</b>     <b>MaxIterations='12000'</b> /&gt; &lt;/Command&gt;</pre>
<b>&lt;Model&gt;</b> other optional attributes	<p><b>revision</b>: (string) HiLiTE allows user to specify the revision of a model in the command file and override the revision number (HAM) inside the model if any exists. This model revision string is displayed verbatim in all HiLiTE reports and vector/TPI files; i.e., no prefixes or suffixes are added by HiLiTE to the string provided in the command file. Specifying revision is optional. If revision is not specified, then HiLiTE will read the "Revision" information specified in HAM "Revision block", if present in the model.</p>	<pre>&lt;Model name='myModel' <b>revision='rev 41'</b> /&gt;</pre>
<b>&lt;Model&gt;</b> <b>&lt;SuppressBlock&gt;</b> <b>&lt;SuppressBlockType&gt;</b> <b>&lt;TemplateProc&gt;</b>	<p>You can suppress test generation by including &lt;SuppressBlock&gt; or &lt;SuppressBlockType&gt; as subelements of the Model element. If no procedures are specified, then all available template procedures will be suppressed.</p> <p><u>Note</u>: If both &lt;Block&gt; and &lt;SuppressBlock&gt; are specified, HiLiTE gives first preference to &lt;Block&gt;. HiLiTE ignores &lt;SuppressBlock&gt; and generates test cases only for the block specified in &lt;Block&gt;. The same logic applies to &lt;BlockType&gt; and &lt;SuppressBlockType&gt;</p> <p>To suppress specific procedures, include the subelement &lt;TemplateProc&gt;, which has one required attribute:</p> <p><b>name</b>: specifies which template procedures to suppress. Only the</p>	<pre>&lt;Model name="TestSum1" &gt;   &lt;SuppressBlock name="SumTwoConst1" &gt;     &lt;TemplateProc name="AllMax [12]" /&gt;   &lt;/SuppressBlock&gt; &lt;/Model&gt;  &lt;Model name="CL_CommandFeature_Misc" &gt;   &lt;SuppressBlockType name="Divide (HW)"&gt;     &lt;TemplateProc name="MAXMIN"/&gt;   &lt;/SuppressBlockType&gt; &lt;/Model&gt;</pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	specified template procedures will be suppressed during test generation.	
<b>&lt;BlockDBFile&gt;</b> <b>&lt;TemplateDBFile&gt;</b>	<p>top level elements that let you supply your own versions of blocks or templates.</p> <p>For either element, the value will be the name of the data base file with your version of the blocks or templates. If you do not supply a path, HiLiTE will find the file in the project path as shown in the example on the right. You can also supply an absolute path (e.g. D:\MyProject\HiliteBlocks_mine.xml) or an environment variable (e.g., %HiLiTE_HOME%\Data\HiliteBlocks_mine.xml).</p>	<pre> &lt;HiLiTEParameters&gt;   &lt;HiLiTEMgmt pauseOnCompletion="no" /&gt;   &lt;ProjectPath&gt;./ProjectPath&gt;   ...   &lt;Extension&gt;HAM&lt;/Extension&gt;   &lt;BlockDBFile&gt;HiliteBlocks_mine.xml   &lt;/BlockDBFile&gt;   &lt;TemplateDBFile&gt;HiliteTemplates_mine.xml   &lt;/TemplateDBFile&gt;   ...   &lt;Command name='TestGen'&gt;     ...   &lt;/Command&gt; &lt;/HiLiTEParameters&gt; </pre>
<b>&lt;OptionSet&gt;</b> <b>&lt;/OptionSet&gt;</b> optional subelement of either <Command> or <Model>	<p>subelement of either the <i>Command</i> element or a <i>Model</i> that holds a group of options. OptionSet has one required attribute and contains one or more &lt;Option&gt; subelements. Each type of OptionSet has its own set of Option keys. The OptionSets and their respective Option keys are described in the following rows of this table.</p> <p><b>name:</b> required attribute that names the specific OptionSet to be used. You must use one of the following names:</p> <p><i>General</i>—Options in this OptionSet are typically used to tell HiLiTE about other, named OptionSets that it needs to look for. Currently used only to name the test harness to be used.</p> <p><i>TestGenLimits</i>—this OptionSet controls search parameters and timeout values during the Simulink data-flow level (model-level or) test generation search performed by HiLiTE.</p> <p><i>TestGenDirectives</i>—lets you control the kinds of tests to be generated and the test generation coverage analysis to be performed as per the desired DO-178B objectives.</p> <p><i>StateflowOptions</i>— includes Option keys for fine tuning the test generation of Stateflow models.</p>	<pre> &lt;Command name="TestGen"&gt;   &lt;OptionSet name="TestGenLimits"&gt;     &lt;Option       key="MaxRecursionLevels"&gt;360&lt;/Option&gt;   &lt;/OptionSet&gt;   &lt;Model name='MyModel1' /&gt; &lt;/Command&gt;  &lt;Command name="TestGen"&gt;   &lt;Model name='MyModel1' &gt;     &lt;OptionSet name="TestGenLimits"&gt;       &lt;Option         key="PropagateConstraints"&gt;No&lt;/Option&gt;     &lt;/OptionSet&gt;   &lt;/Model&gt; &lt;/Command&gt; </pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<p>&lt;OptionSet name="General"&gt; Required subelement &lt;Option&gt; &lt;/Option&gt;</p>	<p>enables major functional components of HiLiTE. Use it to tell HiLiTE about other, named OptionSets that it needs to look for.</p> <p><b>key:</b> required Option attribute that names the option to be used. Available keys for options in the General OptionSet are:</p> <p><i>OutputGenerator</i> specifies the format in which test vectors should be generated. HiLiTE is the default OutputGenerator. HiLiTE will always create a .csv file for the HiLiTE harness. You can specify these additional vector formats:</p> <p><i>CTP</i> specifies that HiLiTE should create output to use in the CTP test harness. If you use this value, you must create a CTP option set in which you specify the character set to use in the output. The example on the right includes a CTP option set. For more information about the CTP option set, refer to Appendix F.</p> <p><i>SATGT</i> (obsolete) specifies the test harness to use for testing AS900 models.</p> <p>Other test harness options will become available in the future.</p> <p><b>Note:</b> the output format you specify becomes the name of a new option set—that is, you must also include an OptionSet with the specified name (&lt;OptionSet name="CTP"&gt;. For more information about CTP option keys, refer to Appendix F.</p> <p><i>DoNotConsiderConditionalSubsystemAsSeperateModel</i> default=no. Use this key to generate tests for certain blocks inside an enabled subsystem where several inputs of the enabled subsystem are tied together or strongly related.</p>	<p>&lt;OptionSet name="General"&gt;   &lt;Option key="OutputGenerator"&gt;CTP&lt;/Option&gt; &lt;/OptionSet&gt;</p> <p>&lt;OptionSet name="General"&gt;   &lt;Option key="OutputGenerator"&gt;CTP&lt;/Option&gt;   &lt;Option key="DoNotConsiderConditionalSubsystemAsSeperateModel"&gt;Yes&lt;/Option&gt; &lt;/OptionSet&gt;</p>
<p>&lt;OptionSet name="TestGenLimits"&gt; Required subelement &lt;Option&gt; &lt;/Option&gt;</p>	<p>controls search parameters and timeout values for Simulink model-level test generation search performed by HiLiTE. This OptionSet must contain one or more &lt;Option&gt; subelements with a single attribute:</p> <p><b>key:</b> required Option attribute that names the option to be used. Choosing yes or increasing a count for these option keys may generate more tests. This increase will improve coverage of requirements, but it will typically take longer for HiLiTE to create tests for the model, giving a tradeoff of execution time vs. coverage. Available keys for options in the TestGenLimits OptionSet are:</p> <p><i>MaxTime:</i> (default=3600) specifies the max number of seconds to use generating tests for a single model. This time limit is cumulative across both Dataflow and State based test generation. By default,</p>	<p>&lt;OptionSet name="TestGenLimits"&gt;   &lt;Option key="MaxTime"&gt;7200&lt;/Option&gt; &lt;/OptionSet&gt;</p>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p>there is no time limit.</p> <p><i>PropagateConstraints</i>: (default=Yes) Specifying <b>Yes</b> instructs HiLiTE to perform a more thorough search by propagating dynamic constraints thru the model graph. This often results in more test vectors to be generated, but may take more time for complex models. If <b>No</b> is specified then constraints are not propagated.</p> <p><i>MaxAlternativeCount</i> In certain models with Data Compare blocks and several mux/demux constructions, some test cases are not generated because there are several alternative combinations to try that exceed the default limits. In these situations, use this option.</p> <p><i>Future: MaxRecursionLevels</i>: (default=200) Specifying a higher value instructs HiLiTE to use more recursion contexts in the search.</p> <p><i>Future: MaxIterations</i>: (default=4000) Specifying a higher value instructs HiLiTE to use more points of search iteration (expansion) for a particular test case step.</p>	<pre>&lt;OptionSet name="TestGenLimits"&gt;   &lt;Option key=     "PropagateConstraints"&gt;No&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="TestGenLimits"&gt;   &lt;Option key=     "MaxAlternativeCount"&gt;1000&lt;/Option&gt;   &lt;!-- 2 hour max per model --&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="TestGenLimits"&gt;   &lt;Option key=     "MaxRecursionLevels"&gt;360&lt;/Option&gt;   &lt;Option key="MaxIterations"&gt;10000&lt;/Option&gt; &lt;/OptionSet&gt;</pre>
<pre>&lt;OptionSet name=   "TestGenDirectives"   Required subelement   &lt;Option&gt;   &lt;/Option&gt;</pre>	<p>controls the kinds of tests to be generated and the test generation coverage analysis for desired DO-178B objectives. This OptionSet must contain one or more &lt;Option&gt; subelements with a single attribute:</p> <p><b>key</b>: required Option attribute that names the option to be used. Available keys for options in the <i>TestGenDirectives</i> OptionSet are:</p> <p><i>InputsRobustnessCoverage</i>: (default=No) Directs HiLiTE to perform robustness coverage analysis on model input variables and generate additional robustness tests (if needed) for the inputs so that each input is tested for a value within, below and above its normal operating range.</p> <p><i>BlocksRobustnessCoverage</i>: (default=No) Directs HiLiTE to tally robustness requirements coverage for all blocks for which robustness requirements have been captured. For each robustness requirement specified for a block, that requirement is considered "covered" if at least one generated test case maps to that requirement.</p> <p><i>InterfaceCoverageForExtFunctions</i>: (default=No) <b>Yes</b> directs HiLiTE to compute only interface coverage (instead of full functional requirements coverage) for blocks that are implemented as separate library functions (e.g. S-Functions). <b>No</b> implies full</p>	<pre>&lt;OptionSet name="TestGenDirectives"&gt;   &lt;Option     key="InputsRobustnessCoverage"&gt;No   &lt;/Option&gt;   &lt;Option     key="BlocksRobustnessCoverage"&gt;Yes   &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="TestGenDirectives"&gt;   &lt;Option     key="InputsRobustnessCoverage"&gt; Yes   &lt;/Option&gt;   &lt;Option     key="InterfaceCoverageForExtFunctions"&gt;</pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p>functional requirements coverage analysis is performed for all blocks. Note that this option doesn't apply to blocks that are not implemented as separate library functions. Interface coverage, when computed for a block, is in lieu of functional requirements coverage for that block. Interface coverage implies the following:</p> <ul style="list-style-type: none"> <li>▪ Each input and output port of the block should be used in at least one test vector generated.</li> <li>▪ If any Interface Requirements are specified for the block, at least one test case generated should map to each such requirement.</li> </ul> <p><i>MakeStimAndObsPointSuggestions</i> (default=off). On directs HiLiTE to suggest observation points and stimulation blocks. The suggestions are now made in modelname_Summary.xml file and the HiLiTE console log.</p> <p><i>UseAlternateTemplateIfPresent</i>. When this option is specified, HiLiTE uses the template of the specified family for a particular block in case a template for the family is provided for that block in the Template Manager. If no family-specific template is provided for a block then HiLiTE uses the Standard template for that block. The name of desired template family is provided as option values.</p>	<pre> Yes &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="TestGenDirectives"&gt;   &lt;Option     key="MakeStimAndObsPointSuggestions"&gt;     On   &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="TestGenDirectives"&gt;   &lt;Option     key="UseAlternateTemplateIfPresent"&gt;     CCC   &lt;/Option&gt; &lt;/OptionSet&gt; </pre>
<pre> &lt;OptionSet name=   "StateflowOptions"   Required subelement   &lt;Option&gt;   &lt;/Option&gt; </pre>	<p>fine tunes StateFlow model test generation. This OptionSet must contain one or more &lt;Option&gt; subelements with a single attribute:</p> <p><i>key</i>: required Option attribute that names the option to be used.</p> <p>Available keys for options in the StateFlowOptions OptionSet are:</p> <p><i>BackwardsSearch</i>: (default=On) enables or disables the backwards search that can find difficult test cases but may also take a lot of time and/or memory.</p> <p><i>MaxDepth</i>: (default=5) specifies the maximum depth of the bounded depth-first backwards search. A value of 5 works for most cases, but if a transition is enabled only after a loop around a series of more than five states, the backwards search may not find this unless MaxDepth is set appropriately.</p> <p><i>MaxTimerIterations</i>: (default=2000) limits the maximum number of time steps in which HiLiTE looks for solutions to timer patterns in the backwards search. This is often a bottleneck for a number of models, because timer patterns that require a very large number of iterations (e.g., 9000) can result in stack overflows. If a timer</p>	<pre> &lt;OptionSet name="StateFlowOptions"&gt;   &lt;Option key="BackwardsSearch"&gt;Off&lt;/Option&gt;   &lt;Option key="MaxDepth"&gt;25&lt;/Option&gt;   &lt;Option key=     "MaxTimerIterations"&gt;5000&lt;/Option&gt;   &lt;Option key=     "MaxExpressionLeafCount"&gt;200   &lt;/Option&gt;   &lt;Option key=     "LargeStateFlowModel"&gt;On&lt;/Option&gt;   &lt;Option key=     "NumTestsPerTPI"&gt;100&lt;/Option&gt;   &lt;Option key=     "ChartInputVectorFiles"&gt;     file_name1     file_name2     directory_name3 </pre>



Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p>pattern is detected that requires more iterations than is allowed by <code>MaxTimerIterations</code>, then a message suggesting the larger value will show up in the <code>modelname_Summary.xml</code> file.</p> <p><i>MaxExpressionLeafCount</i>: (default=10) sets the size of expressions that are eligible for advanced expression simplification. The default value is ten. For some models, increasing this value will significantly decrease the time required to generate tests. However for other models, increasing this value will significantly increase run time. Values between 10 and 500 are recommended.</p> <p><i>LargeStateFlowModel</i>: (default=Off) enables or disables the partitioning of the TPI output into smaller files.</p> <p><i>NumTestsPerTPI</i>: (default=20) specifies the number of test vectors to put in each TPI file when <i>LargeStateFlowModel</i> is enabled.</p> <p><i>ChartInputVectorFiles</i>: (default=empty) lists csv files relative to <code>ProjectPath</code> that should be imported by HiLiTE to provide additional starting points for generating StateFlow tests.</p> <p><i>SetInputPortValue</i>: (default=none) specifies the name of a new OptionSet. This OptionSet may have any name that does not appear in this table. The new OptionSet should contain one or more Options that associate one or more StateFlow chart input variables to specific values. Each Option should have a key that is the fully qualified input variable name and a value that is a double or integer.</p> <p><i>SkipTransReachability</i>: (default=none) specifies the name of a new OptionSet. This OptionSet may have any name that does not appear in this table. The new OptionSet should contain one or more Options, each with a key that is the fully qualified name of a transition source and a value that is the fully qualified name of a transition destination (where the tail name of a junction is "Junction"). All transitions between the source/destination pairs in this OptionSet will be skipped over during backwards search. That is, reachability tests that have not already been generated will not be attempted.</p> <p><i>StateflowTimeoutDuration</i>: (default=-1) specifies a time, in seconds, after which HiLiTE will stop looking to generate new tests. A value of -1 means no timeout</p> <p><i>UseHardRangeConstraints</i>: (default=On) Hard range constraints arise</p>	<pre> &lt;/Option&gt; &lt;Option key=   "StateflowTimeoutDuration"&gt;3600&lt;/Option&gt; &lt;Option key=   "UseHardRangeConstraints"&gt;Off&lt;/Option&gt; &lt;/OptionSet&gt; </pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	solely due to model construction; they denote the feasible input values to the StateFlow chart derived from the data type or from upstream constants or range limiters. This options specifies that HiLiTE should attempt to generate tests by only constraining the ranges of chart input/output values to within these hard constraints that represent values feasible in the model. When this option is Off, HiLiTE generates tests by further constraining the ranges of StateFlow chart inputs to the operational ranges propagated by the upstream blocks based upon operational ranges supplied by the user for the model inputs. <b>Caution</b> Users are advised not to turn this option Off since that will unnecessarily suppress feasible test cases and reduce coverage.	
<OptionSet name= <b>"StateflowDebugVectorsOption"</b> Required sub element <Option> </Option>	fine tunes the test generation of flow debug vectors files. This OptionSet must contain one or more <Option> subelements with a single attribute. <b>key:</b> required Option attribute that names the option to be used. Available keys for options in the <i>StateflowDebugVectorsOption</i> OptionSet are: <i>WriteStateflowDebugVectors</i> : (default=On) specifies whether or not to write out Stateflow debug vectors.	<OptionSet name= <b>"StateflowDebugVectorsOption"</b> > <Option key= <b>"WriteStateflowDebugVectors"</b> > Off</Option> </OptionSet>
<OptionSet name= <b>"SignalBoundaries"</b> Required subelement <Option> </Option>	specifies the boundaries of block types that are visible as global variables for use as observation points in test vectors. It also specifies that certain major blocks (such as a Stateflow chart) can be tested in isolation from the model since they are implemented as a separated function in the generated code. Specify the SignalBoundaries options according to the code-generation options. The default values for the option keys correspond to default options for code generation using HAM & RTW. This OptionSet must contain one or more <Option> subelements with a single attribute: <b>key:</b> required Option attribute that names the option to be used. Available keys for options in the <i>SignalBoundaries</i> OptionSet are: <i>BlockTypeTestPoints</i> . (default=On) implies that internal Block level	<OptionSet name= <b>"SignalBoundaries"</b> > <Option key= <b>"BlockTypeTestPoints"</b> > Off</Option> <Option key= <b>"ModelSpecificTestPoints"</b> > On </Option> </OptionSet>  <OptionSet name= <b>"SignalBoundaries"</b> > <Option key= <b>"BlockIOVariables"</b> > On</Option> </OptionSet>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p>(e.g. inside HAM block) test points are present in the model's code as externally visible global variables, so HiLiTE can use these variables as observation points in the test vectors it generates. Setting this option key to Off instructs HiLiTE not to use internal block-level test points as observation points in test vectors.</p> <p><i>ModelSpecificTestPoints</i>. (default=On) implies that model-level test points ("lollipops") are present in the model's code as externally visible global variables so HiLiTE can use them as observation points in test vectors. Off instructs HiLiTE not to use model-level test points in test vectors.</p> <p><i>BlockIOVariables</i> (default=On) implies that I/O variables around certain types of blocks are present in code and can be used by HiLiTE as observation points in test vectors, irrespective of the other TestPoint options. Off instructs HiLiTE not to assume that I/O variables around certain types of blocks are present in code; other TestPoint options are still applicable.</p> <p><i>IsolateBlocks</i> (default=Off) tells HiLiTE to not generate isolated (standalone) tests for an isolatable block; they are treated same as any other block in the model and model-level (normal) tests are generated for these blocks. On tells HiLiTE to generate isolated tests for each isolatable block in the model in a separate tpi file to be executed separately.</p> <p>(Notes: On value implies On value of <i>BlockIOVariables</i> option. HiLiTE will also generate model-level (normal) tests for these blocks if option <i>TestIsolatedBlocksAtModelLevel</i> is On.)</p> <p><i>TestIsolatedBlocksAtModelLevel</i>. only applicable if option <i>IsolateBlocks</i> is On. It denotes if model-level (normal) tests are to be generated for isolatable blocks in addition to the isolated tests. (Default=Off) denotes that model-level tests should not be generated for isolated blocks. On denotes that model-level test should also be generated for isolated blocks, in addition to the isolated tests. This option can increase test generation time considerably for complex models that have feedback loops among Stateflow charts.</p> <p><i>AllowOverrideOfUnitDelayPastValue</i> improves test generation coverage of models containing complex feedback loops. Default =Off: HiLiTE does not use unit delay overrides. On directs HiLiTE to</p>	<pre> &lt;OptionSet name="SignalBoundaries"&gt;   &lt;Option key="IsolateBlocks"&gt;On&lt;/Option&gt;   &lt;Option     key="BlockIOVariables "&gt;On&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="SignalBoundaries"&gt;   &lt;Option     key="AllowOverrideOfUnitDelayPastValue"&gt;     On&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="SignalBoundaries"&gt;   &lt;Option     key="AllowStimulationBlockOverride"&gt;     On&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="SignalBoundaries"&gt;    &lt;Option key="BlockTypeTestPoints"&gt;     On&lt;/Option&gt;   &lt;Option key="IsolateBlocks"&gt;On &lt;/Option&gt;    &lt;Option key=     "AllowOverrideOfUnitDelayPastValue"&gt;     On&lt;/Option&gt;   &lt;Option key=     "AllowStimulationBlockOverride"&gt;On   &lt;/Option&gt;    &lt;Option key="StimPoints"&gt;sum.O1&lt;/Option&gt;    &lt;Option key="ObservationPoints"&gt;     And2.O1&lt;/Option&gt; &lt;/OptionSet&gt; </pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p>automatically override the past values of all Unit Delay and Frame Delay blocks in the model, as needed.</p> <p><i>AllowStimulationBlockOverride</i> (default=Off) HiLiTE does not allow stimulation block override. On directs HiLiTE to automatically override the signal's upstream value using new HAM stimulation blocks. Note that HiLiTE first attempts to generate each test case without using any overrides of the stimulation blocks. Only if that is not successful, one or more override inputs are used for that test case.</p> <p><i>AllowTestPointBlockOverride</i> (default=off): if enabled, this key enables the Test Point block's (Test Point (HW)) override capability. Used by Epic (PC12) program.</p> <p><i>IgnoreStimBlocksWithPrefixInName</i>: if this key is included, HiLiTE will ignore Stim Numeric or Stim Boolean blocks in a model if the block name starts with a user-specified prefix.</p> <p><i>StimPoints</i> tells HiLiTE to assume a Stimulation Block at a specified point even though the model doesn't include one. This allows quick experimentation without modifying the model.</p> <p><i>ObservationPoints</i> tells HiLiTE to assume an observation point at a specified point even though the model doesn't include one. This allows quick experimentation without modifying the model.</p> <p><b>Note:</b> If you use the TestGenDirectives OptionSet with the Option key <i>MakeStimAndObsPointSuggestions</i>, HiLiTE will include suggested stimulation and observation points in the <i>modelName_Summary.xml</i> file. For more information, see the TestGenDirectives description earlier in this table.</p>	<pre>&lt;OptionSet name="SignalBoundaries"&gt;   &lt;Option key="AllowTestPointBlockOverride"&gt;     On&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="SignalBoundaries"&gt;   &lt;Option key="AllowStimulationBlockOverride"&gt;     On&lt;/Option&gt;   &lt;Option     key="IgnoreStimBlocksWithPrefixInName"&gt;     ST_&lt;/Option&gt; &lt;/OptionSet&gt;</pre>
<pre>&lt;OptionSet name=   "DataTypeAndRange"   Required subelement   &lt;Option&gt; &lt;/Option&gt;</pre>	<p>lets you override the real-data type specification in the model and consider all real numbers as single precision (32-bit) floating points. This OptionSet must contain one or more &lt;Option&gt; subelements with a single attribute:</p> <p><b>key:</b> required Option attribute that names the option to be used. Available keys for options in the <i>DataTypeAndRange</i> OptionSet are:</p> <p><i>AllRealDataTypesAre32Bits</i> Default is Off. Specifying <b>On</b> directs HiLiTE to override real datatype specification in the model and interpret all real data types (for all variables) as 32-bit floating point representation.</p>	<pre>&lt;OptionSet name="DataTypeAndRange"&gt;   &lt;Option key="AllRealDataTypesAre32Bits"&gt;On &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="DataTypeAndRange"&gt;   &lt;Option key="AllRealDataTypesAre32Bits"&gt;On &lt;/Option&gt;   &lt;Option     key="TargetDataTypeForAllReals"&gt;real_T   &lt;/Option&gt;</pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p><i>TargetDataTypeForAllReals</i> allows specification of a data type name (e.g., a specific typedef in a target header file) for all real data types.</p> <p><i>AllowFullRangeOf32BitReal</i> Default is True. <b>True</b> directs HiLiTE to use the full range [-3.4e38, 3.4e38] of the single-precision (32-bits) float data type for testing for a variable, if the data type of that variable is 32-bit real. If <b>False</b> then then a maximum range of [-2E+21, 2E+21] is used.</p>	<pre> &lt;/OptionSet&gt; &lt;OptionSet name="DataTypeAndRange"&gt;   &lt;Option key="AllRealDataTypesAre32Bits"&gt;On &lt;/Option&gt;   &lt;Option key=     "TargetDataTypeForAllReals"&gt;real_T   &lt;/Option&gt;   &lt;Option     key="AllowFullRangeOf32BitReal"&gt;False   &lt;/Option&gt; &lt;/OptionSet&gt; </pre>
<p>&lt;OptionSet name=</p> <p><b>"DisableTestPoints Explicitly"</b></p> <p>Required subelement</p> <p>&lt;Option&gt; &lt;/Option&gt;</p>	<p>Disables a test point when HiLiTE doesn't resolve them properly to the code variable.</p> <p><b>key:</b> required Option attribute that names the test point to be disabled. Default value is <b>False</b>. <b>True</b> directs HiLiTE to disable the test point named as the key.</p>	<pre> &lt;OptionSet name="DisableTestPointsExplicitly"&gt;   &lt;Option key=     "PRIMARY_SOV_DRIVER_TEST_Simplified/     signalConvert6.O1"&gt;True&lt;/Option&gt; &lt;/OptionSet&gt; </pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<p>&lt;OptionSet name= <b>"UseExplicitPortVariableNames"</b> Required subelement &lt;Option&gt; &lt;/Option&gt;</p>	<p>specifies test point variables that HiLiTE couldn't parse for a particular output port. This OptionSet must contain one &lt;Option&gt; subelement with a single <b>key=</b> attribute: the <i>fully qualified name</i> (with proper hierarchy ) of the output port for which HiLiTE is trying to find a test point variable.</p> <p>This option improves test point variable parsing in complex scenarios when signal names occur after the state chart that goes to terminator or other blocks. In this type of scenario, MATLAB generates code in a different way. When there is a test point and the chart output goes to a block that has different signal name, MATLAB gives priority to the signal name and the signal name appears in the code. When the output goes to a terminator, MATLAB gives priority to the chart output name. This option resolves the issues of matching test points by letting you specify it explicitly.</p>	<pre>&lt;OptionSet name=   "UseExplicitPortVariableNames"&gt;   &lt;Option key=     "Chart_To_Terminator/NumState.07"&gt;     Chart_To_Terminator_B.SF_Out7&lt;/Option&gt; &lt;/OptionSet&gt;</pre>
<p><b>&lt;TestOutput&gt;</b> subelements  <b>&lt;User /&gt;</b>  <b>&lt;Root /&gt;</b>  <b>&lt;Args /&gt;</b>  <b>&lt;Version /&gt;</b>  <b>&lt;Category /&gt;</b></p>	<p>use these subelements to make the output path unique for each run that uses the same model. These elements are useful for retesting a model with different inputs or other changes.</p> <p><b>&lt;User /&gt;</b> inserts the name used to log on to the computer.</p> <p><b>&lt;Root /&gt;</b> (example, need explanation) <b>&lt;TestOutput&gt;</b>        <b>&lt;Root&gt;&lt;Args&gt;1&lt;/Args&gt;&lt;/Root&gt;</b>        <b>&lt;Category name="Reports"&gt;Reports&lt;/Category&gt;</b>        <b>&lt;Category name="Vectors"&gt;Vectors&lt;/Category&gt;</b>  <b>&lt;/TestOutput&gt;</b></p> <p><b>&lt;Args /&gt;</b> tells HiLiTE to insert the information from an argument on the command line. Arg 1 is the first item on the command line after the name of the HiLiTE file.</p> <p><b>&lt;Version /&gt;</b> tells HiLiTE to include its current version number in the path name.</p> <p><b>&lt;Category /&gt;</b> will create separate paths for the vector and report files based on their categories (Reports or Vectors)</p>	<pre>&lt;TestOutput root="."&gt;   &lt;Category name="Reports"&gt;     <b>model10_&lt;Date fmt="m_d_yy"/&gt;&lt;Time</b> <b>fmt="hh:mm:ss"/&gt;</b>   &lt;/Category&gt;   &lt;Category name="Vectors"&gt;     TestCases&lt;/Category&gt; &lt;/TestOutput&gt;</pre> <p>Using the Date and Time elements specified above, the test reports will be placed in a folder similar to:  <b>..model10_1_3_0712:15:06</b></p> <pre>&lt;TestOutput root="."&gt;   &lt;Category name="Reports"&gt;     <b>&lt;Args&gt;1&lt;/Args&gt;</b>   &lt;/Category&gt;   &lt;Category name="Vectors"&gt;     <b>&lt;Args&gt;2&lt;/Args&gt;&lt;/Category&gt;</b> &lt;/TestOutput&gt;</pre> <p>Using the &lt;Args&gt; elements as shown above, a command line of: <i>hilite example.hilite myreports testcases</i>, will output the test reports to <b>..myreports</b> and the vectors will be output to <b>..testcases</b></p>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<p>&lt;OptionSet name="Source"&gt;</p> <p>Required subelement</p> <p>&lt;Option&gt; &lt;/Option&gt;</p> <p><i>Required for Test Generation</i></p>	<p>specifies certain characteristics of the source code generated from the model, so that HiLiTE can find proper references to global structures and variables to be used as test vector inputs and observation points. This OptionSet must contain one or more &lt;Option&gt; subelements, each having a single attribute:</p> <p><b>key:</b> required Option attribute that names the option to be used. Available keys for options in the <i>Source</i> OptionSet are:</p> <p><i>Declarations</i> denotes name of a header (.h) file containing variable declarations for use in test vectors. The value can be a <i>direct</i> name or a <i>name template</i> depending on the HiLiTE file location.</p> <p>An example of a <i>direct</i> name is <i>my_model.h</i></p> <p>A name template is a string that contains the keyword substring <b>{model}</b>, each occurrence of which is substitute with the name of the model (minus .mdl extension). E.g., <i>{model}.h</i> gets translated to <i>yourmodel.h</i> for the model <i>yourmodel.mdl</i>.</p> <p>The name may be relative or fully qualified. The name (either direct or generated from the name template) is then used to locate the source code file that includes declarations.</p> <p><i>Format</i> indicates the general source code format of the file named in the Declarations option. This keyword is used by HiLiTE in parsing the file in the appropriate style. <b>HAMRTW</b> is currently the only valid value.</p> <p><i>InPrefix</i>, <i>OutPrefix</i> (optional) indicate the naming prefix used for any external inputs or outputs (respectively) to the model. This can be used to prefix a structure name or a simple string prefix. It can be either "concrete" or a template.</p> <p><i>R2007a</i>: (Default=false) indicates which MATLAB release generated the source code for the model. The default is MATLAB R14. A value of true indicates that the source is R2007a (HAM6), which differs from the MATLAB R14 (HAM5) generated code in the names of the visible variables, structures, and functions. HiLiTE must resolve the references to these entities so that the generated test vectors can be applied to the model's code.</p> <p><i>HAMR12</i> (Default=false): indicates which MATLAB release generated the source code for the model. The default is MATLAB R14</p>	<pre> &lt;OptionSet name="Source"&gt;   &lt;Option     key="Declarations"&gt;{model}.h&lt;/Option&gt;   &lt;Option key="Format"&gt;HAMRTW&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;Option key="Declarations"&gt;   MyFolder/{model}.h &lt;/Option&gt;  &lt;OptionSet name="Source"&gt;...   &lt;Option key="InPrefix"&gt;{model}_U.&lt;/Option&gt;   &lt;Option key="OutPrefix"&gt;{model}_Y.&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="Source"&gt;   &lt;Option key=     "Declarations"&gt;{model}.h&lt;/Option&gt;   &lt;Option key="Format"&gt;HAMRTW&lt;/Option&gt;   &lt;Option key="R2007a"&gt;true&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="Source"&gt;   &lt;Option key="Declarations"&gt;     {model}.h&lt;/Option&gt;   &lt;Option key="Format"&gt;HAMRTW&lt;/Option&gt;   &lt;Option key="HAMR12"&gt;true&lt;/Option&gt; &lt;/OptionSet&gt; </pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p>(HAM5). A value of true indicates that the source is MATLAB R12 (HAM3), which differs from the MATLAB R14 in the names of the visible variables, structures, and functions. HiLiTE must resolve the references to these entities so that the generated test vectors can be applied to the model's code.</p>	
<p>&lt;OptionSet name="ZipFilesOption"</p> <p>Required subelement &lt;Option&gt; &lt;/Option&gt;</p>	<p>used to zip HiLiTE output files such as HTML report, TPI file, and StateflowDebug test vector files. For certain large models, these output files can be very large, consuming much disk space. This OptionSet must contain one or more &lt;Option&gt; subelements, each having a single attribute:</p> <p><b>key:</b> required Option attribute that names the option to be used. Available keys for options in the <i>ZipFilesOption</i> OptionSet are:</p> <p><i>ZipReportFile</i> (Default=Off). On zips the HTML Report.</p> <p><i>DeleteReportFileAfterZipping</i>. (Default=On) tells HiLiTE to delete the HTML Report file after zipping. If HiLiTE could not zip the HTML Report file or the <i>ZipReportFile</i> Option is "Off", HiLiTE will not delete the HTML Report file. Off tells HiLiTE not to delete HTML Report file.</p> <p><i>ZipTPIFile</i> (Default=Off) On means that the TPI file will be zipped.</p> <p><i>DeleteTPIFileAfterZipping</i> (Default=On) tells HiLiTE to delete the TPI file after zipping it. If HiLiTE could not zip the TPI file or the <i>ZipTPIFile</i> Option is turned "Off," HiLiTE doesn't delete the TPI file. Off tells HiLiTE not to delete TPI file.</p> <p><i>ZipStateflowDebugVectors</i> (default=On) StateflowDebug Test vectors (CSV files in StateflowDebug directory) will be zipped. StateflowDebug.zip file will be created inside StateflowDebug directory. StateflowDebug.zip contains all the StateflowDebug test vector files; normal (unzipped) files are not created. Off tells HiLiTE not to zip StateflowDebug Test vectors.</p> <p>Obsolete: <i>DeleteStateflowDebugVectorsAfterZipping</i> (default=On) tells HiLiTE to delete the StateflowDebug Test vectors file after zipping the StateflowDebug Test vectors file. If HiLiTE could not zip the StateflowDebug Test vectors file or <i>ZipStateflowDebugVectors</i> Option is turned "Off," the vectors file will not be deleted. Off tells HiLiTE not to delete StateflowDebug Test vectors file.</p>	<pre>&lt;OptionSet name="ZipFilesOption"&gt;   &lt;Option key="ZipReportFile"&gt;On&lt;/Option&gt;   &lt;Option key="ZipTPIFile"&gt;On&lt;/Option&gt; &lt;/OptionSet&gt;</pre>



Element Tag, Attribute Name, and/or OptionSet	Description	Example
<p>&lt;OptionSet name="UseExplicitSUTName"&gt;  Required subelement  &lt;Option&gt; &lt;/Option&gt;  &lt;/OptionSet&gt;</p> <p>(Optional)</p>	<p>explicitly declares the State chart function names in the hilite command file. If State chart function name is not specified, HiLiTE will generate a default name (based on model name) which will be used as software under test (SUT). This OptionSet must contain one or more &lt;Option&gt; subelements, each having a single attribute:</p> <p><b>key:</b> required Option attribute that names the option to be used. Available keys for options in the <i>UseExplicitSUTName</i> OptionSet are:</p> <p><i>StateChartNameWithFullyQualifiedPath</i> Specify the fully qualified path of the State chart in the model. If the State chart is inside the subsystem, you must correctly specify the State chart name. For example, if model has subsystem "Subsystem1" and State chart "StateChart1" is present inside Subsystem1, then Option key will be "Subsystem1/StateChart1".</p> <p>For corresponding State charts, provide the State chart function name as the Option key value. This State chart function name is the SUT, which is part of the TPI header file. The TPI file generated for the State chart will contain the declared State chart function name as SUT.</p>	<pre>&lt;OptionSet name="UseExplicitSUTName"&gt;   &lt;Option key="     StateChartNameWithFullyQualifiedPath"&gt;     StateChartFunctionName &lt;/Option&gt;   &lt;/OptionSet&gt;  &lt;OptionSet name=" UseExplicitSUTName"&gt;   &lt;Option key="     StateChartNameWithFullyQualifiedPath1"&gt;     StateChartFunctionName1 &lt;/Option&gt;   &lt;Option key="     StateChartNameWithFullyQualifiedPath2 "&gt;     StateChartFunctionName2 &lt;/Option&gt; &lt;/OptionSet&gt;</pre>
<OptionSet> name="CTP"	For more information about the CTP option set and keys, refer to Appendix F.	
Additional commands <Command name="ModelAnalysis">	<p>Propagates ranges through the blocks in the models from model inputs to outputs.</p> <p>Also identifies probable defects (for example: divide by zero or untestable conditions) or design errors in the model.</p>	<Command name="ModelAnalysis">
<OptionSet> name="UseExplicitAT3Anchors"	<p>supports user-supplied AT3 anchors in generated TPI files.</p> <p>Required <b>key:</b>  <i>STP_Stateflow_Isolation/StateChart1</i>: the name of a StateFlow chart.</p>	<pre>&lt;OptionSet name="UseExplicitAT3Anchors"&gt;   &lt;Option key="     "CTP_Stateflow_Isolation/StateChart1"&gt;     Name1 &lt;/Option&gt; &lt;/OptionSet&gt;</pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<p>&lt;OptionSet name="ModelAnalysisDirectives"&gt; Required subelement &lt;Option&gt; &lt;/Option&gt;</p>	<p>gives instructions for analyzing a model. This OptionSet must contain one or more &lt;Option&gt; subelements, each having a single attribute:</p> <p><b>key:</b> required Option attribute that names the option to be used. Use one of the following key values:</p> <p><i>ExtensiveModelAnalysis</i> (default=False) use to analyze complex models, especially models that include complex loops. Set to "True" when suggested by the hilite.log file. You will see messages similar to:</p> <pre>[ERROR] ModelName: Data Type and Range propagation failed; test generation aborted. [ERROR] Attention HiLiTE Users!... ModelName Enable following option in HiLiTE command file... &lt;OptionSet name="ModelAnalysisDirectives"&gt;   &lt;Option key="ExtensiveModelAnalysis"&gt;True&lt;/Option&gt; &lt;/OptionSet&gt;</pre> <p><i>GenerateOutputRangeFile:</i> A range file containing model output names and their operational ranges can be generated using this option. By default, the name of the file is OutputRanges_modelname.csv.</p> <p><i>OutputRangeFileName:</i> Use this option key to specify the name of output range file.</p> <p><i>UpdateOutputRangeFileInPlace:</i> Use this to read an intermediate range file that contains the outputs of many models and just update the values of those symbols whose producer is the model being processed by HiLiTE.</p> <p><i>OutputRangeFileInReportsDir:</i> (default = True) set this option key to "True" or "False" to determine whether HiLiTE should place the output range file in the Reports directory. False implies HiLiTE places the range file in the same directory as ProjectPath</p> <p><i>CheckBlockInputRangesAgainstLimits</i> and <i>BlockInputRangeLimitsFileName</i> are used together to specify the limits for certain block types in a CSV file and supply that file to HiLiTE. See example at right.</p>	<pre>&lt;OptionSet name="ModelAnalysisDirectives"&gt;   &lt;Option key="ExtensiveModelAnalysis"&gt;     True&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="ModelAnalysisDirectives"&gt;   &lt;Option key="GenerateOutputRangeFile"&gt;     True&lt;/Option&gt;   &lt;Option key="OutputRangeFileInReportsDir"&gt;     True&lt;/Option&gt;   &lt;Option key="OutputRangeFileName"&gt;     MyOutputRanges.csv&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="ModelAnalysisDirectives"&gt;   &lt;Option key=     "UpdateOutputRangeFileInPlace"&gt;True   &lt;/Option&gt;    &lt;Option key="OutputRangeFileName"&gt;     IntermediateRanges.csv   &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="ModelAnalysisDirectives"&gt;   &lt;Option key="GenerateOutputRangeFile"&gt;     True&lt;/Option&gt;   &lt;Option key=     "OutputRangeFileInReportsDir"&gt;True   &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="ModelAnalysisDirectives"&gt;   &lt;Option     key="CheckBlockInputRangesAgainstLimits"&gt;     True&lt;/Option&gt;   &lt;Option key="BlockInputRangeLimitsFileName"&gt;     BlockInputsRangeLimits.csv&lt;/Option&gt; &lt;/OptionSet&gt;</pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<OptionSet name= <b>"UnitDelayOverrideVariables"</b> Required subelement <Option> </Option>	<p>used by Epic (PC12) program. If the RTW code generated for the models doesn't contain comments (containing tags) in the D_Work structure, then the "UnitDelayOverrideVariables" OptionSet can be used to explicitly specify the D_Work structure variable name. When comments are missing in the D_Work structure, use this option to explicitly specify them in the .hilit file.</p> <p><b>Key</b> → &lt;ModelNameWithHierarchicalPath&gt;/&lt;SimpleDelayBlockName&gt;/SimpleDelay_1.  SimpleDelay_1 is the testpoint variable name.</p> <p><b>Value</b> → &lt;ModelName&gt;_Dwork.&lt;DWork_StructureVariable name&gt;</p>	<pre> &lt;OptionSet name="UnitDelayOverrideVariables"&gt;   &lt;Option key=     "OV_AutoUnitDelay_Scenario/SimpleDelay/     SimpleDelay_1"&gt;OV_AutoUnitDelay_     Scenario_DWork.SimpleDelay_1_DSTATE   &lt;/Option&gt;   &lt;Option key=     "OV_AutoUnitDelay_Scenario/SimpleDelay1     /SimpleDelay_1"&gt;OV_AutoUnitDelay     _Scenario_DWork.h_SimpleDelay_1     _DSTATE&lt;/Option&gt; &lt;/OptionSet&gt; </pre>
<OptionSet name= <b>"TestTemplates"</b> Required subelement <Option> </Option>	<p>used to specify how HiLiTE will generate test case templates for certain blocks that are listed in Appendix I. This OptionSet must contain one or more &lt;Option&gt; subelements, each having a single attribute:</p> <p><b>key:</b> required Option attribute that names the option to be used. Use one of the following key values:</p> <p><i>LookUpTableInterpolationTests</i> (default=True) A value of True instructs HiLiTE to generate interpolation tests for a lookup table block (1-D, 2-D, or 3-D look up table). By default, HiLiTE generates test cases for a limited number of interpolation points in the table.</p> <p><i>LookUpTableAllInterpolationPoints</i>(default = False) If option <i>LookUpTableInterpolationTests</i> is True, set this option key to "True" when you want HiLiTE to generated all possible interpolation points for a lookup table.</p> <p><i>LookUpTableBeyondTests</i>: (default = True) If True, HiLiTE will generate Beyond Boundary tests (based upon block parameter Clip or Extrapolate) for a lookup table block.</p>	<pre> &lt;OptionSet name="TestTemplates"&gt;   &lt;Option key=     "LookUpTableAllInterpolationPoints"&gt;True   &lt;/Option&gt;   &lt;Option key=     "LookUpTableBeyondTests"&gt;True&lt;/Option&gt; &lt;/OptionSet&gt; </pre>

#### 4.3.4 Miscellaneous Advanced HiLiTE Command-File Elements

Table 3 describes miscellaneous command file elements that can be useful to control certain advanced aspects of HiLiTE execution.

**Table 3. Miscellaneous advanced HiLiTE command file elements**

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<b>&lt;HiLiTEMgmt /&gt;</b>	<p>If this optional element is specified, it must be the first element in a HiLiTE command file. Use it to manage the way HiLiTE runs. Currently HiLiTEMgmt includes these attributes:"</p> <p><i>pauseOnCompletion</i> (Default=no) indicates whether you want the program to pause when exiting. HiLiTE will pause after completing its command sequence. This is useful when you want to look at a HiLiTE generated command window and the progress log. <i>no</i> indicates that HiLiTE will not pause. After completing its command sequence, and if HiLiTE opened the command window, it will exit and the Command window will close. 'no' can be useful when you want to run HiLiTE unattended. By allowing HiLiTE to completely exit, you ensure that all resources (such as MATLAB licenses) are released when processing is complete.</p> <p><i>timeCode</i> (Default = no) indicates whether you want timing information to be included in the HiLiTE.log. Possible values are "yes" or "no."</p> <p><i>cleanlinessCheck</i> (Default= yes) indicates HiLiTE should check the model to make sure it is properly constructed and contains only supported blocks before attempting to generate code or test vectors. Possible values are "yes," which tells HiLiTE to do the check, and "no," which tells it not to.</p> <p><i>CreateModelQueriesFile</i> (Default=no) indicates whether or not you want HiLiTE to create an additional file "&lt;modelname&gt;_Matlab.txt". "Yes" will make HiLiTE to run MATLAB normally and load the model from it. It performs test generation or model analysis as it would do normally, but in addition, HiLiTE will also create a file named "&lt;modelname&gt;_Matlab.txt" that contains all the queries/responses that HiLiTE exchanged with MATLAB when loading the model. No means additional file is not generated.</p> <p><i>ReadModelQueriesFile</i> (Default=no) indicates whether or not you want HiLiTE to read the file "&lt;modelname&gt;_Matlab.txt" instead of opening MATLAB. "Yes" will make HiLiTE not to run MATLAB but</p>	<p><b>&lt;HiLiTEMgmt</b> <i>pauseOnCompletion</i>="yes" /&gt;</p> <p><b>&lt;HiLiTEMgmt</b> <i>timeCode</i> ="yes" /&gt;</p> <p><b>&lt;HiLiTEMgmt</b> <i>cleanlinessCheck</i> ="no" /&gt;</p> <p><b>&lt;HiLiTEMgmt</b> <b>CreateModelQueriesFile</b> ="yes" /&gt;</p> <p><b>&lt;HiLiTEMgmt</b> <b>ReadModelQueriesFile</b>="yes" /&gt;</p>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	read the "<modelname>_Matlab.txt" file and load the model from it. It performs test generation or model analysis as it would do normally. No means HiLiTE opens MATLAB and reads the model from it. Using the Yes option, HiLiTE can be used on machine that cannot run MATLAB provided "<modelname>_Matlab.txt has been created earlier by running HiLiTE on a machine with MATLAB using the <i>CreateModelQueriesFile</i> option.	
<b>&lt;Translation /&gt;</b>	Use this optional sub-element of <Column> to convert column values. Translation uses a simple string substitution and also supports regular expressions (reference a document that defines regular expression syntax). Any number of Translation elements can be contained in each Column element. Translation elements have two required attributes: <i>from</i> and <i>to</i> . Either attribute can contain an empty string.	<Column name="utype" col="1"> < <b>Translation</b> from='flag' to='UniversalBoolean'/> </Column>
<Model> sub-elements <b>&lt;Block/&gt;</b> <b>&lt;BlockType/&gt;</b> <b>&lt;TemplateProc&gt;</b>	Use these elements with the TestGen command to select specific blocks to test rather than testing all blocks in a model. All three elements have a single attribute: <b>name</b> <Block /> and<BlockType /> are sub elements of <Model> <TemplateProc /> is a sub element of <BlockType > or <Block/> In the example on the right, the block with the label 'FaultDetectUnlatch1' and ALL blocks with type 'AND2' will be tested. Only the blocks meeting these criteria will have tests generated. For individual blocks or block types, you can optionally specify which template procedures should be used for test generation. In the example, the template procedure with the name '1 1' will be used to generate tests for the blocks of type 'AND2'. Because a template procedure is specified, only the specified template procedure(s) will be used in an attempt to generate tests. If no procedures are specified, then all available template procedures will be used. Note that template conditions already set some limits; these conditions will still be evaluated.	<Command name='TestGen' >  <Model name='YX440d'> < <b>Block name</b> ='FaultDetectUnlatch1' /> < <b>BlockType</b> name='AND2'> < <b>TemplateProc</b> name="1 1" /> </BlockType>  </Model> </Command>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<OptionSet name= <b>"ReportingOptions"</b> Required subelement <Option key= <b>"GenerateHTMLVec torReport"</b> > </Option> <OptionSet>	This OptionSet and option key combination disables the HTML report. This option is useful when running very large models for which HTML reports take a long time to generate. Set the key to Yes or No (Default=Yes)	<OptionSet name=" <b><i>ReportingOptions</i></b> "> <Option key= <b>"GenerateHTMLVectorReport"</b> >No</Option> </OptionSet>

# 5 Generating Test Cases

HiLiTE generates test cases based on MATLAB Simulink and Stateflow models. You can use the test cases with SATGT, CTP, or other test harnesses to test the code based on the same models. The code you test does not have to be generated by HiLiTE.

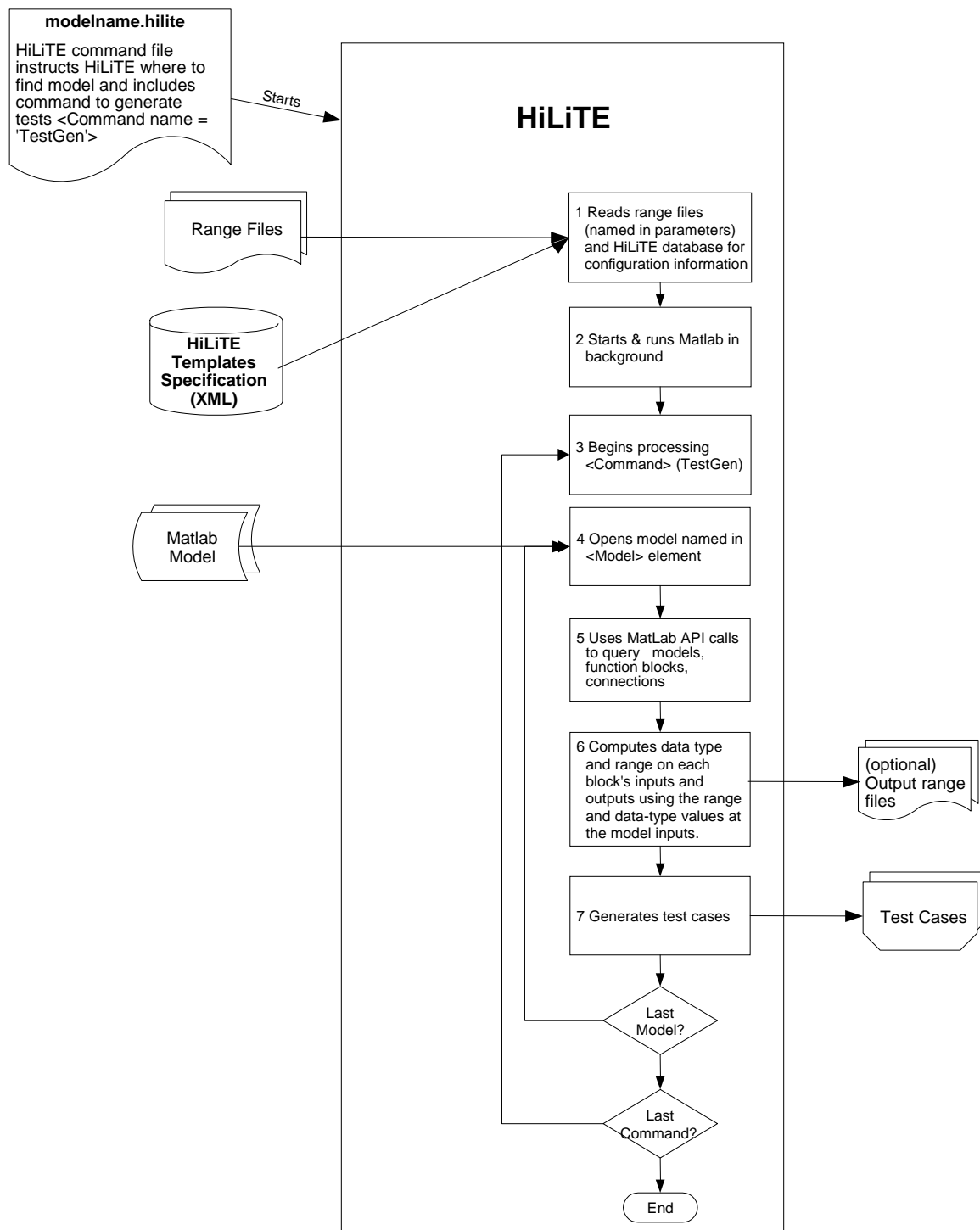
This chapter describes:

- How HiLiTE processes input to create Simulink test cases.
- Examples of command files for generating Simulink test cases.
- How to specify output formats for test vectors.
- How to interpret and use the generated Simulink test cases and accompanying reports.
- How to the specify format (test harness) in which HiLiTE should create test cases.
- Specific test generation examples.
- How to specify tolerances for output values.

For information about generating test cases with Stateflow models, refer to Chapter 6.

## 5.1 HiLiTE Command File Processing

Figure 17 illustrates how HiLiTE interprets information in the command file to process input and generate test cases.



**Figure 17 - How HiLiTE generates test cases**

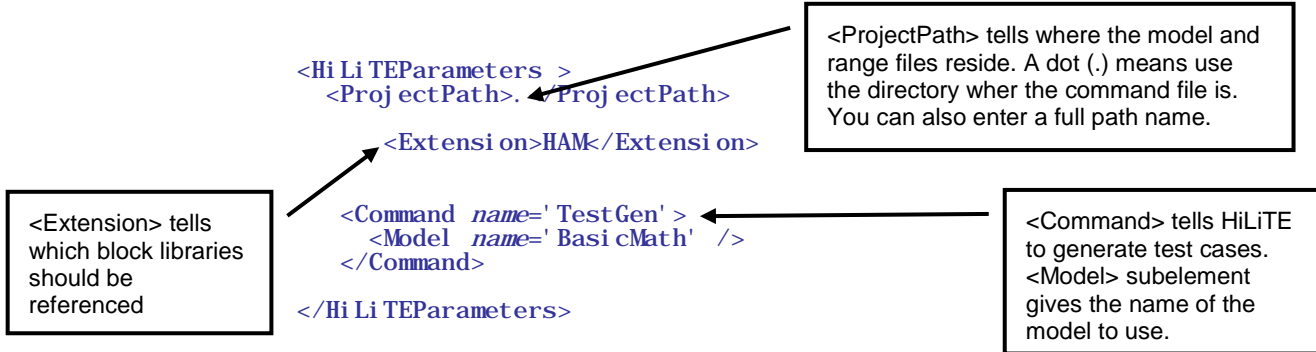
HiLiTE gathers data from all the range files you specify in the command file and then reads in the tool configuration data (block and template specifications). It applies this data to all the models it will process



to generate test cases. (Note: you can modify the tool configuration data, as described in Section 10.5 Managing Configuration of Changes to Blocks and Templates.)

## 5.2 Test Generation Command Files

Figure 18 shows the contents of a simple command file for generating tests from one model.

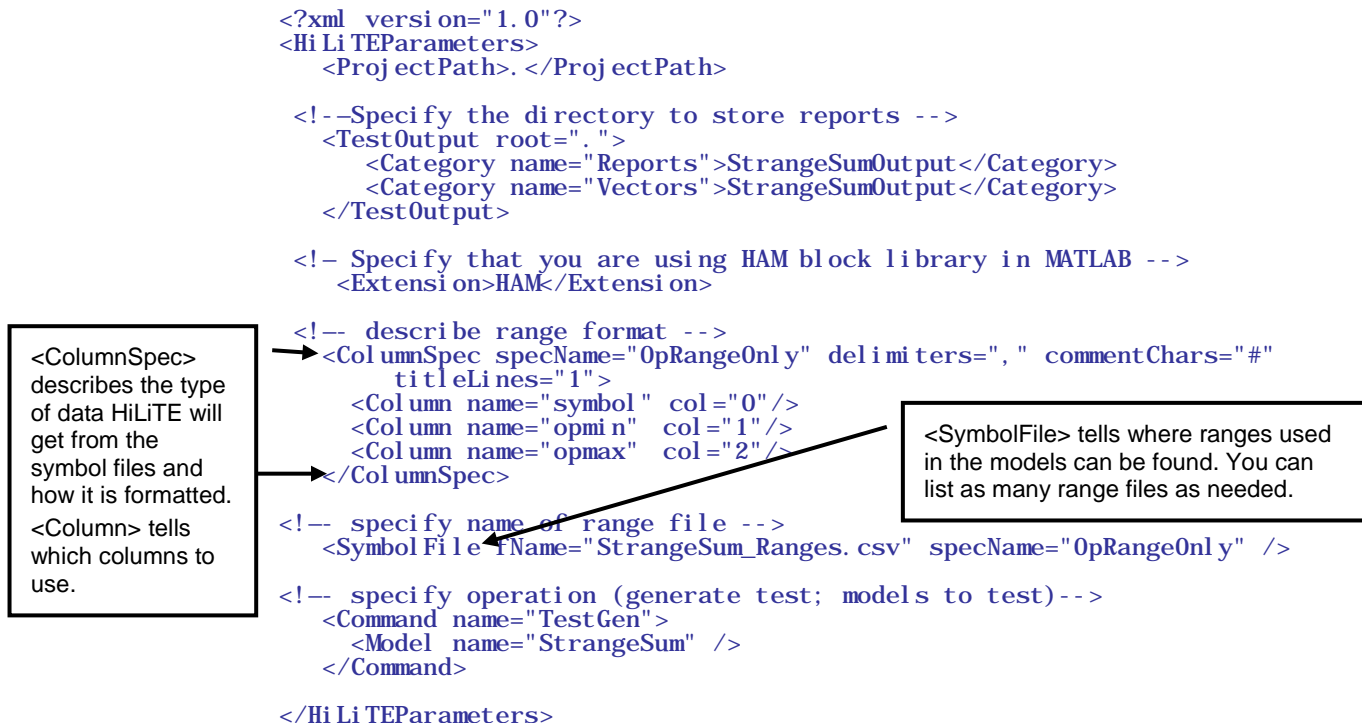


**Figure 18 - Simple test generation command file**

To generate tests in HiLiTE, your command file must include a <Command> element with a *name* attribute describing the command operation (e.g., *TestGen* tells HiLiTE to generate tests for the model).

Optional attributes for the Command tell HiLiTE where to place the reports (ReportPath) and test vectors (TestPath). If you do not include path attributes, test cases (vectors) and reports will be placed in Vectors and Reports subdirectories of the command file root directory.

The Command element must include a <Model> subelement for each model you want to process. Each <Model> element must include the name of the model (without the extension .mdl) for which test vectors will be generated. Figure 19 is a slightly more complex command file that includes <ColumnSpec> and <SymbolFile> elements.



**Figure 19 - Test generation command file with column specification**

### 5.3 Specifying Test Vector Output Formats

By default, HiLiTE generates a .csv file of test vectors formatted for the HiLiTE Test Harness. If you are using a different harness, you must specify the output to generate using an OptionSet element. In Figure 20, we tell HiLiTE to generate test cases in a format (.tpi) for CTP test harness.

```
<?xml version="1.0"?>
<HiLiTEParameters>
  <ProjectPath>./</ProjectPath>

  <!--Specify the directory to store reports -->
  <TestOutput root=".">
    <Category name="Reports">StrangeSumOutput</Category>
    <Category name="Vectors">StrangeSumOutput</Category>
  </TestOutput>

  <!--Specify to HiLiTE the block library being used -->
  <Extension>HAM</Extension>

  <!-- describe range file's format -->
  <ColumnSpec specName="OpRangeOnly" delimiters="," commentChars="#"
    titleLines="1">
    <Column name="symbol" col="0"/>
    <Column name="opmin" col="1"/>
    <Column name="opmax" col="2"/>
  </ColumnSpec>

  <!-- specify name of range file -->
  <SymbolFile fName="StrangeSum_Ranges.csv" specName="OpRangeOnly" />

  <!-- specify operation (generate test, test harness, model) -->
  <Command name="TestGen">
    <OptionSet name="General">
      <Option key="OutputGenerator">CTP</Option>
    </OptionSet>
    <OptionSet name="Source">
      <Option key="Declarations">{model}.h</Option>
    </OptionSet>
    <OptionSet name="CTP">
      <Option key="OutputBy">model</Option>
      <Option key="Partition">QT</Option>
      <Option key="AnchorFormat">CTP_{partition}_{nodename}</Option>
      <Option key="VectorFormat">787_TPI</Option>
      <Option key="FileNameFormat">{partition}_{nodename}_ctp</Option>
    </OptionSet>
    <Model name="StrangeSum" />
  </Command>
</HiLiTEParameters>
```

<OptionSet> can specify several options (see Table 2); in this case we tell HiLiTE to output test cases for the CTP harness. <OptionSet name="CTP"> specifies the format for the output (CTP can use three formats).

**Figure 20 - Test generation command file specifying test case format**

Note: For more information about using the CTP OptionSet, refer to Appendix F.

Note2: This figure illustrates the discussed concepts, the strangesum.hilite file example included in the HiLiTE installation does not include OptionSet elements.

## 5.4 Generating Tests

The command file example in Figure 21 generates test cases for a HAM model.

```
<Hi Li TEParameters>
  <ProjectPath>C:\Hi Li TE\exampl es\Hi Li TE- TG- HAM\MathExampl es</ProjectPath>

  <TestOutput root=". ">
    <Category name="Reports">TestGenOutput</Category>
    <Category name="Vectors">TestGenOutput</Category>
  </TestOutput>

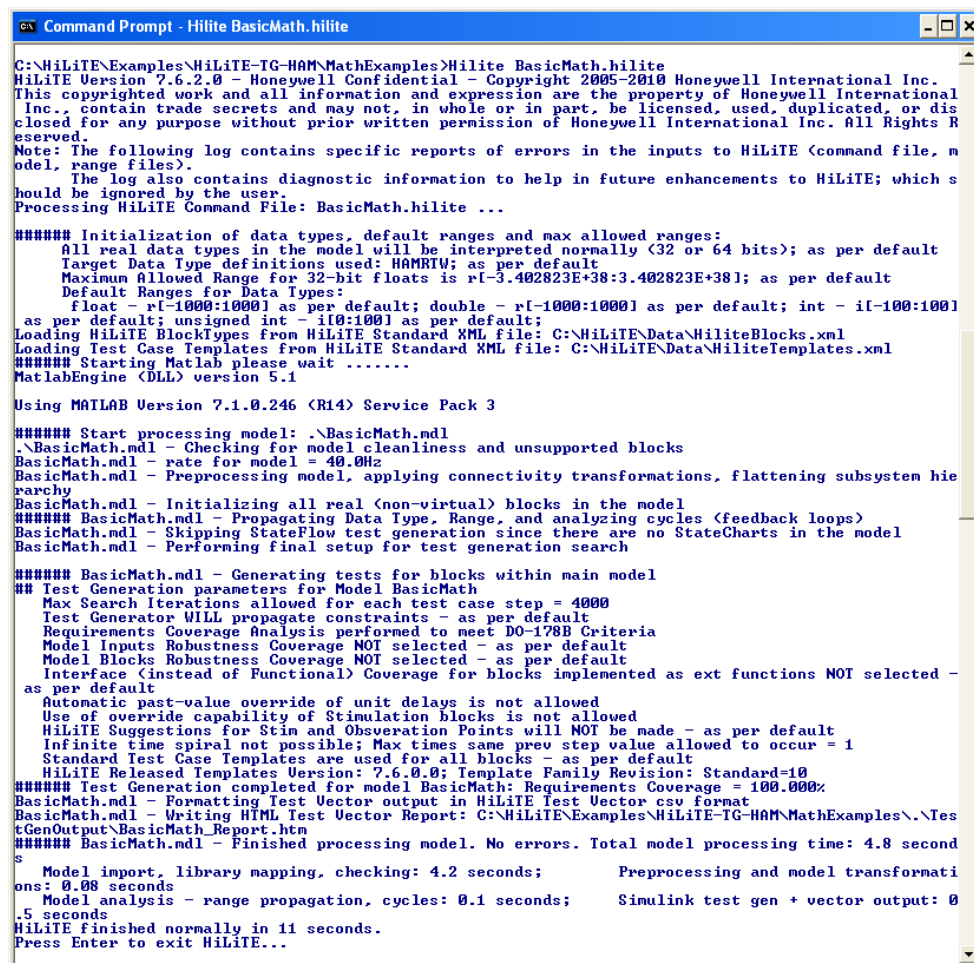
  <Extensi on>HAM</Extensi on>

  <Command name=' TestGen' >
    <Model name=' Basi cMath' />
  </Command>

</Hi Li TEParameters>
```

**Figure 21 - Test generation command file**

This command file is a variation of the that shown in Figure 18. It differs from the previous example in that the file includes specific path references for finding the input (<ProjectPath>) and placing the output (Vectors and Reports in the TestOutput element). The command window information is shown in Figure 22.



```

C:\HiLiTE\Examples\HiLiTE-TG-HAM\MathExamples>Hilite BasicMath.hilite
Hilite Version 7.6.2.0 - Honeywell Confidential - Copyright 2005-2010 Honeywell International Inc.
This copyrighted work and all information and expression are the property of Honeywell International Inc., contain trade secrets and may not, in whole or in part, be licensed, used, duplicated, or disclosed for any purpose without prior written permission of Honeywell International Inc. All Rights Reserved.
Note: The following log contains specific reports of errors in the inputs to Hilite (command file, model, range files).
The log also contains diagnostic information to help in future enhancements to Hilite; which should be ignored by the user.
Processing Hilite Command File: BasicMath.hilite ...

##### Initialization of data types, default ranges and max allowed ranges:
All real data types in the model will be interpreted normally (32 or 64 bits); as per default
Target Data Type definitions used: HAMRTW; as per default
Maximum Allowed Range for 32-bit floats is [-3.402823E+38;3.402823E+38]; as per default
Default Ranges for Data Types:
float - [-1000;1000] as per default; double - [-1000;1000] as per default; int - [-100;100] as per default; unsigned int - [0;100] as per default;
Loading Hilite BlockTypes from Hilite Standard XML file: C:\HiLiTE\Data\HiliteBlocks.xml
Loading Test Case Templates from Hilite Standard XML file: C:\HiLiTE\Data\HiliteTemplates.xml
##### Starting Matlab please wait .....
MatlabEngine (DLL) version 5.1

Using MATLAB Version 7.1.0.246 (R14) Service Pack 3

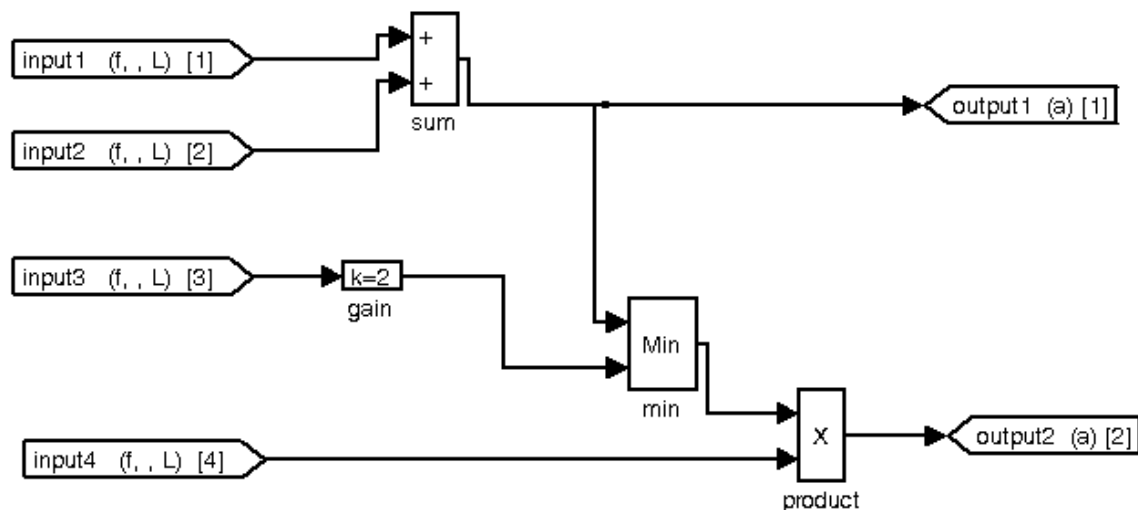
##### Start processing model: .\BasicMath.mdl
.\BasicMath.mdl - Checking for model cleanliness and unsupported blocks
BasicMath.mdl - rate for model = 40.0Hz
BasicMath.mdl - Preprocessing model, applying connectivity transformations, flattening subsystem hierarchy
BasicMath.mdl - Initializing all real (non-virtual) blocks in the model
##### BasicMath.mdl - Propagating Data Type, Range, and analyzing cycles (feedback loops)
BasicMath.mdl - Skipping StateFlow test generation since there are no StateCharts in the model
BasicMath.mdl - Performing final setup for test generation search

##### BasicMath.mdl - Generating tests for blocks within main model
## Test Generation parameters for Model BasicMath
Max Search Iterations allowed for each test case step = 4000
Test Generator WILL propagate constraints - as per default
Requirements Coverage Analysis performed to meet DO-178B Criteria
Model Inputs Robustness Coverage NOT selected - as per default
Model Blocks Robustness Coverage NOT selected - as per default
Interface (instead of Functional) Coverage for blocks implemented as ext functions NOT selected - as per default
Automatic past-value override of unit delays is not allowed
Use of override capability of Stimulation blocks is not allowed
Hilite Suggestions for Stim and Observation Points will NOT be made - as per default
Infinite time spiral not possible; Max times same prev step value allowed to occur = 1
Standard Test Case Templates are used for all blocks - as per default
Hilite Released Templates Version: 7.6.0.0; Template Family Revision: Standard=10
##### Test Generation completed for model BasicMath: Requirements Coverage = 100.000%
BasicMath.mdl - Formatting Test Vector output in Hilite Test Vector csv format
BasicMath.mdl - Writing HTML Test Vector Report: C:\HiLiTE\Examples\HiLiTE-TG-HAM\MathExamples\.\TestGenOutput\BasicMath_Report.htm
##### BasicMath.mdl - Finished processing model. No errors. Total model processing time: 4.8 seconds
Model import, library mapping, checking: 4.2 seconds; Preprocessing and model transformations: 0.08 seconds
Model analysis - range propagation, cycles: 0.1 seconds; Simulink test gen + vector output: 0.5 seconds
Hilite finished normally in 11 seconds.
Press Enter to exit Hilite...
  
```

**Figure 22 - Command execution information for BasicMath.hilite test generation**

## 5 Generating Test Cases

Figure 23 shows the model against which the command file instructions are run and Figure 24 shows the range values and data type for input2 of the model.



**Figure 23 - Simulink model for BasicMath**

The screenshot shows the 'BasicMath/input2' dialog box. The fields are as follows:

Field	Value
Port Name:	input2
Port Number:	2
Dimensions (-1 for dynamic):	-1
Signal Name (see below):	<
(≤: Propagate from upper; ≥: Propagate port name)	
Define as Bus Object:	<input type="checkbox"/>
Data Type:	double/float
Lower Range:	6
Upper Range:	12
Signal Units (manual entry):	
Signal Units (standard entry):	Manual Selection
Software Initial Input:	
DEOS Consumed Type:	Latched

Buttons at the bottom: Help, Revert, Done.

**Figure 24 - Input details showing Data Type and Normal Operating Range**

Because BasicMath.mdl already includes range values for all inputs, it is not necessary to include a range file with the model or to reference one in the command file.

### 5.4.1 Generated Tests and Test Reports

HiLiTE test generation produces four types of test-related files:

- **Test cases** in a format specific to the test harness to be used. HiLiTE produces one test case file for each model. It names the file for the model with a harness-appropriate extension, for example `<ModelName>_Vectors.csv` or `<PartitionName>_<ModelName>_ctp.tpi`. This file includes actual test cases that will be run in the test harness against code. While this file may have a fairly simple format, it is not intended to be human-readable. If you request CTP test output, HiLiTE generates a text file that identifies the test case templates used in each of the vectors (.tpi) files. Externally visible variables associated with inputs, outputs, and observation points appear explicitly in the test vector files, but constants used in a model do not since they are hardcoded in the model's code and cannot be manipulated from a test harness. If a constant feeds an input of a block being tested, the vector file will not show any variable set for that input. Also see \*Note below for more information.
- **Test case Report** in html format. Reports are named for the model, e.g. `<ModelName>_Report.htm`. Figure 25 shows part of the test report for the BasicMath model shown in Figure 23. Note that this is an “unqualified” output of HiLiTE, intended for user guidance only. See \*Note below for more information.
- **Status Report** in html format (partial report shown in Figure 26) that tell which procedures could be generated and which could not. You must manually the procedures that could not be generated. Check list file names are in the format `<ModelName>_StatusReport.html`. "Completeness" as described a status report refers to completeness as requested in the command file—all the template procedures you requested with valid conditions successfully created test cases. It does NOT mean that ALL the available template procedures resulted in test cases. See \*Note below for more information.
- **Model Summary** in xml format that summarizes the HiLiTE results, showing the models used, success of the Simulink and Stateflow test generation, success of the model analysis, and error status. The file name format is `<ModelName>_Summary.xml`. Note that this is an “unqualified” output of HiLiTE, intended for user guidance only.
- **Writer Report** in text format that lists for each local and output variable the actions that write to that variable only for Stateflow charts. This is useful for understanding uninitialized value error messages. Reports are named for the statechart, e.g., `<chartName>_WriterReport.txt`. Note that this is an “unqualified” output of HiLiTE, intended for user guidance only.

\*Note that when the the LargeStateflowModel option is used, HiLiTE splits the Simulink and Stateflow level test cases into multiple csv or tpi files by appending 000<Num> at the end of the file name. For example: `<ModelName>_Vectors0001.csv`(or `<PartitionName>_<ModelName>_ctp0001.tpi`), `<ModelName>_Vectors0002.csv`(or `<PartitionName>_<ModelName>_ctp0001.tpi`) and so on... The Reports are split into multiple files in the same manner, e.g., `<ModelName>_Report0001.htm` `<ModelName>_Report0002.htm`, and so on.

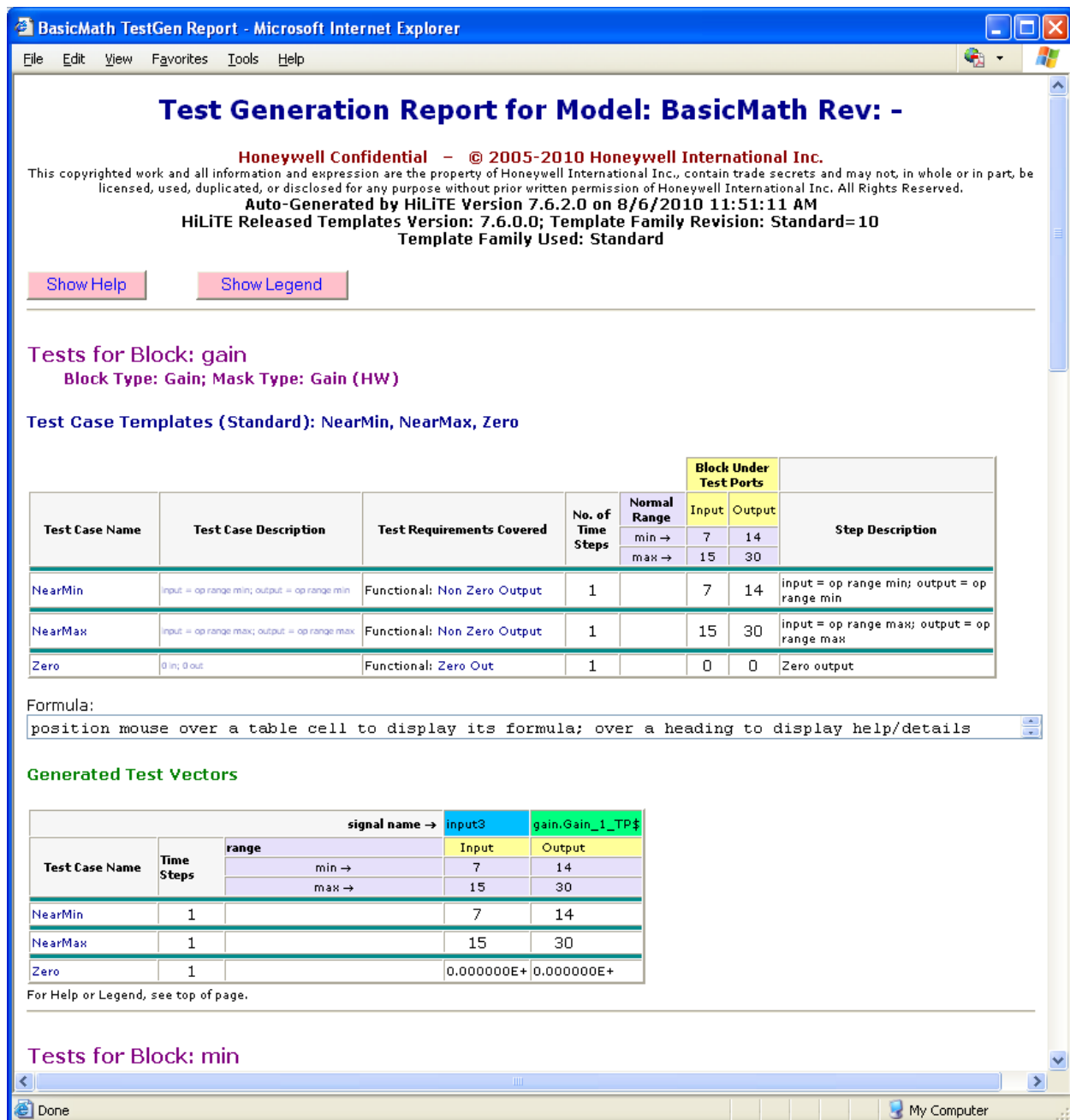


Figure 25 - Test report for BasicMath

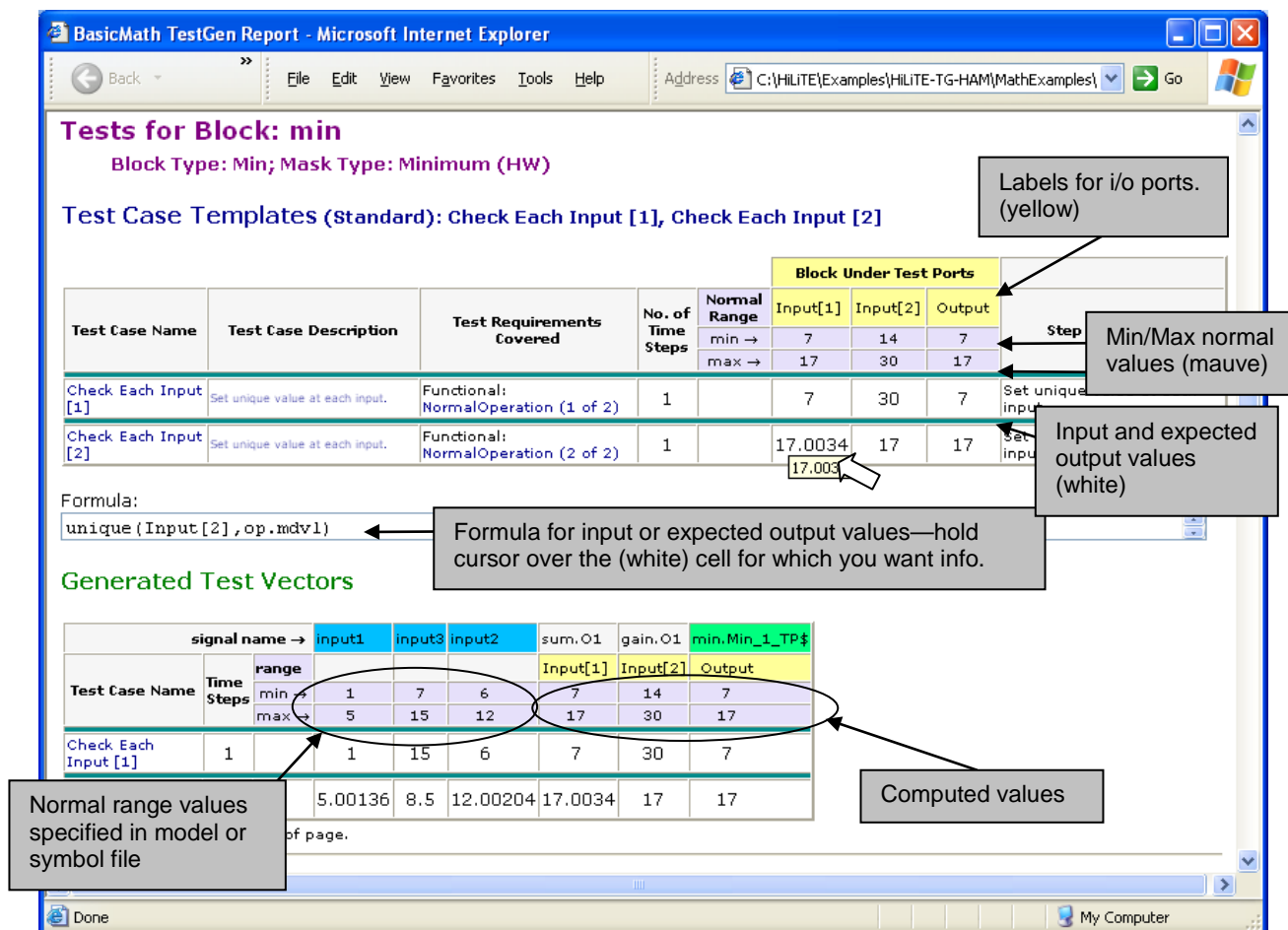


Figure 26 - Status report for BasicMath.mdl

### 5.4.2 Browsing Test Reports– Test Case Templates and Generated Vectors

A test report (*model\_Report.htm*) is a readable version of generated tests. It includes color-coded tables of the templates and the test cases they generated. You can read this file in any browser. (Note that these reports are “unqualified” and cannot be used to claim credit for DO-178B objectives. The content and format of the fields may change with HiLiTE releases.)

The report shows the tests in alphabetical order of the blocks. For example, the first item in the report shown in Figure 25 for the model in Figure 23 is **Block: Gain**, while the first block in the diagram is **Sum1**. For each block, the report includes a description of the test template and a table of the test cases produced with the template. Figure 27 illustrates template and test vector tables for the **Min** block in the same model.



**Figure 27 - Template and test case for min block**

For each block, the test report includes two tables: a test case template (labeled “Test Case Templates”) and a test case (labeled “Generated Test Vectors”). The green labels in each table indicate the input and output ports for the block. In the template procedure, mauve rows are the minimum and maximum allowed values for the ports. Additional blue rows in the test case label the input and output according to the model diagram. White rows hold the inputs and expected outputs for each step of the procedure. Click on “Show Legend” for information on legend being used in the *model\_Report.htm*.



## Test Generation Report for Model: BasicMath Rev: -

Auto-Generated by HiLiTE V. 7.5.0.1 on 1/7/2010 2:01:54 PM  
 HiLiTE Released Templates Version: 7.5.0.0; Template Family Revision: Standard=4  
 Template Family Used: Standard

[Show Help](#)
[Hide Legend](#)

### Legend:

<span style="background-color: #00FFFF; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : model input signal (input)	<span style="background-color: #00FF00; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : model output signal (output)
<span style="background-color: #FFFF00; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : input or output port of Block Under Test	<span style="background-color: #008080; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : a constant
<span style="background-color: #D3D3D3; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : min or max value of normal operating range for this column's signal	<span style="background-color: #FFDAB9; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : extra execution step inserted for signal setup/pass-thru
<span style="background-color: #FFA500; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : A value in this cell was altered to fit within Max Allowable Range, review template to ensure this is not an error	
<span style="background-color: #FF00FF; border: 1px solid black; display: inline-block; width: 15px; height: 10px;"></span> : The formula value in this cell is either malformed or entirely outside the Max Allowable Range; correct the template	
<span style="color: green;">—</span> : a normal test case not requiring model initialization before its first step	
<span style="color: magenta;">—</span> : a test case that requires model initialization before its first step	
<span style="color: green;">number, X</span> : Don't Care - nominal value computed as necessary	

Figure 28 - Legend in the *model\_Report.htm*

When you move the cursor over a cell in a test case step, the text box below the template table shows the formula used to compute that value. In Figure 27, the **Formula** box shows how the value for the cell under the cursor is calculated.

In general, test case tables show input columns first with their expected outputs following. For an input that depends on output from a preceding block, the sequence of inputs to the preceding output is also included. For example, in Figure 27, **Min** depends on the output of both the **Sum** and **Gain** blocks (labeled above as **sum.O1** and **gain.O1**). Sum is calculated from **Input1** and **Input2**.

### 5.4.3 Using Observation Points for Measuring Expected Output

Forward propagation methods can run into difficulties due to complexities downstream from the block under test. The model in Figure 29 illustrates this issue.

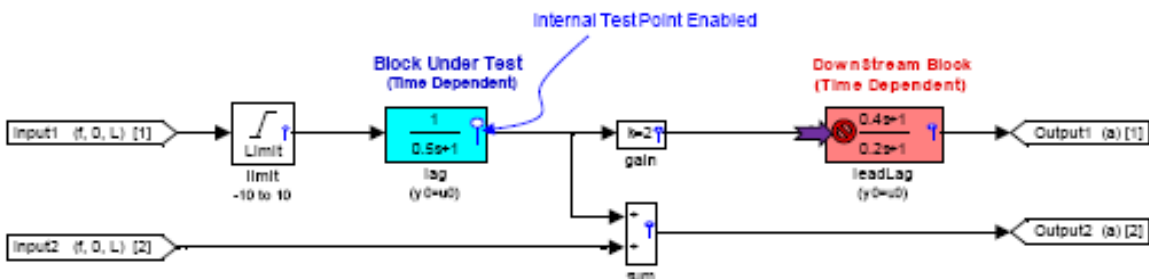


Figure 29 - Forward propagation difficulty with time-dependent blocks in series

The example model has no backward-propagation difficulty in generating the test cases for block *lag*. However, in the forward path to the measurable point *Output1*, forward propagation must occur through the *gain* and then *leadLag* blocks. The test case for the lag block is a time-sequence of vectors that must be propagated forward through another complex time-dependent block, *leadLag*. Using an observation point on the intermediate signal in the path to output can circumvent the difficulties presented by this forward propagation.

Observation points (also called test points in MATLAB terminology) are externally visible global variables in the model's generated code. Three types of observation points are available:

1. Implicit block type test points are internal within most HAM blocks
2. Explicit, model-specific test points can be included at the model level.
3. BlockIOVariables are implicit test points at the input of certain special blocks such as Stateflow charts and conditional subsystems.

### Specifying use of observation points

When code is generated for a model from HAM and RTW, you can selectively enable or disable specific types of observation points. When a type of observation point is disabled, HiLiTE will not generate any global variable for it in the code. The default code generation options enable all three types of observation points (they correspond to default options for code generation using HAM and RTW).

Option keys in the `SignalBoundaries` option set let you specify which types of observation points are enabled in the code generated for a model. The `SignalBoundaries` Option keys related to observation/test points are: `BlockTypeTestPoints`

*BlockTypeTestPoints.* The default of On indicates that that internal block-level (e.g. inside HAM block) test points are present in the model's code as externally visible global variables, so HiLiTE can use these variables as observation points in the test vectors it generates. Setting this option key to Off instructs HiLiTE not to use internal block-level test points as observation points in test vectors.

*ModelSpecificTestPoints.* The default of On indicates that model-level test points ("lollipops") are in the model's code as visible global variables, so HiLiTE can use them as observation points in test vectors. Setting the option key to Off instructs HiLiTE to not use model-level test points in test vectors.

*BlockIOVariables.* The default of On indicates that I/O variables for certain types of blocks are in code and HiLiTE can use them as observation points in test vectors, irrespective of the other test point options. Setting the key value to Off instructs HiLiTE that other TestPoint options are still applicable.

The `UseExplicitPortVariableNames` OptionSet lets you specify explicit testpoint variables in the HiLiTE Command file. Use this OptionSet to give the fully qualified names of the output port for which HiLiTE needs to find a test point variable.

### Specifying port variable names

`UseExplicitPortVariableNames` is also useful when HiLiTE is unable to find a variable for the output port. In that case, HiLiTE logs an error in the log file. For example, in this snapshot from the HiLiTE.log, HiLiTE couldn't find a variable for chart output port 7 ( `NumState.O7`):

```
[ERROR] No match found for Test Point: output port: NumState.O7, Test point block
name: NumState_TP$
[ERROR] Parsing for test points failed. No CTP output will be generated for model
Chart_To_Terminator
[ERROR] Attention HiLiTE Users!!!! Use (or Check) following option in hilite file.
```

The value of the key is the actual test point variable for that output port. To find the variable for the output port that HiLiTE couldn't find, read the `.h` code in `Block_IO` structure.

For Example:

```
/* Block signals (auto storage) */
typedef struct {
    int32_T oHamState;
    boolean_T SF_Out6;
    boolean_T SF_Out7;
    boolean_T SF_Out8;
```

```

} BlockIO_Chart_To_Terminator;
/* Block signals (auto storage) */
extern BlockIO_Chart_To_Terminator Chart_To_Terminator_B;

```

The value for the key with the Block\_IO Structure variable and dot operator, here **Chart\_To\_Terminator\_B.SF\_Out7**. In the command file, you would include this parameter:

```

<OptionSet name= "UseExplicitPortVariableNames">
  <Option key= "Chart_To_Terminator/NumState.07">
    Chart_To_Terminator_B.SF_Out7
  </Option>
</OptionSet>

```

## 5.5 Examples of Test Generation

The following examples show you how HiLiTE handles specific model features. The examples include illustrations of the model features and relevant portions of the html reports generated from each model.

### 5.5.1 Validated Localizer

The Validated Localizer model illustrated in Figure 30 uses HAM blocks and was developed from the following requirements for localizer signal processing:

- The incoming localizer signal shall be declared valid for use by the system if the raw signal valid is true continuously for at least one second.
- The incoming localizer signal shall be declared invalid for use by the system if the raw signal valid is false continuously for at least 0.5 seconds.
- The incoming localizer signal shall be attenuated by -3dB at a frequency of 5 Hz and by -20dB at 50 Hz.
- When the raw localizer signal is invalid, the input to the attenuation filter shall be held to the last valid value of the raw input.
- When the incoming localizer signal is declared invalid for use by the system, the output of the value to the system shall be set to 600 microAmps.

HiLiTE generates tests from this model, then runs the tests to validate the assumptions in the model (that is, it does not test against the original requirements). If the test harness output does not reflect the expected output, the code in the model is not properly implemented. Figure 30 through Figure 32 show the models and parts of the HiLiTE testing reports.

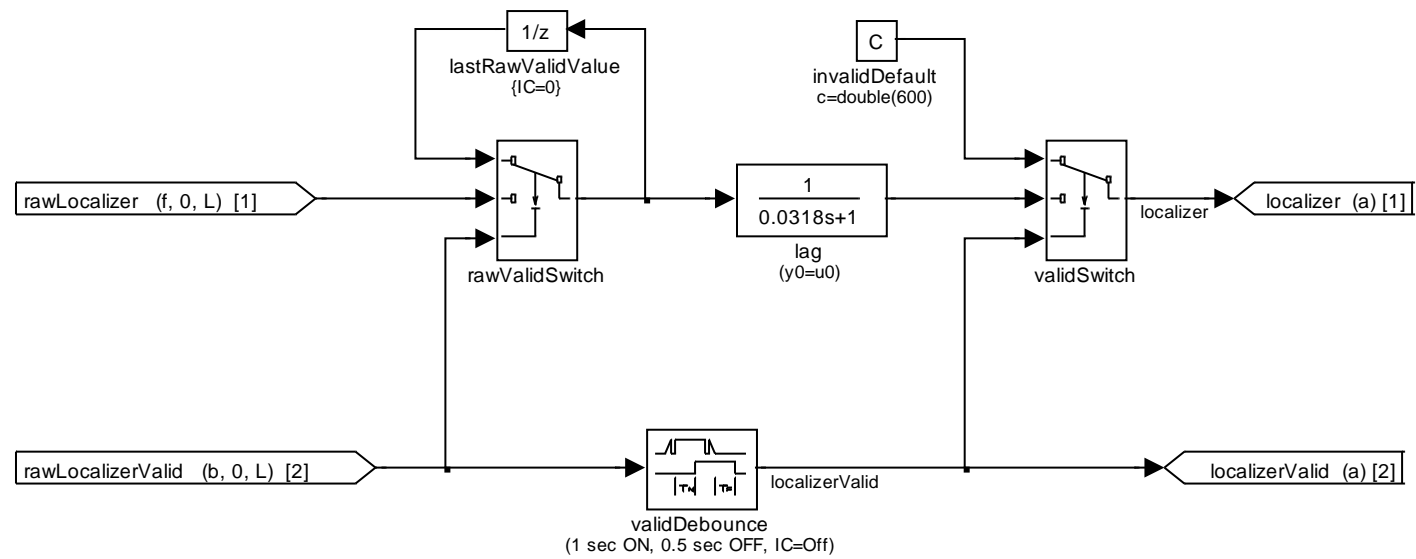
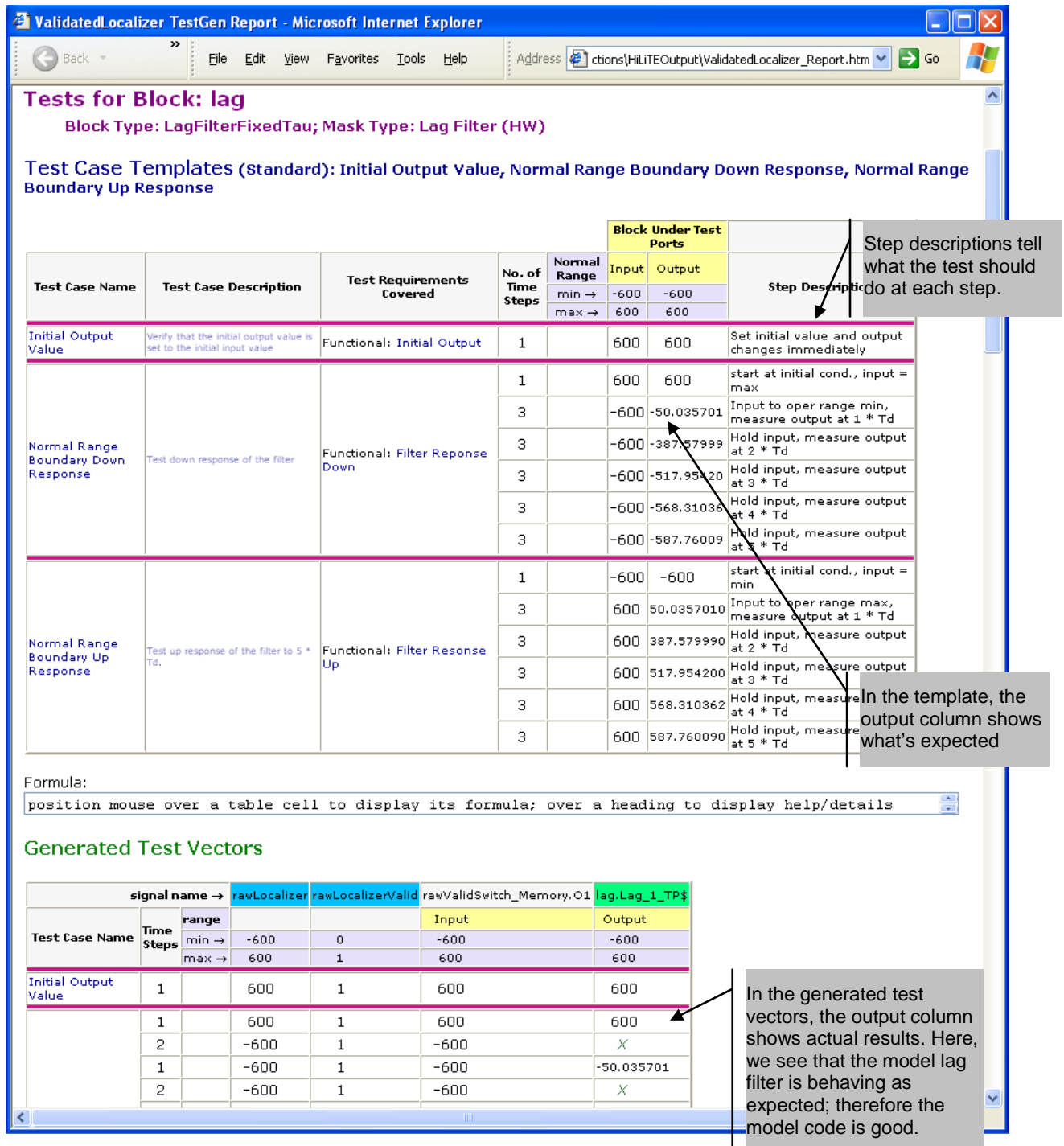


Figure 30 - Validated localizer model

Figure 31 shows the portion of the HiLiTE test report relating to the Lag filter (middle block in Figure 30). The top table is the test case template and the bottom table shows the generated testing results. There is only one test procedure covering the lag filter.



**Figure 31 - Template and test vectors for lag filter block**

The procedures do not require the model to be initialized (noted by the green line above each template and test case). The annotations for the test template and test case results Figure 31 are fairly detailed,

## Section 5 Generating Test Cases

giving the reader a good understanding of what is being tested. The results show that the template expectations are being met.

Figure 32 shows a report of the test procedure for the rawValidSwitch1\_Memory pattern. This pattern is a combination of the rawValidSwitch and the lastRawValidValue blocks. The pattern output goes through the lag filter (see previous example) before input to the ValidSwitch block.

ValidatedLocalizer TestGen Report - Microsoft Internet Explorer

Back File Edit View Favorites Tools Help Address -TG-HAM\ComplexFunctions\HILITEOutput\ValidatedLocalizer\_Report.htm Go

### Tests for Block: rawValidSwitch\_Memory

Block Type: SwitchUnitDelaySubSystem; Mask Type: not found

Test Case Templates (standard): InitialConditionTest, MaxInputToOutputWith2StepMemory, MemorySwitchOperationSequence, MinInputToOutputWithMemory

Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Normal Range	Block Under Test Ports			Step Description
					True	Flag	SwitchOut	
					min →	0	-600	
InitialConditionTest	missing in the template	Functional: Initial Condition	1	min →	-600	0	-600	
MaxInputToOutputWith2StepMemory	missing in the template	Functional: Normal Operation	1	max →	600	1	600	
			1	X	0	600		
			1	X	0	600		
MemorySwitchOperationSequence	normal operation of memory switch	Functional: MemorySwitchOperationSequence	1		-600	1	-600	set memory from input
			1		-599	0	-600	ignore current input value
			1		-600	1	-600	accept input value
			1		599	0	-600	ignore changed input value
			1		600	1	600	accept changed input value
MinInputToOutputWithMemory	missing in the template	Functional: Normal Operation	1		-600	1	-600	
			1		X	0	-600	

Formula: in2.op.min

### Generated Test Vectors

Test Case Name	Time Steps	signal name → rawLocalizerValid		rawValidSwitch_Memory.O1	lag.Lag_1_TP\$
		range	Flag	SwitchOut	
		min →	0	-600	-600
InitialConditionTest	1	max →	1	600	600
MaxInputToOutputWith2StepMemory					
MemorySwitchOperationSequence					
MinInputToOutputWithMemory					

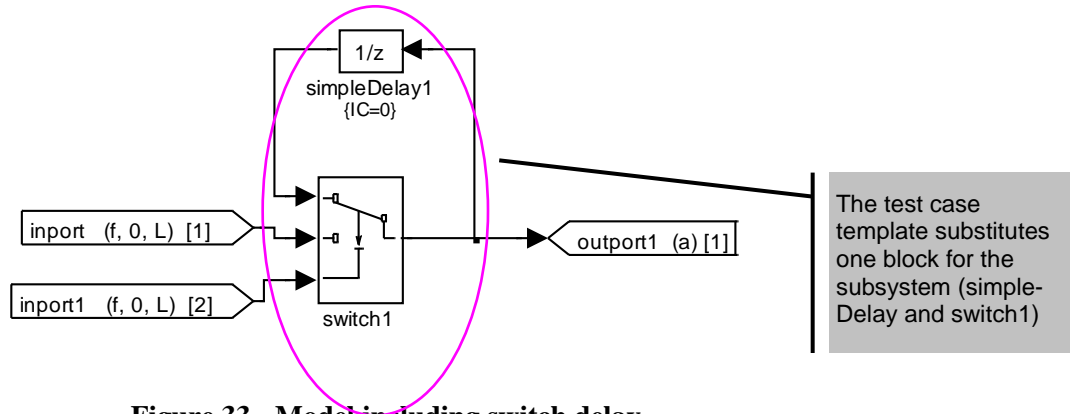
For Help or Legend, see top of page.

**Figure 32 - Template and test cases for block rawValidSwitch1\_Memory**

The validSwitch memory procedure is described in the first table and the generated tests are shown in the second table. The procedure requires initialization (indicated by the red bar).

### 5.5.2 Memory Switch Patterns

HiLiTE supports all variations of memory switch patterns with a Frame Delay block feeding back to the True or False input of the Switch block. The model in Figure 33 is one example of a memory switch pattern.



**Figure 33 - Model including switch delay**

HiLiTE substitutes a single block for the Unit Delay and Switch subsystem with their feedback loop. Figure 34 shows the entire test report for this model (in this case, SwitchUnitDelayFalseSubSystem) having two inputs and one output.

Note: This model is %HiLiTEHome%\Examples\HiLiTE-TG-HAM\ComplexFunctions\MemorySwitch.mdl. It's associated command file is %HiLiTEHome%\Examples\HiLiTE-TG-HAM\ComplexFunctions\MemoryTest.hilite.

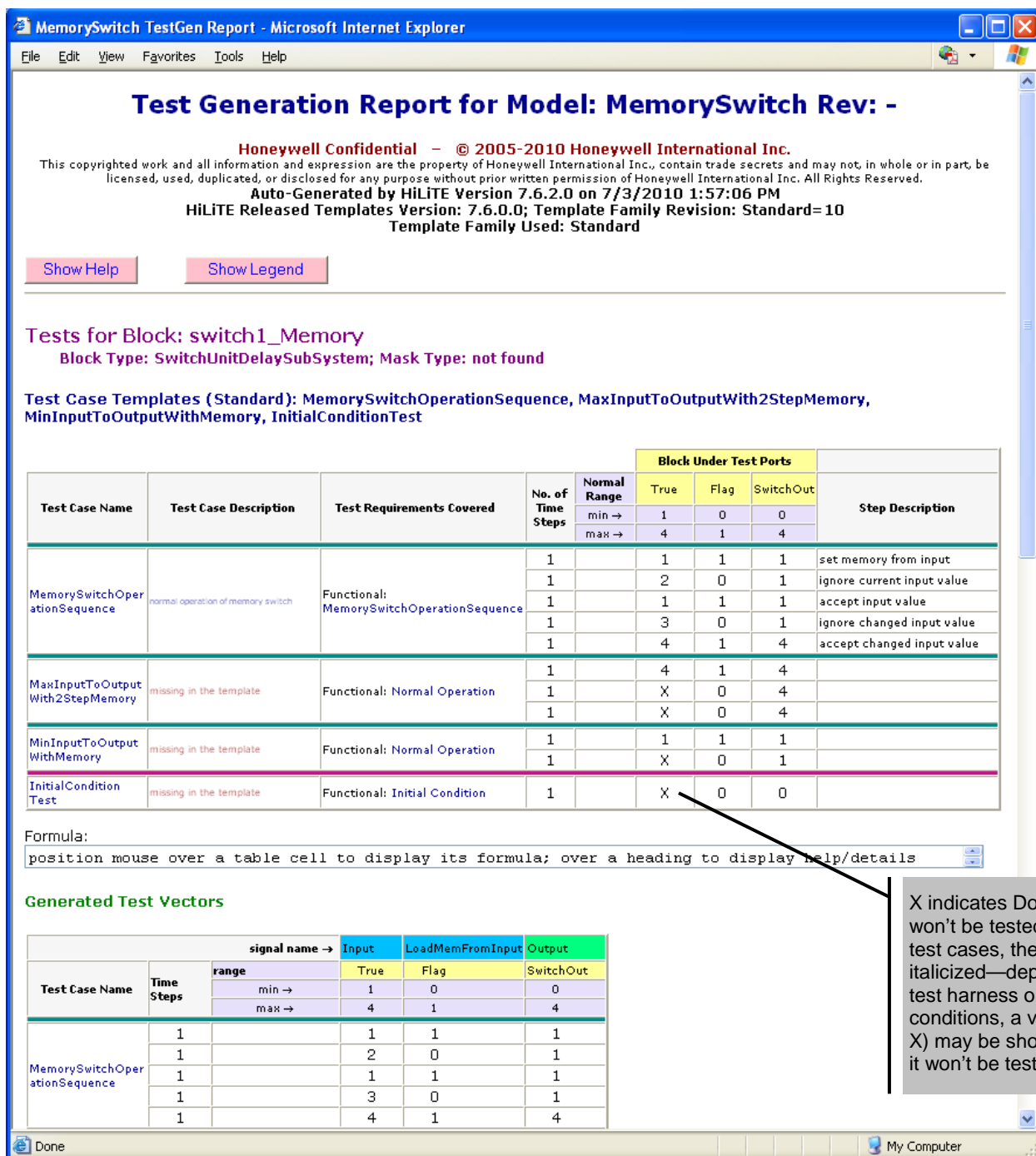


Figure 34 - Test report for memory switch

### 5.5.3 State Flow Charts

State flow chart test case generation is described in Section 6.



## 5.6 Specifying Tolerance in Measurement of Expected Output Values

Each test case contains the expected values of a model output or an observation point variable. To ascertain the pass/fail status of the test case, the expected value is compared to the actual value measured on the model's object code. The pass/fail criteria contains the notion of *tolerance*; i.e., the actual value must be within a certain *tolerance* of the expected value for the test vector to pass.

**Significance of tolerance in measurement of floating point values.** Floating point computations that use blocks such as sum, product, divide, filters, numerical errors can occur due to the machine's floating point representation inaccuracies as described in IEEE standard 754. Therefore, a tolerance must be used when comparing expected output values with actual measured values for floating point variables. The tolerance depends on the size of the target floating point data type. For 32-bit (single) floating point implementation, the tolerance is higher than for 64-bit (double) floating point implementation. The tolerance also depends on the type of operation being performed and whether operations are cascaded one after another (allowing for the accumulation of errors).

### 5.6.1 Default Tolerances used in Expected Output Measurement

The following tolerances are default HiLiTE behavior for the data types:

#### Integer

Tolerance =  $\pm 0\%$  (i.e., expected and actual values must be exactly equal)

#### Boolean

Tolerance =  $\pm 0\%$  (i.e., expected and actual values must be exactly equal)

#### Floating Point

Tolerance =  $\max(5.0 \cdot 10^E, L)$

$E = \text{floor}(\log(\text{TestRangeMax} - \text{TestRangeMin})) - N$

*TestRange* is the range of values that normal tests can span;

i.e., it is the operational range at the output being measured

$N = 4$  for Single Precision (32-bits) or  $N = 8$  for Double Precision (64-bits)

$L = 5.0 \cdot 10^{-6}$  for Single Precision or  $L = 5.0 \cdot 10^{-14}$  for Double Precision

(Tolerance used for CTP Output)

#### Floating Point Tolerance used by HiLiTE Test Harness

(note that floating point tolerance used by by HiLiTE Test Harness (HTH) is different than the formula described above; see Section 11.5 for details.)

Single Precision Floating Point  $\pm 0.01\%$  of expected value, with minimum value of .0001

### 5.6.2 Need for Specifying a Higher Tolerance than Default

Sometimes, a higher than default tolerance is required because of inherent numerical error in operations of floating point numbers, arising from the machine's floating point representation inaccuracies (IEEE standard 754). The following are examples of such situations<sup>1</sup>:

1. **Multiple cascaded function blocks.** Cascaded product, divide, and/or sum (subtract) operations introduce increased likelihood of inherent numerical error due to the machine's floating point representation inaccuracies as described in IEEE standard 754. For example, the final tolerance of

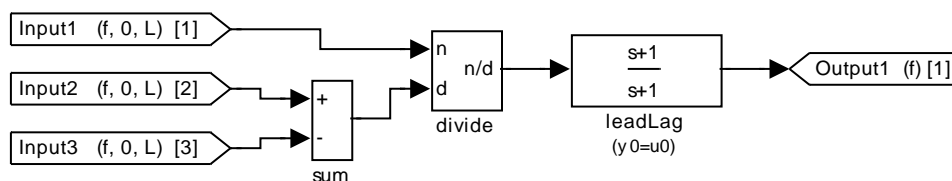
<sup>1</sup> Excerpted from: SOFTWARE TESTING GUIDELINES for the AS900 Electronic Control Unit, W4207112-1003, Rev -, 30 June 2004

a product of two uncertain factors is approximately equal to the sum of the numerical error of the factors. Similarly, the final tolerance of a subtraction of two values is should be computed based on the sum of the absolute values of the two operands—not on the subtraction result.

2. **Certain filters, differentiators, integrators, and other complex blocks.** The implementations of certain complex library blocks use cascaded subtract, product, and divide operations whose results get accumulated in internal state variables over multiple frames of execution. This situation can introduce a high inherent numerical error that requires a higher tolerance in test case execution. One guideline states that a tolerance of up to 1% can be used for these measurements.

**Specifying a higher tolerance.** HiLiTE provides a mechanism (similar to specifying model input ranges in a range file) in a CSV file for specifying a percentage tolerance higher than default for certain outputs or observation points. The CSV file will include the *tolpct* column (see Table 1. Basic HiLiTE command file elements).

Consider the example in Figure 35, which contains a series of blocks introducing high numerical error. The user needs to specify a tolerance higher than the default (0.0.1%) for the measurement of *Output1*.



**Figure 35 - Example model with series of blocks introducing high numerical error**

The following example shows how to include an appropriate specification in the HiLiTE command file:

```

<ColumnSpec specName='Tolerances' delimiters='t' commentChars='#'
  titleLines='1' >
  <Column name='symbol' col='0' />
  <Column name='tolpct' col='1' />
</ColumnSpec>
<SymbolFile fName='OutTolerances.csv' specName='Tolerances' />
  
```

Figure 36 shows the content of the file *OutTolerances.csv*, which specifies the tolerance of 0.1% for the variable *Output1*.

	A	B	C	D
1	# Model output name or Block output port's name	% tolerance value		
2	# Two Alternative Formats:			
3	# Tolerance specification when expected output is measured at a model output:			
4	# ModelOutputName	number		
5	# Tolerance specification when expected output is measured at a Test Point after a block:			
6	# modelName/blockfullName.OutputPortName	number		
7	#			
8	Output1	0.1		
9				

**Figure 36 - CSV file input to HiLiTE to specify the tolerance for an output measurement**

### 5.6.3 Guidelines for Specifying Tolerance to Override The Default

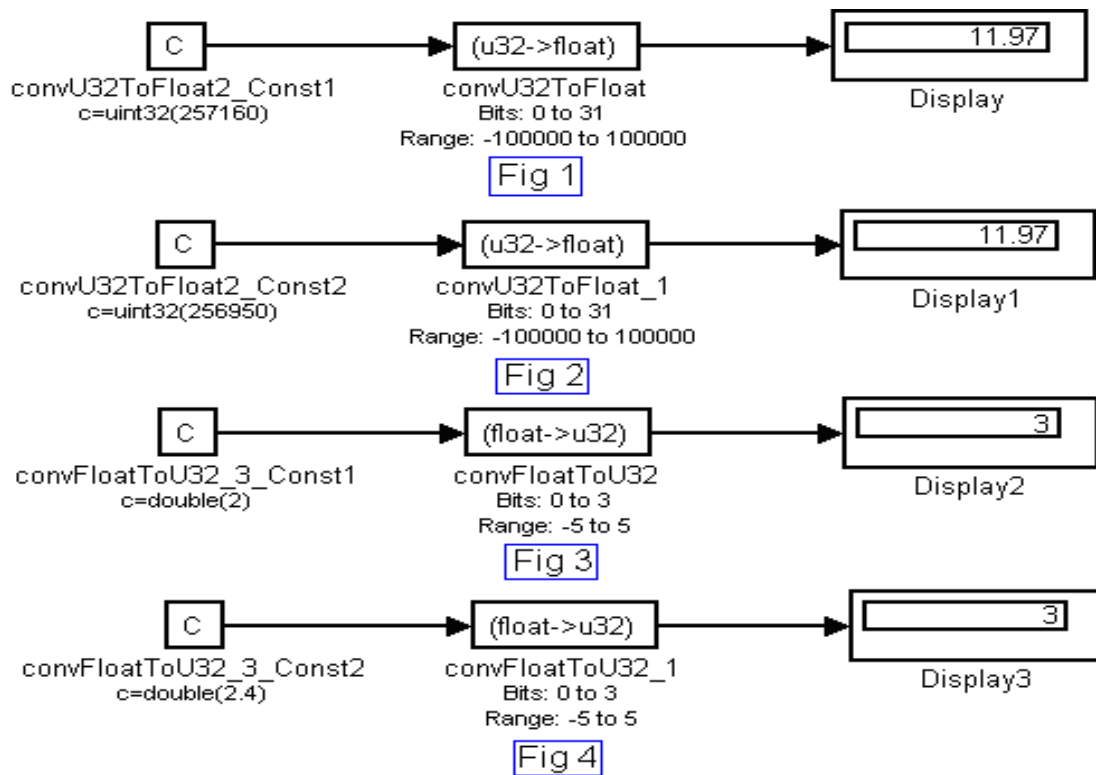
Tolerance must be specified in accordance with the testing guidelines for a product line or system being certified. The following examples describe where a larger value of tolerance may be need to be specified, but reasoned in conjunction with the testing guidelines.

1. Use of single-precision floating point in the target object code in conjunction with function blocks such as filters and integrators in the models that have complex mathematical operations that can accumulate numerical errors.
2. Use of subtraction operation in models (where two input values can be close), followed by mathematical operations such as divide and multiply. The numeric error is exacerbated if single-precision floating point data type is used.
3. A combination of single-precision floating point data type usage, large values of normal operating range specified for model inputs, and numerical error-prone mathematical operations can lead to large numerical errors at the model output values.
4. Use of multiple filters, integrators, or rate limiters types of blocks (that can produce/accumulate numerical error) in series can lead to large numerical errors at the output.

### 5.6.4 Floating Point Precision Issues for Specific Blocks

#### Test vector failures for blocks convFloatToU32 and convU32ToFloat due to precision

Precision loss may cause some test cases to fail for the blocks “Conv Float to U32 (HW)” and “Conv U32 to Float (HW)” ; i.e. if there is a small precision difference in input float value of “Conv Float to U32 (HW)” block, it leads to a considerable difference in calculating the output due to which test case may fail. A similar problem may occur for a “Conv U32 to Float (HW)” block.



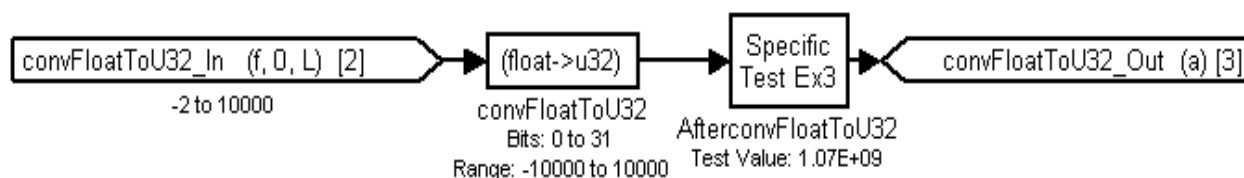
**Figure 37 - “Conv Float to U32 (HW)” and “Conv U32 to Float (HW)” blocks simulated in MATLAB by giving constant input giving same outputs for different inputs.**

In Figure 37, the block *Conv U32 to Float (HW)* in models Fig 1 and Fig 2 gives the same output value with different input values. In Fig 3 and Fig 4 the block *Conv Float to U32 (HW)* also gives the same output value with different input values.

#### Test vector failures for after block “AfterconvFloatToU32” due to precision

In the Figure 38 the After block is asking for a value “1.07E+09” from the convFloatToU32 block for which HiLiTE reverse propagated and calculated 4999.5 as input value. Precision loss in the input value “4999.5” causes the test cases to fail.

To overcome this failure, increase the tolerance value of the output port (convFloatToU32\_Out) should be increased in Range file (See reference [15] Section 4.2.1.1 “HiLiTE Command Parameters File Elements Supported by HiLiTE-TG” for option *‘tolpct’* to specify tolerance)



**Figure 38 - An example of test case failure for after block”AfterconvFloatToU32”**

Column Content	Input	Output	Execution Result
Column Name	convFloatToU32_in	convFloatToU32_Out	
Vector	4999.5	1.07E+09	Failure convFloatToU321_Out (Output:4) expected 1073634450 found 1073634422

#### Test vector failures for convU32ToFloat due to overflow problem

Sometimes test cases may fail for the blocks “Conv U32 to Float (HW)” because of an overflow issue. Specify input ranges in such way that output value does not overflow. To detect such errors, check the log file for Overflow.

## 6 Generating Stateflow Test Cases

This section gives guidance for using HiLiTE to create test cases from Stateflow models. It:

- Describes the categories of Stateflow test cases
- Tells how to test isolated Stateflow charts
- Gives instructions for modifying or creating test vectors to help HiLiTE transition to states
- Guides you in selecting Stateflow options to include in the command file
- Describes how to use the Stateflow example supplied with the HiLiTE installation

### 6.1 Stateflow Test Case Categories

Each Stateflow chart is a separate hierarchical state machine within the Simulink model. HiLiTE generates a separate set of test cases for each chart for the low-level requirements embedded in each chart. Test cases for Stateflow charts fall into these categories: state reachability, transition reachability, action coverage, and condition coverage. The status report that HiLiTE generates when running a model, as well as the onscreen progress, will show you how well each category is covered. The example in Figure 39 shows that HiLiTE reached 100% coverage for States, Transitions, and Conditions. The model had no actions.

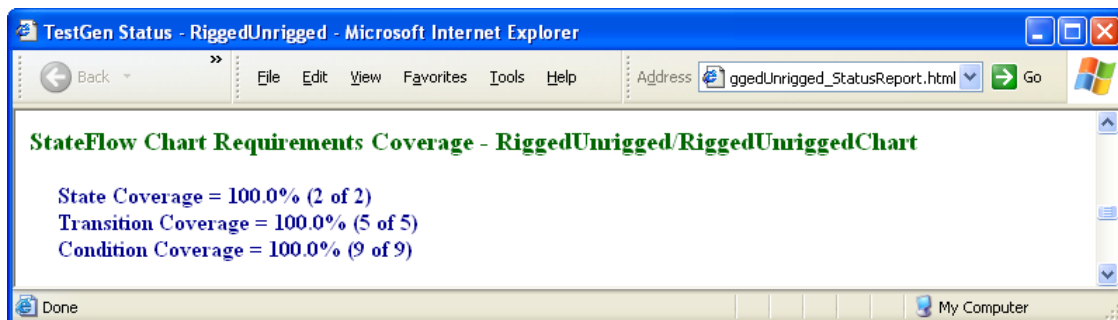


Figure 39 - Portion of Status report showing reachability for Stateflow test cases

### 6.1.1 State Reachability

HiLiTE generates a test case for each state (or combination of states if the model contains parallel states) the test generator is able to reach. This test vector will reproduce the *first* path found to the state, not the shortest path and not every path. The test cases are named *Reach\_State(s)/State names* if only one state is active (e.g., *Reach\_State/complete* or *Reach\_States/testRunning/computeAvg*) in the HTML Report file and any output files specified in the HiLiTE options. These tests are written as CSV files named *Chart name\_Reach\_State(s)\_leaf state names.csv* (e.g., *WheelRiggingTest\_Reach\_State\_computeAvg.csv*) in the *StateflowDebug* directory in the vector output directory, where leaf states are active states that have no children.

### 6.1.2 Transition Reachability

HiLiTE generates a test case for each transition the test generator is able to fire. The test case is named *Reach\_Trans/Origin name\_to\_Destination name[index]* where ‘index’ differentiates multiple transitions between the same pair of states (e.g., *Reach\_Trans/complete\_to\_defaultFlags[000]*). Note that a transition begins and ends at states, not at junctions. A transition may be composed of multiple segments between junctions as long as the source and destination are states. Segments branching off of junctions can result in multiple logical transitions sharing one or more segments. For example, if a segment connects a state with a junction from which two more segments exit, two logical transitions with a common initial segment result. HiLiTE will generate Transition Reachability test cases for all segments. In addition to the HTML Report file (and any other output files specified in the HiLiTE options), these transition reachability test cases are written as CSV files named *Chart name\_Reach\_Trans\_Origin name\_to\_Destination name[index].csv* (e.g., *WheelRiggingTest\_Reach\_Trans\_complete\_to\_defaultFlags[000].csv*) in the *StateflowDebug* directory in the vector output directory.

### 6.1.3 Action Coverage

In addition to reaching states, HiLiTE tracks during and exit actions of states to make sure they fire. As entry action coverage is synonymous with state reachability (as condition and transitions action coverage are synonymous with transition reachability), they are not tracked separately. After the state and transition reachability test cases have been generated, any during or exit actions on the states it did reach that have not fired in the course of the other test cases are specifically targeted. HiLiTE will generate test cases named *Reach\_Action/state name\_on\_action type* where action type is either “During” or “Exit” (e.g., *Reach\_Action/complete\_on\_During*) in the HTML report any output files specified in the HiLiTE options. Additionally, these test cases are written as CSV files named *Chart name\_Reach\_Action\_state name\_on\_action type.csv* (e.g., *WheelRiggingTest\_Reach\_Action\_complete\_on\_During.csv*) in the *StateflowDebug* directory in the vector output directory.

### 6.1.4 Condition Coverage

HiLiTE generates multiple test cases for each truth table decision condition and each transition that has a guard containing a Boolean expression. For transitions consisting of multiple segments, a conjunction of the multiple guards are formed. Condition coverage test cases are generated for all of these.

The condition coverage test cases show that every atomic Boolean expression has an effect on the decision or guard. On transitions, they show that each event in a disjunction of event triggers fires the transition and that an event that is not a trigger will not fire it. They show that each atomic Boolean expression in a guard or condition affects the outcome by testing each expression at a time while holding the others constant such that the overall expression can evaluate to true for the appropriate value of the “expression under test.”

For transitions, test cases are named *Trans/Origin\_name\_to\_Destination\_name/length\_x/event(Event\_name)/TEST\_TYPE/(Expression\_under\_test)*. Here *x* is the number of transition segments of

which the transition consists, *Event\_name* is the name of the event signal, TEST\_TYPE is either TEST\_PASS or TEST\_FAIL, and *Expression\_under\_test* is the Boolean expression that is being tested.

For truth tables, test cases are named TruthTable/*TruthTable\_name*/DecisionX/ConditionY==VALUE/TEST\_TYPE/( *Expression\_under\_test*). Here *X* is the Decision number, *Y* is the Condition number, VALUE is either TRUE or FALSE dependent upon the associated entry in the TruthTable, TEST\_TYPE is either TEST\_PASS or TEST\_FAIL, and *Expression\_under\_test* is the Boolean expression that is being tested.

### 6.1.5 HiLiTE Limitations for Stateflow Testing

Stateflow charts that contain bitwise XOR operators (‘^’ when C-bit operations are enabled), functions, history junctions, temporal logic, and/or external data stores are not supported by HiLiTE-TG. Input, output, and local events, string data types, and user-defined data types (e.g., “structs”) are not supported. One-dimensional arrays of primitive types in any size supported by Simulink/Stateflow are supported; however, array notation is not supported—scalar expansion is the only supported mechanism for working with arrays. Integration with Simulink is limited in that only uniform arrays (arrays in which all values are equal) are supported.

## 6.2 Testing a Stateflow Chart as an “Isolated Block” within the Model

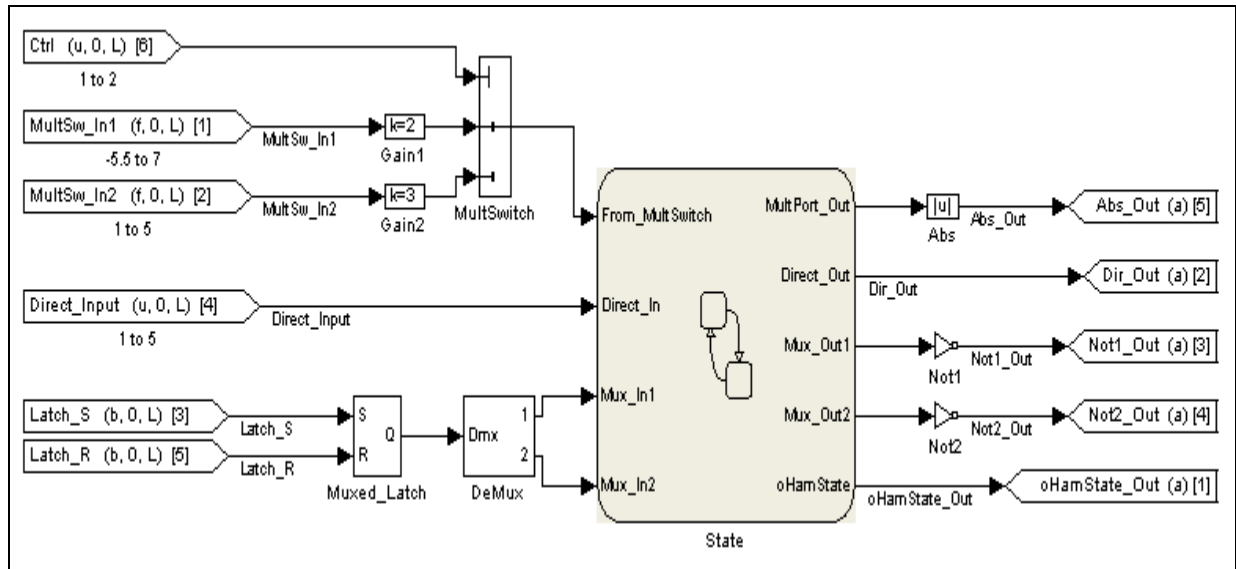
When both a Stateflow chart and its context within a model are complex, HiLiTE can often create test cases for testing the states and transitions in the chart but cannot propagate all of these test cases through the model blocks upstream of the chart to the external model inputs. In this case you may need to create tests for the chart in isolation to make sure the maximum numbers of test cases are generated. The whole model may not require all possible Stateflow behaviors, so isolating tests will make certain that all behaviors are tested.

To generate the vectors for the Stateflow chart without the constraints of the rest of the model, include *SignalBoundaries* options in the HiLiTE command file:

```
<OptionSet name="Signal Boundaries">
  <Option key="IsolateBlocks">On</Option>
</OptionSet>
```

Setting the option key *IsolateBlocks* to On tells HiLiTE to generate tests for each isolatable block in a separate test vector file. The isolated file generated is named *<PartitionName>\_<ModelName>\_<StateChartName>\_ctp.tpi* (or *.csv*). It consists of only isolated vectors. If the model includes more than one Stateflow chart, HiLiTE-TG generates isolated files for each Stateflow chart. These isolated vectors will not appear in the *<ModelName>\_Vectors.csv* test vector file. HiLiTE-TG correctly sets the model-level inputs if they are connected directly to the chart or the first available test point variable appearing upstream the chart. The outputs are the chart outputs.

Consider the example in Figure 40 to see how the inputs and outputs are set in the isolated TPI file. The model used is *Statechart\_In\_Isolation.mdl* (Under ExampleModels).



**Figure 40 - An example Stateflow model to be tested in isolation**

The model in Figure 40 includes Simulink blocks upstream of the Stateflow chart. When the chart is tested in isolation, the model context is ignored and it is tested a standalone against the chart function `<ModelName>_<ChartName> (void)` in the .c file. The inputs and the outputs set in the isolated TPI file will be the global input variables and the chart outputs. “Global inputs” indicates either the model inputs when chart inputs are connected directly to the model-level inputs or the test point variables found in the blocks upstream of the chart. When virtual blocks come immediately before the chart, HiLiTE traverses further backwards until it finds a global variable to set the inputs. In Figure 40, the fourth input will be the test point of the Latch block, as Demux is a virtual block. Figure 41 shows the inputs and the outputs used in the isolated TPI. Note: The number of test points will always be zero in an isolated TPI.

```

; #Inputs #Outputs #TestPoints #Testcases
; 4 5 0 11
; all variable datatypes
; Inputs:
uint32_T ;Direct_Input
real64_T ;Statechart_In_Isolation_B.MultSwitch
boolean_T ;Statechart_In_Isolation_B.Latch_1[0]
boolean_T ;Statechart_In_Isolation_B.Latch_1[1]
; Outputs:
int32_T ;oHamState_Out
uint32_T ;Dir_Out
boolean_T ;Statechart_In_Isolation_B.Named_Sig
real64_T ;Statechart_In_Isolation_B.MultPort_Out
boolean_T ;Statechart_In_Isolation_B.Mux_Out2
; end datatypes
; all variable names
; Inputs:
Direct_Input
Statechart_In_Isolation_B.MultSwitch
Statechart_In_Isolation_B.Latch_1[0]
Statechart_In_Isolation_B.Latch_1[1]
; Outputs:
oHamState_Out
Dir_Out
Statechart_In_Isolation_B.Named_Sig
Statechart_In_Isolation_B.MultPort_Out
Statechart_In_Isolation_B.Mux_Out2
; end variables

```

**Figure 41 - Partial view of the isolated TPI file showing the inputs and outputs**



### 6.2.1 Script to Isolate a Large Stateflow Chart into a Separate Model

The method to improve test generation coverage by considering a StateFlow chart as an isolated block for testing purpose is sufficient for most cases. Some models, however, may be too large for HiLiTE to load from MATLAB and process without running out of memory. When this scenario occurs with a large Stateflow chart embedded within a large Simulink model, a script is provided (`isolateStateflow.m`) to isolate the Stateflow chart in a separate model and generate the tests independently from that isolated model. The isolated StateFlow chart preserves the model rate and certain properties of chart inputs/ outputs so that the test vectors properly reference the StateFlow chart’s inputs/outputs as embedded in the original model’s code. The inputs and the outputs set in the isolated TPI file will be the global input variables and the chart outputs. “Global inputs” indicates either the model inputs when chart inputs are connected directly to the model-level inputs or the test point variables found in the blocks upstream of the chart. When virtual blocks come immediately before the chart, HiLiTE traverses further backwards until it finds a global variable to set the inputs. Refer Figure 38 and 39 for example. The generated tests can then be run on the original source code generated from the original integrated model.

#### Using the script

1. Set the correct directory path in the MATLAB path
2. Copy the Source code of the model into the directory having the same name as the model. The source code directory and model should be at one folder level
3. Type `isolateStateflow('<modelname>')` in the Matlabh command prompt. (Do not include the extension `.mdl` after the model name.)

Figure 42 shows the model snapshot which is created by the script with proper inputs and outputs set.

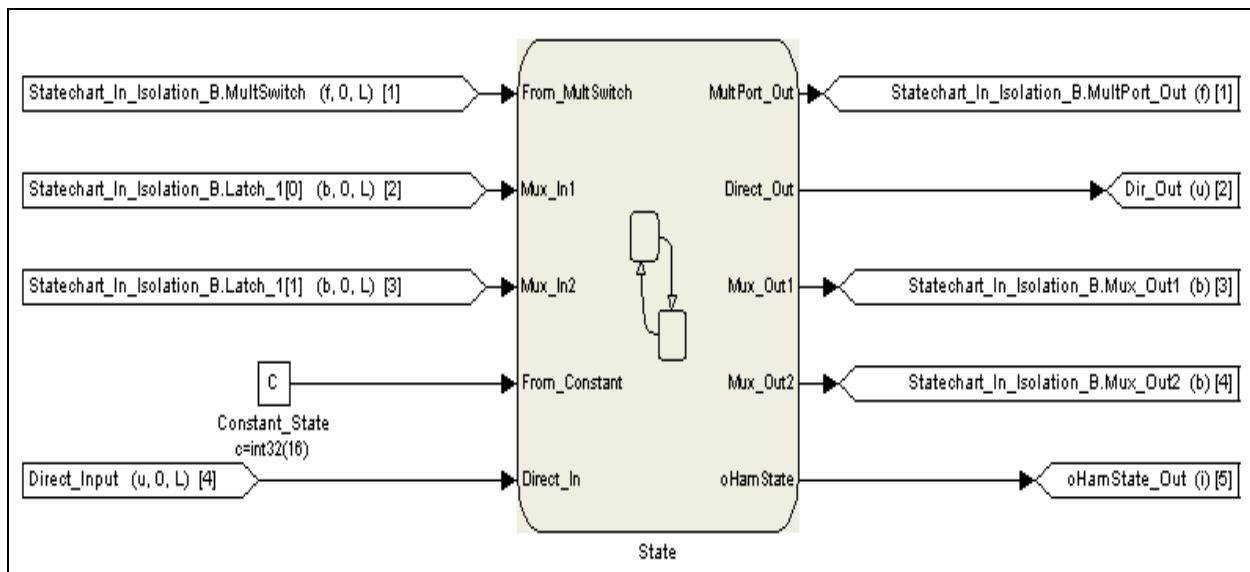
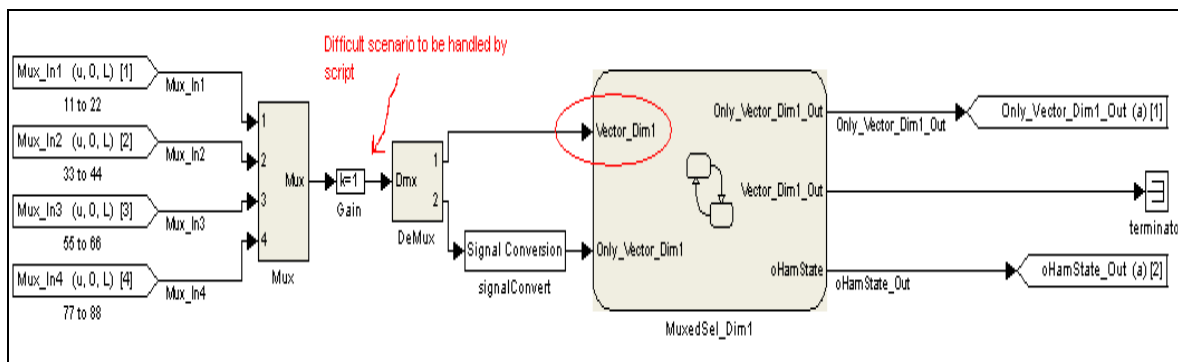


Figure 42 - The isolated model created by the `isolateStateflow.m` script

#### Error handling in `isolateStateflow.m` script

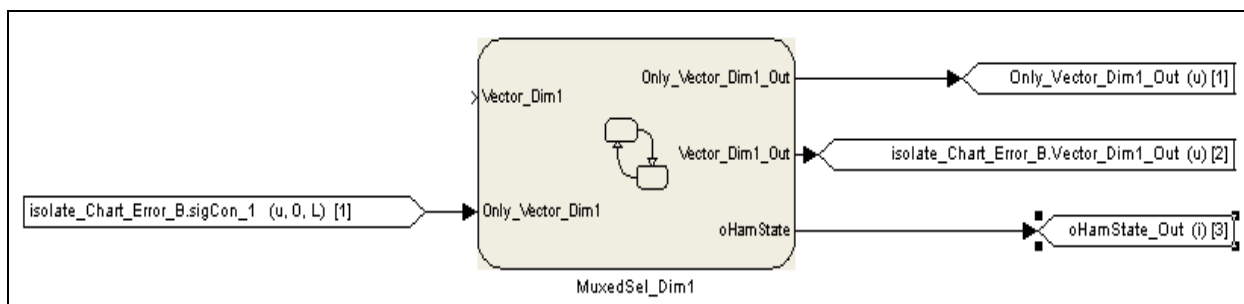
In some complex scenarios, the script may not figure out the test Points for a particular chart input. In such cases, the script will give an error in the MATLAB command prompt indicating the exact Input/Output port number of the chart (see Figure 44). These ports must be connected manually to the Input/Output blocks with correct input/output variables (Figure 45). In addition, the script may add unconnected input/output blocks in the model. These unconnected input/output blocks must be deleted manually from the model.



**Figure 43 - Difficult scenario for the Script; it will not be able to resolve Input port 1 of the chart.**

```
>> isolateStateflow('isolate_Chart_Error')
Running HAM Stateflow Add Trace Anchor Version : 3.0
An error occurred for input 1 of chart MuxedSel_Dim1. Please Add the input manually
Running HAM Stateflow Add Trace Anchor Version : 3.0
Input port 1 of 'isolate_Chart_Error_MuxedSel_Dim1_sf/MuxedSel_Dim1' is not connected.
>>
```

**Figure 44 - The error reported by the script in MATLAB command prompt indicating the port number of the chart**



**Figure 45 - incomplete model created by the script with input unattached**

### 6.3 User Supplied Test Vectors (USV) to Complete Test Generation

State-space explosion, complex cross-state interactions, and so on may prevent HiLiTE from processing all models or may require an extensive amount of time to reach one part of a model. User-supplied test vectors (USV) provide a powerful approach to get past model structures that HiLiTE cannot process efficiently. You can supply a hand-generated or a modified vector that reaches a state, fires an action, or otherwise sets up further tests. HiLiTE can proceed from the USV to generate additional downstream tests. In other words, USVs can jump-start Stateflow test generation in HiLiTE.

Refer also to the document, *HiLiTE Test Vector File Format* [15].

#### Steps to creating a USV

1. Identify the tests that HiLiTE failed to generate.
2. Analyze the missed tests in the context of the model to determine the originating state and/or the particular transition that HiLiTE was unable to fire.

3. Copy a reachability test case for the source state or for a nearby state from the Debug Vectors directory.
4. Modify and/or add additional vector rows in order to fire the target transition.
5. Create new .hilite file that will import the user-supplied vector.

```
<OptionSet name="StateflowOptions">
```

```
  <Option key="ChartInputVectorFiles">UserSupplied_Vector1.csv </Option>
```

```
</OptionSet>
```

6. Re-run HiLiTE using the new .hilite file.

### 6.3.1 Debug Vectors

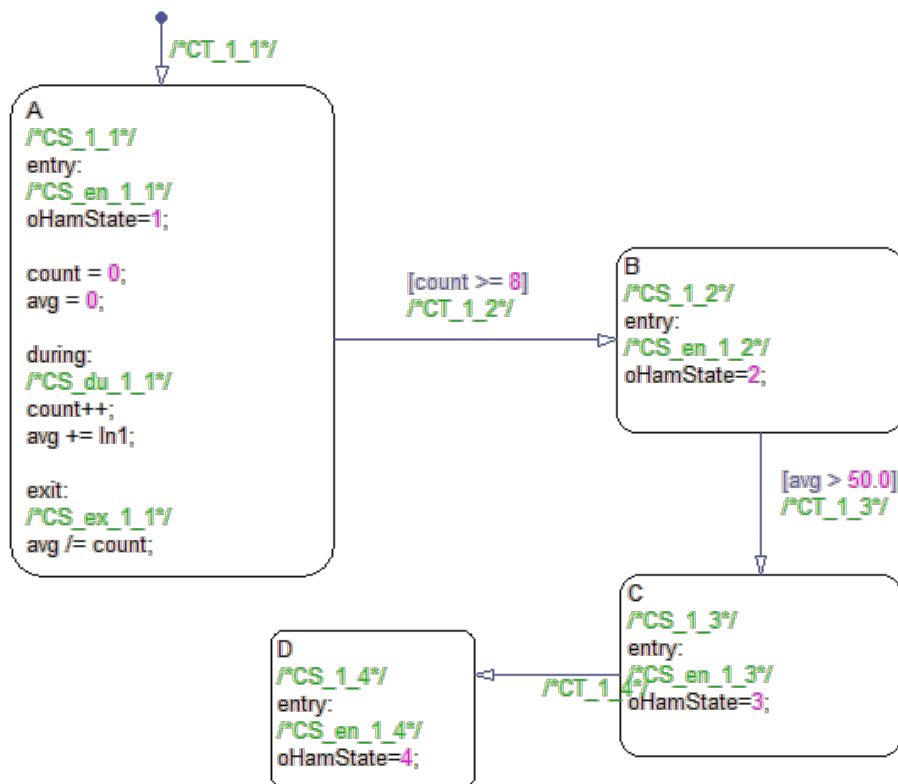
Debug vectors are output by HiLiTE to Vectors\StateflowDebug\StateflowDebug.zip. The zip file contains a CSV file for each State reached, each of the Transitions fired, and each set of condition tests generated. The debug vectors generally comply with the specification provided in %HiLiTE\_Home%\Doc\HiLiTE-TestVectorFileFormat.doc and specify the time sequence of required inputs and expected outputs associated with each test. The CSV files can be copied and modified to support manual generation of USVs.

### 6.3.2 Writer Report

The Writer Report is written to \Reports\<StatechartName>\_WriterReport.txt. It has two parts. The first part describes, for each local and output variable, where in the model the variable is set and the assignment statement that sets the variable. The second part describes, for each State in the model, the range of possible values of all local and output variables in that state. This file can be accessed to support manual creation of USVs.

```
And_Out (UINT32) : 1: OUTPUT_DATA
  CBit_EnabledOperators/And_0p/And_Out1
    (And_Out:=(And_In1 BIT_AND And_In2))
  CBit_EnabledOperators/And_0p/And_Out2
    (And_Out:=(And_In1 BIT_AND And_In2))
*****
oHamState(INT32) : 2: OUTPUT_DATA
  CBit_EnabledOperators/And_0p/And_Out1
    (oHamState:=2. 0)
  CBit_EnabledOperators/And_0p/And_Out2
    (oHamState:=3. 0)
  CBit_EnabledOperators/And_0p/Enter
    (oHamState:=1. 0)
*****
*****
CBit_EnabledOperators/And_0p/And_Out1
  And_Out: r(-Infinity:Infinity)
  oHamState: r[2]
CBit_EnabledOperators/And_0p/And_Out2
  And_Out: r(-Infinity:Infinity)
  oHamState: r[3]
CBit_EnabledOperators/And_0p/Enter
  And_Out: r[0]
  oHamState: r[1]
*****
```

Consider the example in Figure 46—HiLiTE cannot reach State C or any state beyond C. The variable AVG is incremented in State A in the example. Upon exit, it is divided by the number of increments to compute the average. Later timesteps read the value of AVG to determine if transition CT\_1\_3 can fire. Since In1 in State A starts at the default value of 0, AVG = 0 when it gets to State B—too late to be set properly. A USV can set In1 to a better value.



**Figure 46 - An example Stateflow model in which HiLiTE cannot fire the transition from B to C.**

After running HiLiTE, the StateflowDebug directory will contain all the Stateflow vectors that HiLiTE can generate. The vector that reaches State B can be modified and the command file updated to use it, so HiLiTE can reach State C (and, from there, State D).

Figure 47 shows a vector generated by HiLiTE that has been modified to reach State B.

HiLiTE-1-0	ColumnC	Vector	Initialize	ColumnN	Comment	Information		
Informatic Chart="SC Date="6/2 HiLite Version="7.3.0.9.B2_12169"								
ColumnC	Repetition	Event	Input:1	Output:1	State:0	State:1	State:2	State:3
ColumnName			In1	oHamStat	A	B	C	D
Comment								
Initialize								
InterfaceC	1	1	1	4				
Comment	REACH State A							
Comment	REACH Transition segment from Initial to A							
Vector	1	T		1	A			
Comment	REACH During Action on A							
Vector	7	T		51	1	A		
Vector	1	T		51	1	A		
Comment	REACH exit actions for state A							
Comment	REACH State B							
Comment	REACH Transition segment [count >= 8] from A to B							
Vector	1	T		2	I	A		

**Figure 47 - Hand-modified vector so HiLiTE can reach state C (and progress to D).**

Chart-level vectors specify chart inputs and outputs in port order and the states within the chart. Each line is either a command (e.g., “Initialize” indicating the next step is the first step in the chart), a comment, or a vector. Vector lines specify input values (blank for don’t care), expected outputs, and state indicators: “A” active, “I” for inactive, or blank for don’t care states. When the values in the vectors on lines 12 and 13 are changed to 51 ( shown in red), HiLiTE can progress to State C.

In this example, the first seven lines describe the model and provide other incidental information about HiLiTE that, except in the case of multi-chart models, need not be changed from the original StateFlow Debug Vector. Vectors that apply to a single chart in a multi-chart model must specify the chart name in the Information line (see Line 2 in Figure 47). If no chart name is identified in the information line, the vector applies to all charts in the model.

Lines 8 and 9 are comments that explain what will happen in the next line. Line 10 is the first Vector line, which will fire the default transition and enter state A. The value of in1 is a don’t care because it is not read in that time step. The value of oHamState is set to 1 by the entry action on A. Line 11 is another comment describing what happens next—the system will stay in A, firing its *during* action. The second column is the execution count, so line 12 describes 7 event broadcasts holding the value of in1 at 51. State A remains active and oHamState remains 1. Line 13 also remains in State A, holding the value of in1 at 51. By setting the value of in1 to 51 for the eight time steps it spends in State A, the value of avg upon exiting A will be 51. This will allow HiLiTE to reach State C and, from there, D. Finally, in line 17, the transition to State B fires, setting oHamState to 2, inactivating State A, and activating State B.

After making these modifications, save the new vector to a file and add the following lines to the HiLiTE command file:

```
<OptionSet name="StateflowOptions">
  <Option key="ChartInputVectorFiles">Vector File</Option>
</OptionSet>
```

Where *Vector File* is the path to the newly created input vector, specified relative to the project path. When HiLiTE is run with the new option, it will generate the additional test cases to reach States C and D and report complete coverage. The ChartInputVectorFiles option can specify multiple vector file names, separated by spaces.

For more information about creating test vector files, refer to *HiLite Test Vector File Format* [15]

## 6.4 Using the StateflowOptions OptionSet

When you submit models that include Stateflow charts to HiLiTE for test generation, you can control how the test are generated by including the SateFlowOptions OptionSet in your command file. This OptionSet must contain one or more <Option> key-value subelements as shown here:

```
<OptionSet name="StateflowOptions">
  <Option key="BackwardsSearch">Off</Option>
  <Option key="MaxDepth">25</Option>
</OptionSet>
```

The following table describes each option key, along with its possible values, and some rationale for using them.

Stateflow Option Key Descriptions	Examples
<p><i>BackwardsSearch</i>: (default=On) enables or disables backward search for test cases. For complex models, this option key can be valuable for completing difficult test cases. Note that using this will take more time and memory.</p> <p><i>MaxDepth</i>: (default=5) can only be used in conjunction with the BackwardsSearch option key. It specifies the maximum depth of the</p>	<pre>&lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="BackwardsSearch"&gt;On&lt;/Option&gt;   &lt;Option key="MaxDepth"&gt;25&lt;/Option&gt;   &lt;Option     key="MaxTimerIterations"&gt;5000&lt;/Option&gt; &lt;/OptionSet&gt;</pre>

Stateflow Option Key Descriptions	Examples
<p>bounded depth-first search. A value of 5 works for most cases, but the backwards search may not find a transition that is enabled only after a loop around a series of (more than five) states, unless <b>MaxDepth</b> is set appropriately.</p> <p><b>MaxTimerIterations</b>: (default=2000) can only be used in conjunction with the <b>BackwardsSearch</b> option key. It limits the maximum number of time steps that HiLiTE looks for solutions to timer patterns in the backwards search. This situation is the bottleneck for a number of models, because timer patterns that require very many iterations (e.g., 9000) can result in stack overflows. If a timer pattern is detected that requires more iterations than is allowed by <b>MaxTimerIterations</b>, then a message suggesting the larger value will show up in the <code>modelname_Summary.xml</code> file.</p> <p><b>MaxExpressionLeafCount</b>: (default=10) sets the size of expressions that are eligible for advanced expression simplification. This option key has an unpredictable effect, but is worth experimenting with to decrease test generation time. For some models, increasing the default value will significantly decrease the time required to generate tests. However, for other models, increasing this value will significantly increase run time. Values between 10 and 500 are recommended.</p> <p><b>LargeStateflowModel</b>: (default=Off) enables or disables the partitioning of the TPI output into smaller files instead of creating a very large single file. This is valuable for decreasing memory use when processing large models.</p> <p><b>NumTestsPerTPI</b>: (default=20) is only useful in conjunction with the <b>LargeStateflowModel</b> option key. It specifies the number of test vectors to put in each TPI file.</p> <p><b>ChartInputVectorFiles</b>: (default=empty) lists user-supplied vectors (csv files) that HiLiTE must import to provide additional starting points for generating Stateflow tests. Give file names relative to <b>ProjectPath</b>.</p> <p><b>SetInputPortValue</b>: (default=none) specifies the name of a new OptionSet for specifying constant values for input variables. Such constants can “give a hint” to HiLiTE to make test generation easier. For example, in the example shown in Figure 4, you could set <b>ln1</b> to 51 rather than supplying a manual vector. This option is not appropriate for all Stateflows. Give the OptionSet a name that is not already specified (see tables in Section 3). Include one or more Option key</p>	<pre> &lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="MaxExpressionLeafCount"&gt;200 &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="LargeStateflowModel"&gt;On &lt;/Option&gt;   &lt;Option key="NumTestsPerTPI"&gt;100&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="ChartInputVectorFiles"&gt;     file_name1     file_name2     file_name3   &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="SetInputPortValue"&gt;     SF_InputPortValues   &lt;/Option&gt; &lt;/OptionSet&gt; &lt;OptionSet name="SF_InputPortValues"&gt;   &lt;Option key="model_name/chart_name/     port_name"&gt;Desired_Value &lt;/Option&gt; &lt;/OptionSet&gt; </pre>

Stateflow Option Key Descriptions	Examples
<p>values that associate Stateflow chart input variables to specific values. Each Option key value pair is the fully qualified input variable name and a value that is a double or integer.</p> <p><i>SkipTransReachability</i>: (default=none) specifies the name of a new OptionSet. Use the new option set to tell HiLiTE to skip charts that include unreachable transitions. Give the OptionSet a name that is not already specified (see tables in Section 3). The new OptionSet should contain one or more Options, each with a key that is the fully qualified name of a transition source and a value that is the fully qualified name of a transition destination (where the tail name of a junction is "Junction"). HiLiTE will skip all transitions between the source/destination pairs in this OptionSet; that is, reachability tests that have not already been generated will not be attempted.</p> <p><i>StateflowTimeoutDuration</i>: (default=-1) specifies a time, in seconds, after which HiLiTE will stop looking to generate new tests. Note that HiLiTE can spend a considerable amount of time on one test, and it won't timeout until the current test is complete. A value of -1 means no timeout.</p> <p><i>UseHardRangeConstraints</i>: (default=On) Hard range constraints arise solely due to model construction; they denote the feasible input values to the StateFlow chart derived from the data type or from upstream constants or range limiters. This options specifies that HiLiTE should attempt to generate tests by only constraining the ranges of chart input/output values to within these hard constraints that represent values feasible in the model. When this option is Off, HiLiTE generates tests by further constraining the ranges of StateFlow chart inputs to the operational ranges propagated by the upstream blocks based upon operational ranges supplied by the user for the model inputs.</p> <p><b>Caution:</b> Users are advised not to turn this option Off since that will unnecessarily suppress feasible test cases and reduce coverage.</p>	<pre> &lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="SkipTransReachability"&gt;     SF_SkipTransOptionSet &lt;/Option&gt; &lt;/OptionSet&gt; &lt;OptionSet name="SF_SkipTransOptionSet"&gt;   &lt;Option key="model_name/chart_name/     source_state_name"&gt;     model_name/chart_name/ dest_state_name   &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="StateflowTimeoutDuration"&gt;3600 &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="StateflowOptions"&gt;   &lt;Option key="UseHardRangeConstraints"&gt;Off &lt;/Option&gt; &lt;/OptionSet&gt; </pre>

## 6.5 Using the Stateflow Example

HiLiTE is installed with an example command file and Stateflow model to demonstrate how it works and what type of output to expect. The example is useful as a template for creating your own HiLiTE command files. The files you need for creating the Stateflow application and generating test cases are installed under the main HiLiTE installation directory.

%HiLiTE\_HOME%\Examples\HiLiTE-TG-Stateflow\ includes a model "RiggedUnrigged," and a command file for generating test cases.

### 6.5.1 The RiggedUnrigged Stateflow Model

RiggedUnrigged.mdl is a simple model (Figure 48) showing the Rigged/Unrigged status of an avionics maintenance system. HiLiTE generates state and transition reachability and MCDC tests for the model.

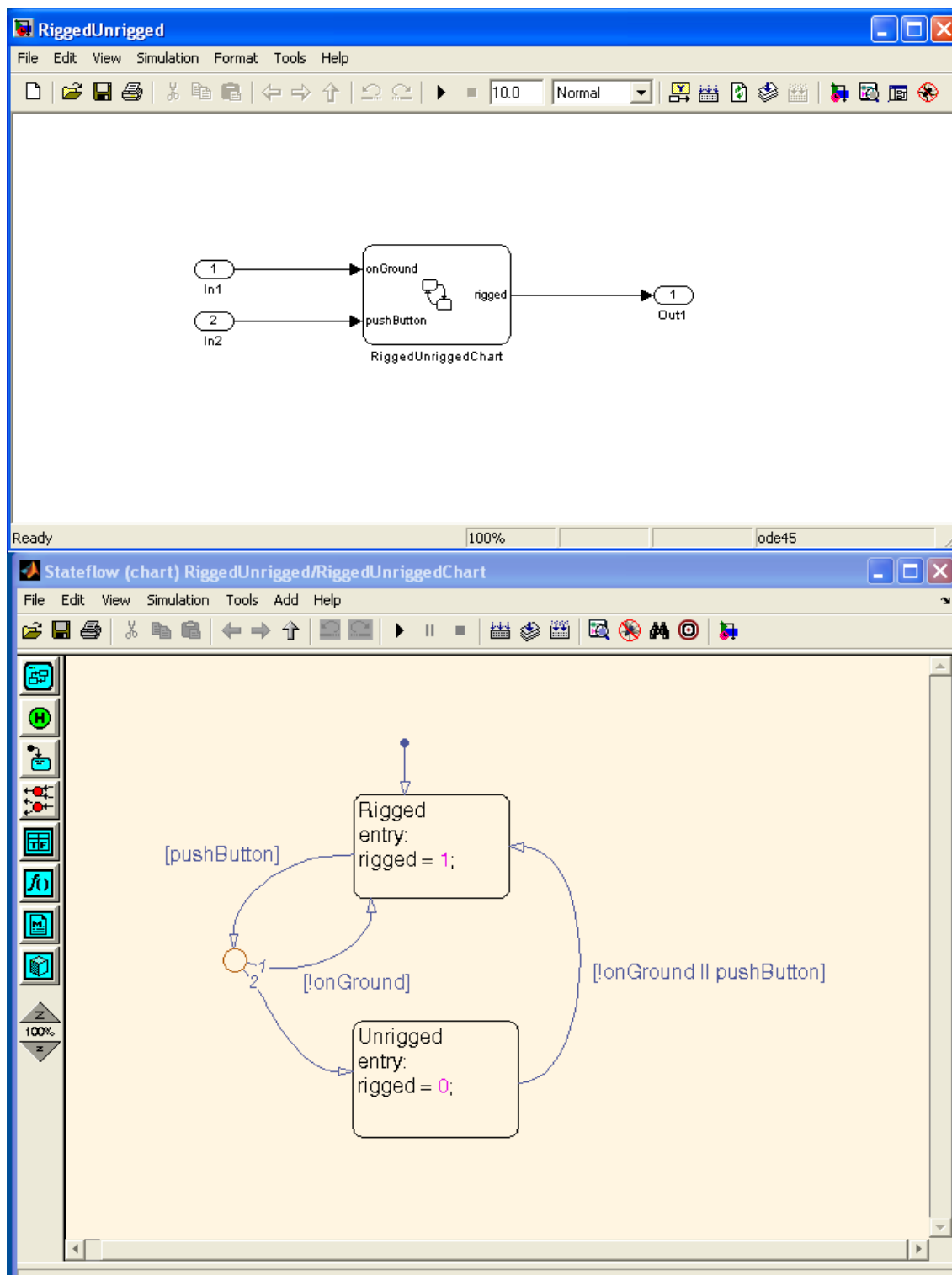


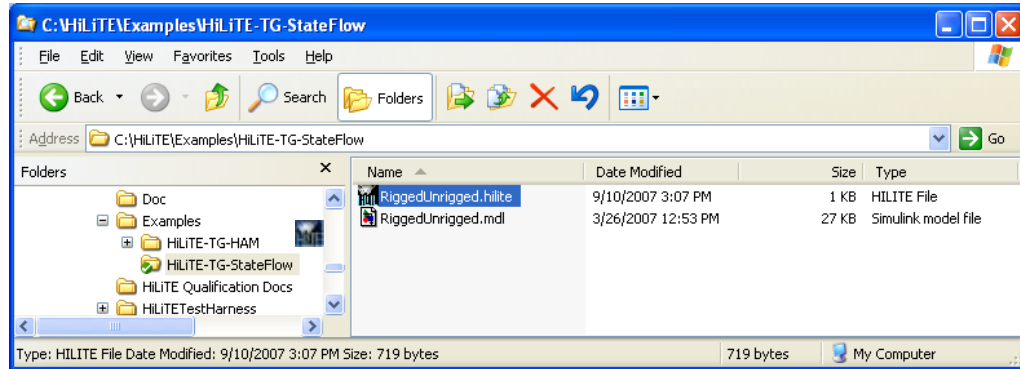
Figure 48 - RiggedUnrigged model for Stateflow example



## 6.5.2 Generating Test Cases for the RiggedUnrigged model

Use this procedure to generate test cases for the RiggedUnrigged model:

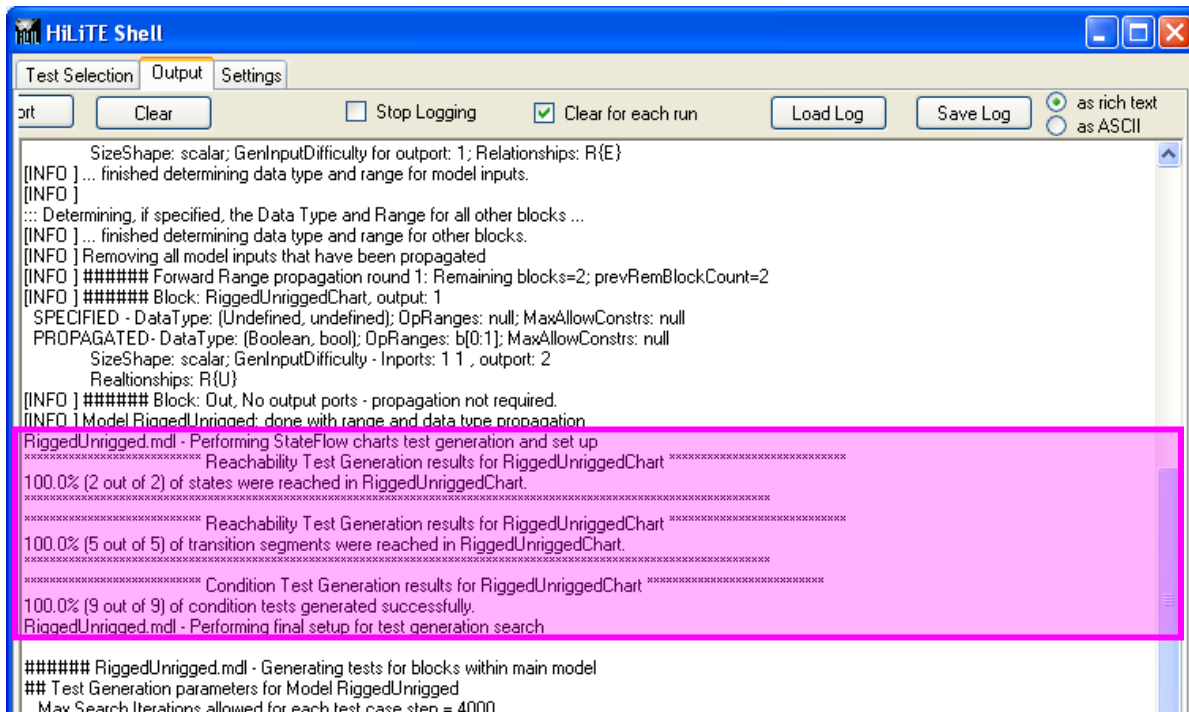
1. In a file browser, display the contents of %HiLiTE\_HOME%\Examples\HiLiTE-TG-Stateflow.



**Figure 49 - RiggedUnrigged Stateflow example model path in HiLiTE installation directory**

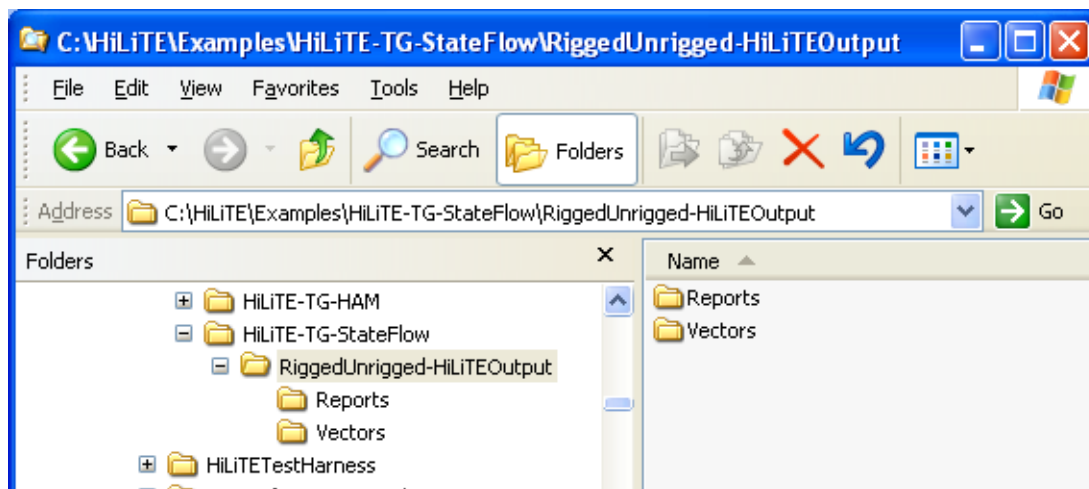
(Note about ranges: The RiggedUnrigged model uses all Boolean inputs and, therefore, needs no range file. If the model included any inputs that required a range of values, HiLiTE would require a range file to successfully complete test cases.)

2. Execute **RiggedUnrigged.hilite** in HiLiTE Shell or using a command prompt. HiLiTE opens a command window and displays its progress.
3. When HiLiTE completes the command, the output in the window will look something like the next figure. The highlighted statements show how many tests were generated and how many should have been generated (for example, 4 out of 5). If HiLiTE is completely successful, as it is here, each test type (state, transition, and condition) will show 100%.



**Figure 50 - RiggedUnrigged model execution complete status**

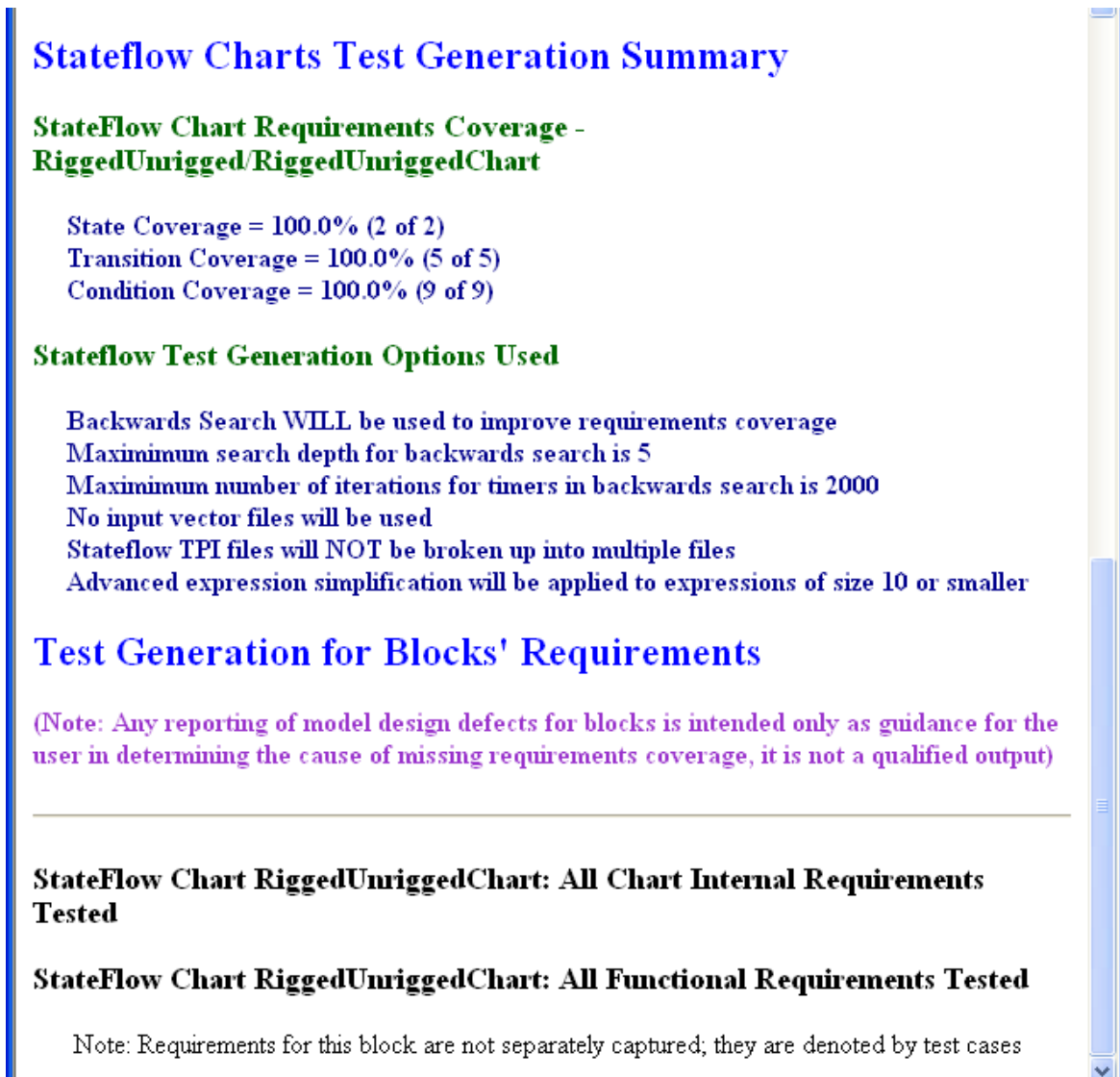
The folder from which you ran HiLiTE will include a new subfolder: \RiggedUnrigged-HiLiTEOutput which contains subfolders for the reports and vectors.



**Figure 51 - Location of results for RiggedUnrigged model**

You can define this directory structure by modifying the .hilite file.

4. Open the folder \RiggedUnrigged-HiLiTEOutput\Reports
5. Double-click **RiggedUnrigged\_StatusReport.html** Your browser will display the status report for the HiLiTE's execution using the RiggedUnrigged model. It should look like this:

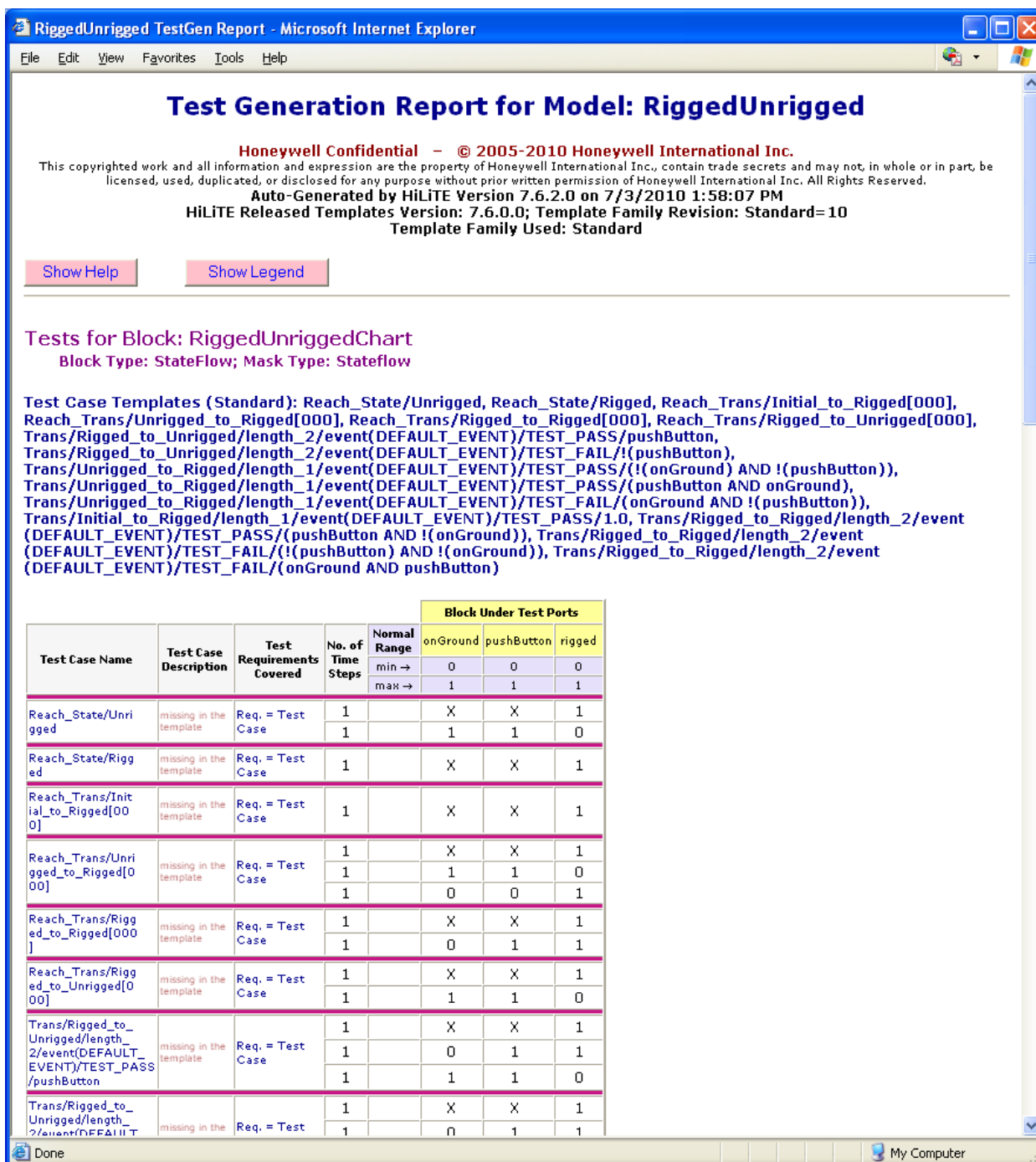


**Figure 52 - Partial view of RiggedUnrigged\_StatusReport.html**

The report lets you know that all necessary data was included in the model and that all test cases ran to completion.

6. Double-click on **RiggedUnrigged\_Report.htm**. Your browser displays the HiLiTE test generation report as shown below in three figures.

The top portion of the report enumerates the tests. Note that there are three types of tests (State, Transition, and Condition); the test case names show the type of test.



**Figure 53 - Partial view of test generation report RiggedUnrigged\_Report.htm**

The first table (on the left below) in the report shows the input and output port values of the Stateflow block required for each test case. Note that all ports in the example above are boolean.

The second table, Generated Vectors, shows the input values of the parent model of the Stateflow block required for each test and the output values that should be measured at the nearest test point. These are the actual vectors for use with a test harness and are copied from the CSV files stored in the folder ..\RiggedUnrigged\RiggedUnrigged-HiLiTEOutput\Vectors.

Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Normal Range min → max →	Block Under Test Ports		
					onGround	pushButton	rigged
					0 1	0 1	0 1
Reach_State/Rigged	missing in the template	Req. = Test Case	1		×	×	1
Reach_State/Unrigged	missing in the template	Req. = Test Case	1 1		×	×	1 0
Reach_Trans/Initial_to_Rigged[00]	missing in the template	Req. = Test Case	1		×	×	1
Reach_Trans/Rigged_to_Rigged[000]	missing in the template	Req. = Test Case	1 1		×	×	1 1
Reach_Trans/Rigged_to_Unrigged[000]	missing in the template	Req. = Test Case	1 1		×	×	1 0
Reach_Trans/Unrigged_to_Rigged[000]	missing in the template	Req. = Test Case	1 1 1		×	×	1 0 1
Trans/Initial_to_Rigged/length_1/event (DEFAULT_EVENT)/TEST_PASS/1.0	missing in the template	Req. = Test Case	1		×	×	1
Trans/Rigged_to_Rigged/length_2/event (DEFAULT_EVENT)/TEST_FAIL(!pushButton) AND !(onGround))	missing in the template	Req. = Test Case	1 1 1		×	×	1 1 1
Trans/Rigged_to_Rigged/length_2/event (DEFAULT_EVENT)/TEST_FAIL(!onGround AND pushButton)	missing in the template	Req. = Test Case	1 1 1		×	×	1 1 0
Trans/Rigged_to_Rigged/length_2/event (DEFAULT_EVENT)/TEST_PASS/(pushButton AND !(onGround))	missing in the template	Req. = Test Case	1 1 1		×	×	1 1 1
Trans/Rigged_to_Unrigged/length_2/event (DEFAULT_EVENT)/TEST_FAIL(!pushButton)	missing in the template	Req. = Test Case	1 1 1		×	×	1 1 1
Trans/Rigged_to_Unrigged/length_2/event (DEFAULT_EVENT)/TEST_PASS/pushButton	missing in the template	Req. = Test Case	1 1 1		×	×	1 1 0
Trans/Unrigged_to_Rigged/length_1/event (DEFAULT_EVENT)/TEST_FAIL(!onGround AND !(pushButton))	missing in the template	Req. = Test Case	1 1 1		×	×	1 0 0
Trans/Unrigged_to_Rigged/length_1/event (DEFAULT_EVENT)/TEST_PASS/(!(onGround) AND !(pushButton))	missing in the template	Req. = Test Case	1 1 1		×	×	1 0 1
Trans/Unrigged_to_Unrigged/length_1/event (DEFAULT_EVENT)/TEST_PASS/!(onGround)	missing in the template	Req. = Test Case	1 1 1		×	×	1 0 1

**Figure 54 - Input and output port values.**

signal name →			In1	In2	Out
Test Case Name	Time Steps	range	onGround	pushButton	rigged
		min →	0	0	0
		max →	1	1	1
Reach_State/Rigged	1		X	X	1
Reach_State/Unrigged	1		X	X	1
	1		1	1	0
Reach_Trans/Initial_to_Rigged[00]	1		X	X	1
Reach_Trans/Rigged_to_Rigged[000]	1		X	X	1
	1		0	1	1
Reach_Trans/Rigged_to_Unrigged[00]	1		X	X	1
	1		1	1	0
Reach_Trans/Unrigged_to_Rigged[0]	1		X	X	1
	1		1	1	0
	1		0	0	1
Trans/Initial_to_Rigged/length_1/event (DEFAULT_EVENT)/TEST_PASS/1.0	1		X	X	1
Trans/Rigged_to_Rigged/length_2/event (DEFAULT_EVENT)/TEST_FAIL(!pushButton AND !onGround))	1		X	X	1
	1		0	1	1
	1		0	0	1
Trans/Rigged_to_Rigged/length_2/event (DEFAULT_EVENT)/TEST_FAIL(onGround AND pushButton)	1		X	X	1
	1		0	1	1
	1		1	1	0
Trans/Rigged_to_Rigged/length_2/event (DEFAULT_EVENT)/TEST_PASS(!pushButton AND !onGround))	1		X	X	1
	1		0	1	1
	1		0	1	1
Trans/Rigged_to_Unrigged/length_2/event (DEFAULT_EVENT)/TEST_FAIL(!pushButton)	1		X	X	1
	1		0	1	1
	1		1	0	1
Trans/Unrigged_to_Rigged/length_1/event (DEFAULT_EVENT)/TEST_FAIL(onGround AND !pushButton))	1		X	X	1
	1		1	1	0
	1		1	0	0
Trans/Unrigged_to_Rigged/length_1/event (DEFAULT_EVENT)/TEST_PASS(!onGround AND !pushButton))	1		X	X	1
	1		1	1	0
	1		0	0	1
Trans/Unrigged_to_Rigged/length_1/event (DEFAULT_EVENT)/TEST_PASS(pushButton AND onGround)	1		X	X	1
	1		1	1	0
	1		1	1	1

**Figure 55 - Generated test vectors**

## Section 6 Generating Stateflow Test Cases

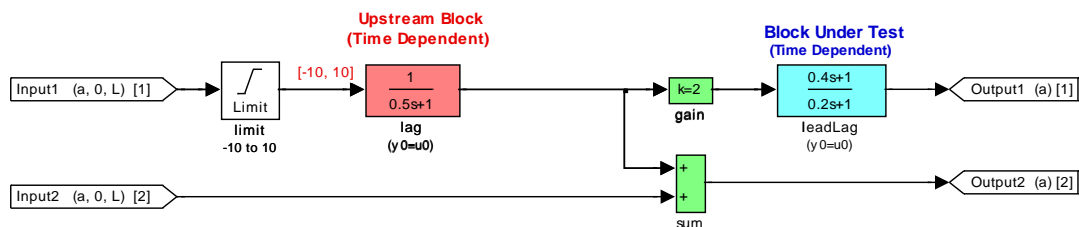
(This page intentionally left blank)

## 7 Improving Test Generation Coverage Using Additional Observation and Stimulation Points

Section 5.4.3 described observation points and how you can make explicit use of them for measuring expected output. This section describes how to use additional observation and stimulation points to enable HiLiTE to generate the test cases when model complexity makes test generation difficult for the tool. The approach complies with DO-178B objectives of verification that the executable object code (EOC) complies with low-level requirements and is described more fully in Ref. [1].

### 7.1 Using Secondary Stimuli

The method for propagation of block-level test cases at the model level, described in Ref. [1], can run into difficulties for complex model situations. Figure 56 illustrates a model situation where HiLiTE is not able to generate test cases for the *leadLag* block that test the transfer function of the filter over multiple time steps. The difficulty arises in backward propagation through the upstream block *lag* which itself is time dependent and its input is constrained to range [-10, 10].



**Figure 56 - Time dependent blocks in series with limits on input – difficulty in model-level test generation**

### 7.1.1 Definitions

**Primary Means of Stimulus.** These are the external inputs to the unit (component) under test that will be exercised when the unit is integrated with other units within the system. In this case, the unit is the code for the model. It is desirable that only the model inputs be used in creating test cases for a model's blocks. Thus, model inputs are the *primary means of stimulus*.

**Secondary Means of Stimulus.** Secondary means are ways of injecting test stimulus to the unit in an alternative manner from the model's inputs to test specific LLTRs (low-level test requirements) of blocks within the model.

- Secondary means shall be used only when primary means are not effective in creating a test case for an LLTR.
- Only approved secondary means shall be used – we will require concurrence from both OBARs and ARs prior to a secondary means being considered “approved”.
- Secondary means are applied to executable flight code as is, without requiring any modifications.
- Secondary means will only be used to test software downstream/outside of the stimulus.

### 7.1.2 Approach

When HiLiTE is not able to generate tests for one or more LLTRs, one or both of the following two options will be used, depending upon the model and specifics within the model:

**Optional1—Use one or more approved means of secondary stimulus** as a test injection method (to test downstream/outside of the injection point). Proposed secondary stimulus means:

1. Use the past value of the unit delay block as a stimulus input. Each unit delay block in the model is represented by an externally (privately) visible global variable in flight code that represents the feedback coupling one execution of the unit to the next. Its “past value” is set by the flight code at the end of the unit's execution and is used (read) by the code in the next execution of the unit. The past value can be manipulated externally in between two executions of the unit to create a secondary stimulus.
2. Insert a *Stimulate* block (from the HAM Block Library) in the model at the point where secondary stimulus is desired. This modified version of the model is used to generate the flight code; i.e., the Stimulate block becomes a permanent (designed) part of the model. The stimulus is applied by overwriting externally (privately) visible global variables in the code that represent constants inside the Stimulate block.
3. Link in an alternative form of the S-Function library that contains “pass through plus a test point block”.

**Note:** Flight code will NOT be modified to use the secondary stimulus means AND secondary stimulus will only be used during unit test on the model's code to test software downstream/outside of the stimulus.

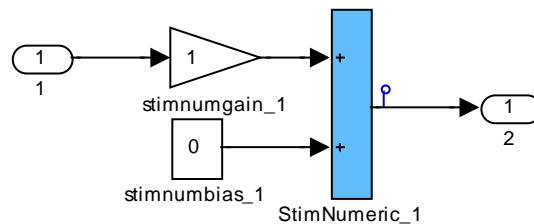
**Option2—Manually create test vectors to augment the HiLiTE-generated vectors.** These test vectors will have to be manually reviewed with a formal process using a checklist. The trade-off between the two options is that inserting many Stimulate blocks in a model can have adverse impact on the throughput whereas manually creating a test vectors has higher associated costs.

### 7.1.3 Secondary Stimulus Usage Examples

Two stimulation blocks (*Stimulate Numeric* and *Stimulate Boolean*) are provided in the HAM block library to create a signal override for ease of testing downstream blocks.



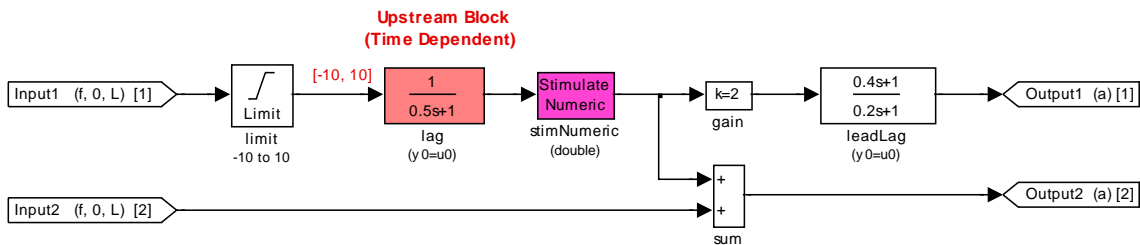
**How Stimulate Block Works.** In operation in the target system (flight code), the Stimulate block works as a transparent pass-through block; i.e., it flows its input signal unaltered to its output. However, its behavior can be controlled during unit test of the software for a model by the test harness (CTP) using externally visible variables. Figure 57 illustrates the internal construction of the Stimulate Numeric block. The values of *gain* and *bias* constants are generated in the code as global variables with default values of 1 and 0 respectively. By setting the variable *gain* to 0 and *bias* to the desired value in a particular step of a test case, the stimulate block can be made to override the value coming on its input (from the upstream block) and put the desired value at its output. In other words, when the override capability of a stimulate block is used, the backward propagation of a test case for a downstream block stops at the stimulate block and the value required by the test case can be supplied using an external variable of the stimulate block. Note: The values used for the override are still subject to upstream range constraint as discussed in the policies.



Note: The default values for embedded parameters "stimnumgain\_1" and "stimnumbias\_1" are only modified in a test environment.

**Figure 57 - Internal Construction of Stimulate Numeric Block**

**Usage Example.** Consider the example of Figure 56 which illustrates the difficulty of test generation when multiple time-dependent blocks are connected in series. Figure 58 shows a block called *stimNumeric* that is inserted after the *lag* block in so that the test case for block *leadLag* can be generated without the need for backward propagation through the lag block. Note that test cases for the *gain* and *sum* blocks will be generated without using the stimulation capabilities of the *stimNumeric* since the *gain* and *sum* blocks are not time dependent and their test case generation does not require secondary stimulus.



**Figure 58 - Use of Stimulation Block to Ease Test Generation of Downstream Block leadLag**

## 7.2 Using HiLiTE's Suggestion Capability

HiLiTE includes a capability for suggesting secondary stimuli—observation points and Stimulate blocks—when it has difficulty generating test cases. Users can use this capability for experimenting with what-if scenarios to determine where to place additional observation points and stimulation points in a model to improve the test generation coverage provided by HiLiTE. Note that this capability is intended solely for user experimentation and is not a qualified aspect of the tool; thus the outputs of this HiLiTE capability cannot be directly used to meet any certification objectives.

The suggestion capability is enabled through the `TestGenDirectives` OptionSet, using the option key `MakeStimAndObsPointSuggestions`. Include these parameters in the Command file:

```
<OptionSet name="TestGenDirectives">
  <Option key="MakeStimAndObsPointSuggestions">On</Option>
  ... other options ...
</OptionSet>
```

HiLiTE then includes suggestions in *modelname\_Summary.xml* file and the HiLiTE console log.

Note these points concerning the suggestion capability:

- Do not include any source code related options (e.g., generation of HiLiTE Test Harness (HTH) code or TPI files) when using the suggestion capability.
- Do not include CTP Output generation options.
- You can use range files for model inputs, but they are not necessary. If input range files are supplied, then the ranges do not have to be accurate.
- Do not use or execute HiLiTE output test vectors that result from the suggestion capability on the code.
- Make sure you take advantage of existing observation points (test points) in the model—ensure that no option key values will turn off (disable) the *BlockTypeTestPoints* and *ModelSpecificTestPoints*. It is not necessary to have code generated with the test points turned on or off, since the model's code is not referenced by HiLiTE for making suggestions. Similarly, if you specify the *IsolateBlocks*, set it to On.
- Take advantage of unit-delay overrides and Stimulate blocks already in the model. In the *Signal Boundaries* OptionSet, set the Option keys *AllowOverrideOfUnitDelayPastValue* and *AllowStimulationBlockOverride* to On.

```
<OptionSet name="Signal Boundaries">
  <Option key="AllowOverrideOfUnitDelayPastValue">On</Option>
  <Option key="AllowStimulationBlockOverride">On</Option>
</OptionSet>
```

## 7.2.1 Examining HiLiTE's Suggestions

HiLiTE's observation point suggestions are given in the *modelname\_Summary.xml* file. Even with *BlockTypeTestPoints* turned on, several blocks (such as Multipoint Switch, S-Function blocks) do not have any observation points. It is always less intrusive to add model-level observation points to help generate tests. HiLiTE performs a comprehensive analysis and experimentation with alternative observation points to decide a minimal number of observation points (if any) required in a model. Always add these model-level observation points before adding any Stim blocks. Note that if a Stim block must be added, then an observation point is no longer needed in the same location (if HiLiTE suggested one).

**Observation point suggestions in *modelname\_Summary.xml*.** The observation point suggestions appear in the *modelname\_Summary.xml* file and the HiLiTE log. The following is an example output of HiLiTE in *modelname\_Summary.xml* file:

```
<!-- Suggestions for model-level Observation Points (Test Points) -->
<SuggestedObservationPoints>And2.01 </SuggestedObservationPoints>
```

**Stimulation point suggestions in *modelname\_Summary.xml*.** When the options for making suggestions are used, HiLiTE makes the suggestions for Stim Points in the *modelname\_Summary.xml* file and also in the HiLiTE log. The suggestions are in two XML elements as shown below.

```

<!-- Suggestions for all possible Ports at which Stim Blocks can be place
      (in order of importance) -->
<!-- Format for Port Name:
      blockname1.01 or blockname2.I2
      (0 = output port, I = Input Port) -->
<!--Note that a only a small fraction of these Stim
      points will actually be needed -->
<PossibleStimPointsDetails>
  <Port Name="lag. 01" Importance="102" />
  <Port Name="gain. 01" Importance="- 11" />
</PossibleStimPointsDetails>

<!-- Recommended Stim Points Suggestions that can be used in HiLiTE Command File -->
<!-- Note: user must experiment with combinations of
      subsets of these -->
< StimPointsSuggestions>lag. 01 </StimPointsSuggestions>

```

The *PossibleStimPointsDetails* element lists all possible Stim Point *Ports* in decreasing order of importance. The complete list is provided for reference only; very few Stim points are actually needed to generate all required test cases in a model.

The element ***StimPoints-Suggestions*** lists a subset of all possible Stim points in a format suitable for experimentation.

**Port Name Format.** A *Port* is either an input or an output port of a block at which a stimulation block can be placed. The format for naming a port is: `Hi erachi cal Bl ockName. [0|I]n` Where O denotes an output port, I denotes an input port and n is the port index. E.g., *lag. 01* denotes the first output port of block *lag. subsystem1/sum2. I2* denotes the second input port of block *subsystem1/sum2*.

## 7.2.2 Experimenting with Placement of Observation and Stimulation Points

Using the suggestion capability of HiLiTE described in the previous section, the user can obtain a list of places in a model where stimulation and observation points can be placed to improve test generation coverage of HiLiTE. The suggestion list provide by HiLiTE, however, can be much larger than an optimal or minimal set of stimulation points needed to improve coverage, especially in complex models. In many situations, it is desired to minimize the number of these points in a model to reduce adverse impact on throughput and model rework.

HiLiTE provides a capability to conduct what-if experiments with subsets of suggested stimulation and observation points to determine a minimal set to provide desired coverage. This experimentation is done by providing a list in the HiLiTE command file, without making any change in the model, as described below.

**Observation point placement.** If HiLiTE gives suggestions for additional observation points in a model, it is recommended that all of the observation points suggestions be implemented since their impact on throughput is very low. In what-if experiments, the user can instruct HiLiTE to assume specific observation points in the model by using the following option in the HiLiTE command file. E.g.:

```

<OptionSet name="Signal Boundaries">
  ... other options ...
  <!-- Assumed Observation Points, format: blockname1.01 blockname2.01 -->
  <Option key="ObservationPoints">And2. 01</Option>
</OptionSet>

```

**Stimulation point placement.** The user should try what-if experiments on the model using a subset combination of the examples suggested in the element *StimPointsSuggestions*. Using a subset will minimize the number of Stim Points you need to obtain full possible requirements coverage. The *StimPoints* option key tells HiLiTE to assume a Stimulation Block at that point even though there isn't one in the model.

```

<OptionSet name="Signal Boundaries">
  ... other options ...
  <!-- format: blockname1.01 blockname2.I2 -->
  <Option key="StimPoints">product. 01 subsys2/sum2. 01 </Option>
</OptionSet>

```

One strategy for experimenting is to start with half the number of Stim Points listed in *StimPointsSuggestions*, run HiLiTE, and observe whether full requirements coverage is obtained with those points. If you do not get full coverage, read the file *modelname\_Summary.xml* to find out which additional Stim Points HiLiTE suggests and add those to the *StimPoints* option. In certain complex models, it may be useful to experiment with alternative subsets to find the optimum minimal set of Stim Points.

### SignalBoundaries OptionSet Summary

The following list further summarizes the SignalBoundaries OptionSet Option keys highlighted in the examples above.

*AllowOverrideOfUnitDelayPastValue*. The default Off value improves test generation coverage of models containing complex feedback loops. HiLiTE will not use unit delay overrides. This option is provided to improve the test generation coverage of models containing complex feedback loops. HiLiTE can automatically override (i.e., use as a secondary means of stimulus) the past value of unit delay blocks without requiring any change to code or a different version of flight code. To do this, HiLiTE uses externally accessible fields of the *D\_Work\_modelname* structure from the *modelname.h* file that represent the past value of the unit delay blocks to be used in the next execution of the model. Setting the key value to On directs HiLiTE to automatically override the past values of all Unit Delay and Frame Delay blocks in the model, as needed.

*AllowStimulationBlockOverride*. The default Off value indicates that HiLiTE will not allow stimulation block override. Setting the key value to On directs HiLiTE to automatically override the signal's upstream value using new HAM stimulation blocks. This option is provided to improve the test generation coverage of models. A signal's upstream value can be overridden using HAM *Stimulation* blocks. HiLiTE can automatically override an upstream signal value without requiring any change to code or a different version of flight code. To do this, HiLiTE makes use of externally accessible fields of the *Parameters\_modelname* structure, in the *modelname.h* file, that represent the stimulation points.

Note that HiLiTE first attempts to generate each test case without overriding any of the stimulation blocks. Only if that is not successful, one or more override inputs are used for that test case.

*StimPoints* tells HiLiTE to assume a Stimulation Block at a specified point even though the model doesn't include one. This allows quick experimentation without modifying the model.

## 8 Using and Interpreting HiLiTE Output for Status and Errors

Each time you run HiLiTE, it generates output to the console/log (HiLiTE Shell interface or the DOS command window, and log file), several report files, and test vector files. The content of files containing test vectors and diagnostics test vector files is described in Sections 5 and 6. The intent of this section is to describe the HiLiTE output that denotes the status of HiLiTE execution (including model processing, analysis, and test case generation) and any errors encountered in that process.

The type of status/error information you get depends on the HiLiTE command (e.g., mode analysis, test generation) used and other attributes/options specified in the HiLiTE command file. The following HiLiTE outputs are produced:

- **Model Status Report:** In most cases, you will find all the information you need to determine success and how to correct errors by reading the model status reports that are delivered to your output directories.
- **HiLiTE Log:** This is an execution log that is produced on the console and in a log file. This log provides indication of major errors encountered during HiLiTE execution and additional information for diagnostic purposes.
- **Model Summary File:** This is a machine-readable file in XML format that provides HiLiTE execution and test generation summary statistics, and

### 8.1 HiLiTE Log in Console and Log File

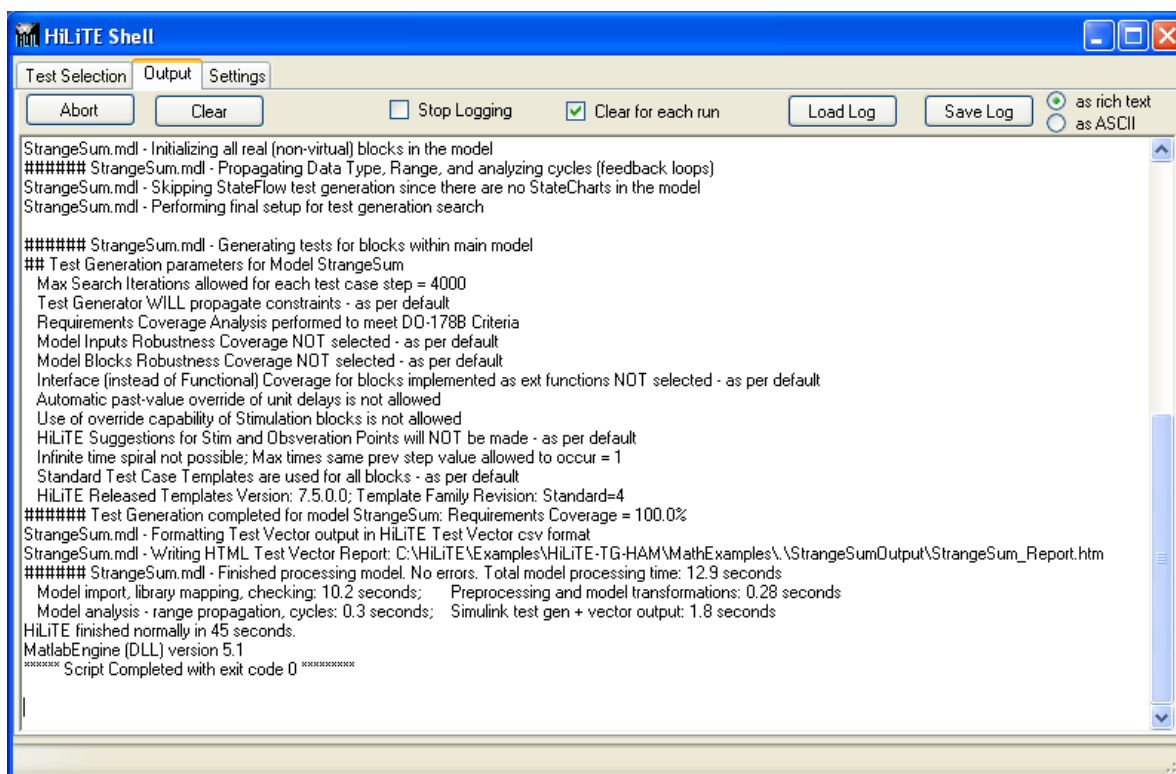
When you run HiLiTE either in a Microsoft command window or in HiLiTE Shell, it displays information on the console window to let you know its progress and any problems encountered. The same information is also replicated in a log file that HiLiTE creates. Users should note that the primary reference for status and errors is the model status report (Section 8.3). HiLiTE log should be reviewed by the user only for major errors that HiLiTE encountered and/or any specific error messages indicated below that are not

listed in the status report. A more common use of HiLiTE log is for diagnostic purposes (and not for formal test runs) as indicated later in this section.

**HiLiTE Log Locations:** HiLiTE log is produced in the console window and also in a log file; both have the same content in the released version of HiLiTE.

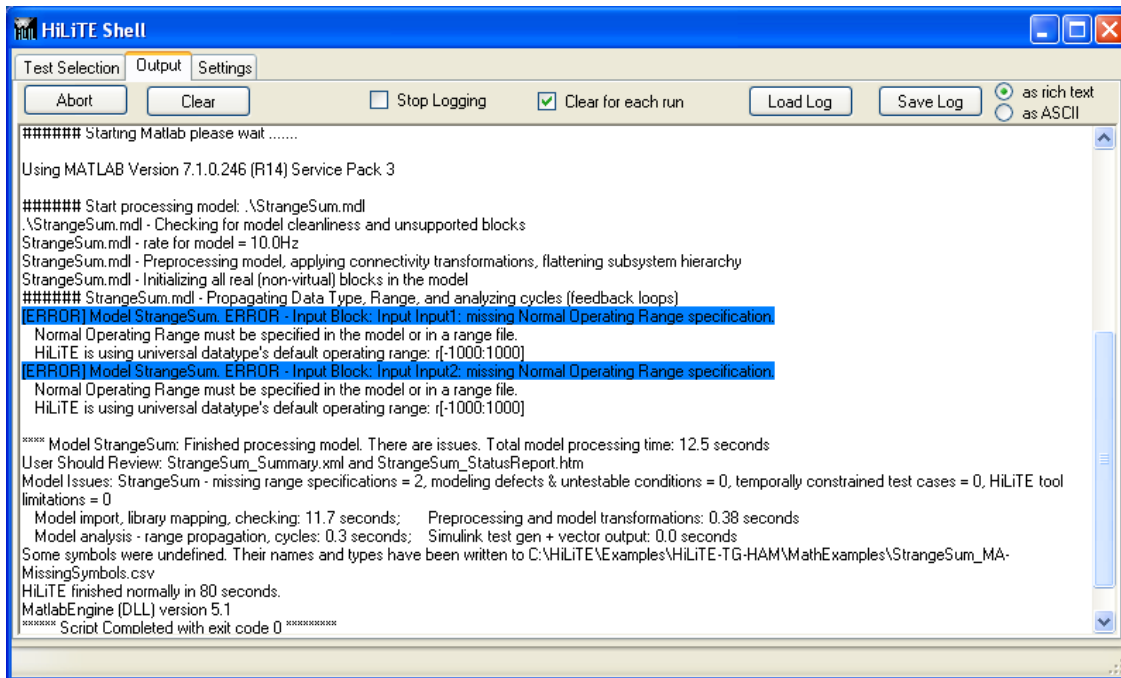
- **Console:** The console is the HiLiTE program's standard (and standard error) output that is produced in the window where HiLiTE is executed. Examples of console are the Microsoft command window, a UNIX-style shell window, or the output window of HiLiTE Shell. If HiLiTE is executed from a script then the console of script becomes the console for HiLiTE log. If the program output is redirected to a file then the console is represented by the contents of that file.
- **Log File:** This is a file named %HILITE\_HOME%\Build\bin\HiLiTE.log that is updated with log contents each time you run HiLiTE. Note that %HILITE\_HOME% refers to the HiLiTE installation directory.

The example in Figure 59 shows the log on the console (HiLiTE Shell output page) of a successful completion of test generation without major errors. Even though the run found no errors, you should check the *modelname\_StatusReport.htm* to verify different aspects of test generation such as potential testability limitations and test cases not generated by HiLiTE. A cautionary note is that an indication of success and/or absence of errors/warnings in the log do not preclude the review of model status report by the user.



**Figure 59 - HiLiTE Shell Output page showing successful test generation**

Figure 60 shows error messages (note the [ERROR] at the beginning of the lines) in the log indicating that the user did not specify operating ranges for some of the model inputs. Note that HiLiTE produces the same error message in the model status report and thus the message in the log is redundant; given that the primary output of HiLiTE to be reviewed by the user is the status report.



**Figure 60 - Example command window showing errors**

The following subsection provides detail on what type of error/warning messages should be investigated by the user and which should be ignored.

### 8.1.1 User Review and Disposition of Errors and Warnings in HiLiTE Log

HiLiTE log provides warning and error messages for user errors in the HiLiTE command file, the model, or in the execution environment. In addition to this information, these messages may occasionally contain diagnostics information about that can help the user diagnose testability limitations in the model or help identify future HiLiTE enhancements. Occasionally, the user may need to send copies of the log to HiLiTE support personnel for further diagnosis.

HiLiTE's general guidance is that the user should ignore any messages, other than those clearly intended for user attention. Some examples and specific cases are listed below:

#### Log Error Messages that Require User Review and Action

The following error message must be addressed by the user, they do not appear in the model status report since the status report may not even be generated in some of these cases or may be incomplete.

- Errors in command line arguments used to run HiLiTE
  - missing or incorrect name of HiLiTE command file
  - non-existent HiLiTE command file
- HiLiTE installation environment errors
  - MATLAB is not installed properly or is not accessible.
  - MATLAB license cannot be obtained.
  - a subassembly (DLL) of HiLiTE containing classes cannot be accessed or is not found
  - HiLiTE configuration data (e.g., templates file is missing)
- HiLiTE Command File errors
  - Illegal/miss-spelt name of a command file element

- Illegal value provided for an attribute or option
- Illegal order/nesting of command file elements
- Missing command file elements, e.g. missing Model element
- Name of a non-existent or inaccessible model file
- Following types of model errors
  - Error in loading of the model due to the model being too large or linked model files missing.
  - Malformed model; e.g., unconnected blocks, missing links, not a proper HAM model
  - Model contains blocks that are not supported by HiLiTE
- Certain errors in symbol files (note: some of these errors may be indicated in the model status report but the details are found in the log)
  - Name of a non-existent or inaccessible symbol file
  - Formatting errors in symbol files in rows and columns
  - Illegal content of a symbol column: e.g., range min value is not in numeric format
- Template Errors (in user modified test case templates)
  - Erroneous construction of a test case
  - Illegal template formula language (TFL) keywords used
  - Template values do not evaluate properly within constraints
- Errors in user supplied vectors (USVs) for StateFlow charts
  - Illegal row/column format of USV
  - An illegal USV that is not feasible in the chart
- Errors encountered in resolving test point and intermediate variables to model's code
  - The code referenced in the command file is not the correct version of code
  - Additional issues encountered by HiLiTE that require use of certain command-file options. HiLiTE indicates such issues specifically along with the options to be used.
- Major error in HiLiTE execution
  - HiLiTE may encounter a major error due to memory and file system issues and may not be able to complete model analysis and test generation. These errors are marked as such in the log as a major failure conditions.

### Spurious/Redundant Log Error Messages that should be Ignored by User

There are certain error messages in the log that are spurious and should be ignored by the user. Additionally, there are certain error messages that are redundant – i.e., they also appear in the model status report.

- Range-related errors and warnings; these are already present in the model status report
  - Missing ranges and data types for model inputs and feedback loops
  - Duplicate range specifications in symbol files
  - Range overflow conditions
  - Other range propagation errors
- Additional spurious data type warnings that should be ignored
  - The following type of error message:  
[ERROR] Port gain.O1: \*\*\*ERROR: propagated Data Type (Real, float64) is incompatible with specified data type LangIndep\_Float32 #  
[WARN ] Ignoring above warnings



### 8.1.2 Control of Logging by HiLiTE

HiLiTE's logging control information is stored in *HiLiTE.exe.config* file. In the released configuration of HiLiTE, the default (released) version of *HiLiTE.exe.config* is placed by the installer in the following location: "%HILITE\_HOME%\Build\bin".

The file contains specification of XML elements named "logger", where each logger denotes a particular category of logging messages that are generated by HiLiTE. The logging level for each logger can be independently controlled by setting the *value* attribute of the *level* element within the scope of the logger. The legal values are "FATAL", "ERROR", "WARN", "INFO", and "DEBUG", in decreasing order of the severity for the selection of information to be logged. The default setting for most logger elements is "ERROR", some loggers are set at WARN level.

Users are allowed to modify the *HiLiTE.exe.config* file for diagnostic/experimentation purposes. The only change permitted is to change the level of a logger to another value; changing to less severity will produce more diagnostic information in the log. As a safeguard against inadvertent modifications, *HiLiTE.exe.config* includes comments that explain which elements should not be modified. Modifying these elements may cause unknown errors.

**Additional Contents in the Log are Not Defined by HiLiTE.** When the levels of one or more loggers are modified, the content of the log generated by HiLiTE changes. HiLiTE ensures there is no change in the other outputs produced, including test case files and Status Report. However, there is no intent by HiLiTE to define or support the additional content generated in the log when a logger's level is changed to less severity. This additional log information should only be used for diagnostic purposes and should not be enabled in formal test runs.

#### Example Modifications of Logger Levels

**Model Analyzer.** Set this logger to INFO level produces detailed log of shape (dimension), data type, and range propagation information for every block and link in the model:

```
<logger name="HiLiTE.ModelAnalysis.ModelAnalyzer">
  <level value="INFO" />
</logger>
```

**User-Error Specific Loggers.** These loggers control the logging of user errors; e.g., errors in symbol files, command file, model, etc. The following specific loggers are defined with their default level values:

```
<logger name="HiLiTELogger.User.SymbolFile">
  <level value="WARN" />
</logger>
<logger name="HiLiTELogger.User.CommandFile">
  <level value="ERROR" />
</logger>
<logger name="HiLiTELogger.User.Templates">
  <level value="ERROR" />
</logger>
<logger name="HiLiTELogger.User.Model">
  <level value="ERROR" />
</logger>
<logger name="HiLiTELogger.User.Miscellaneous">
  <level value="ERROR" />
</logger>
```

## 8.2 Model Summary Report

HiLiTE generates an XML document that summarizes the HiLiTE execution of the model and lists any model-design defects found by HiLiTE. If you run one command for several models, each model will have its own summary file. The file is named *model name\_Summary.xml* and can be found in the Reports (as named in the TestOutput element of the command file) directory for the model.

**Note:** Model summary files are “unqualified” output of HiLiTE; intended only to give a snapshot of model success or failure. Do not use summary files to claim credit for a DO-178B objective. The contents of the fields may change with HiLiTE releases.

To display a model summary, open the file in your web browser (double-click the file name) or any text editor. Figure 61 is an example of a model summary showing the results of a successful execution of a simple model.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Summary of HiLiTE execution on a model -->
<HiLiTESummary HiLiTEStatus="Normal">
  <!-- Information about the model -->
  <ModelInfo Name="BasicMath" UserRev="-" />
  <!-- Information about the HiLiTE execution run: HiLiTE version, date, and run-time (seconds) for the model -->
  <HiLiTERunInfo HiLiTEVersion="7.5.0.1" ModelStartTime="1/12/2010 11:06:33 AM" ModelRunTime="13.1" />
  <!-- Stage 0, Initialization: HiLiTE Initialization (including templates) before loading the model -->
  <Intialization HiLiTEStatus="Normal" />
  <!-- Stage 1, ModelLoading: Model Import from MATLAB and Loading inside HiLiTE -->
  <ModelLoading HiLiTEStatus="Normal" />
  <!-- Stage 2, ModelPreprocessing: Model Preprocessing and Flattening of subsystem hierarchy -->
  <ModelPreprocessing HiLiTEStatus="Normal" />
  <!-- Stage 3, ModelAnalysis: Model Analysis including data type and range propagation -->
  <ModelAnalysis HiLiTEStatus="Normal" />
  <!-- Stage 4, StateFlowTestCreation: Creation of Test Cases Inside each StateFlow chart in model, but not generated at the model-level yet -->
  <StateFlowTestCreation HiLiTEStatus="NotApplicable" HiLiTEMetric="0.0%" />
  <!-- Stage 5, SimulinkTestGeneration: Test Case Generation at the Simulink (model) level, including test cases for any StateFlow charts -->
  <SimulinkTestGeneration HiLiTEStatus="Normal" HiLiTEMetric="100.0%">
    <RequirementsCoverage HiLiTEStatus="Normal" HiLiTEMetric="100.0%" />
    <TotalNumebrOfBlockInstances>4</TotalNumebrOfBlockInstances>
  </SimulinkTestGeneration>
  <!-- Stage 6, UserSpecifiedVectorFileOutput: Test Vector File output (if selected) in User-Specified Format -->
  <UserSpecifiedVectorFileOutput HiLiTEStatus="NotApplicable" />
  <!-- Stage 7, HiLiTEVectorFileOutput: Test Vector File output in HiLiTE Test Vector Format (.csv) -->
  <HiLiTEVectorFileOutput HiLiTEStatus="Normal" />
  <!-- Stage 8, TestHarnessCodeGen: HiLiTE Test Harness Code Generation (if selected) -->
  <TestHarnessCodeGen HiLiTEStatus="NotApplicable" />
  <!-- Listing of Model Defects, if any -->
  <Defects />
</HiLiTESummary>
```

Figure 61 - Model summary for BasicMath example

### 8.2.1 Model Summary Elements

The model summary includes elements that indicate whether and to what extent HiLiTE completed actions from the command file. Each element tag indicates the action and a *HiLiTEStatus*. Some action tags also indicate a *HiLiTEMetric*.

**HiLiTESuccess.** Attributes for this tag are:

*Normal* status implies that part of the HiLiTE command worked normally without encountering any failures.

*Error* or *Failure* status signifies a major problem in execution.

*Unknown* status should be ignored—it typically implies HiLiTE features that could have been used but were not included in the HiLiTE command file.

*NotApplicable* status implies that this aspect of HiLiTE execution does not apply to the particular model.

**HiLiTEMetric.** This attribute shows a percentage. It is specific to the XML element in which it is embedded, and its interpretation depends upon the XML element tag. This attribute is primarily used in regression testing to compare the metrics against previous runs.

#### 8.2.1.1 SimulinkTestGeneration

The SimulinkTestGeneration tag shows whether the model produced Simulink tests, the number of block instances tested, and how well it met requirements. Here, the HiLiTEMetric subelement indicates the success of test case generation. **Hi Li TE Metric = 100%** indicates that HiLiTE successfully generated test cases/vectors for 100% of the test case templates specified for library blocks at the Simulink model level. This is a tracking metric for regression purposes only.

For the *RequirementsCoverage* sub-element: HiLiTEMetric indicates the informal coverage of the library blocks' requirements by test cases generated by HiLiTE.

Note: If HiLiTEMetric in RequirementsCoverage sub-element is less than 100%, some tests for some requirements were not generated. You should review the *modelname\_StatusReport.htm* to see which requirements were not tested.

#### 8.2.1.2 StateflowTestCreation

The StateflowTestCreation element reports the metrics for each Stateflow chart in the model. Note: **Hi Li TE Metric = NotApplicable** indicates the model does not have a Stateflow chart in it.

The StateflowTestCreation element contains subelements for each chart. Each Chart element includes the name of the chart and the combined percentage of coverage tests that were successfully generated. The StateCoverage element within the Chart element lists the percentage of states that were successfully activated. The TransitionCoverage element lists the percentage of transition segments that were fired. The ConditionCoverage element lists percentage of condition tests that were generated. If the HiLiTESuccess in each of the sub element is less than 100%, this implies that some tests for some of the coverage (State/Transition/Condition) were not generated. The user must then review the *modelname\_StatusReport.htm* to see which requirements were not covered. The TotalTestCases attribute on each of the coverage elements indicates the total number of tests that should have been generated (the denominator of the percentage calculation used for HiLiTESuccess).

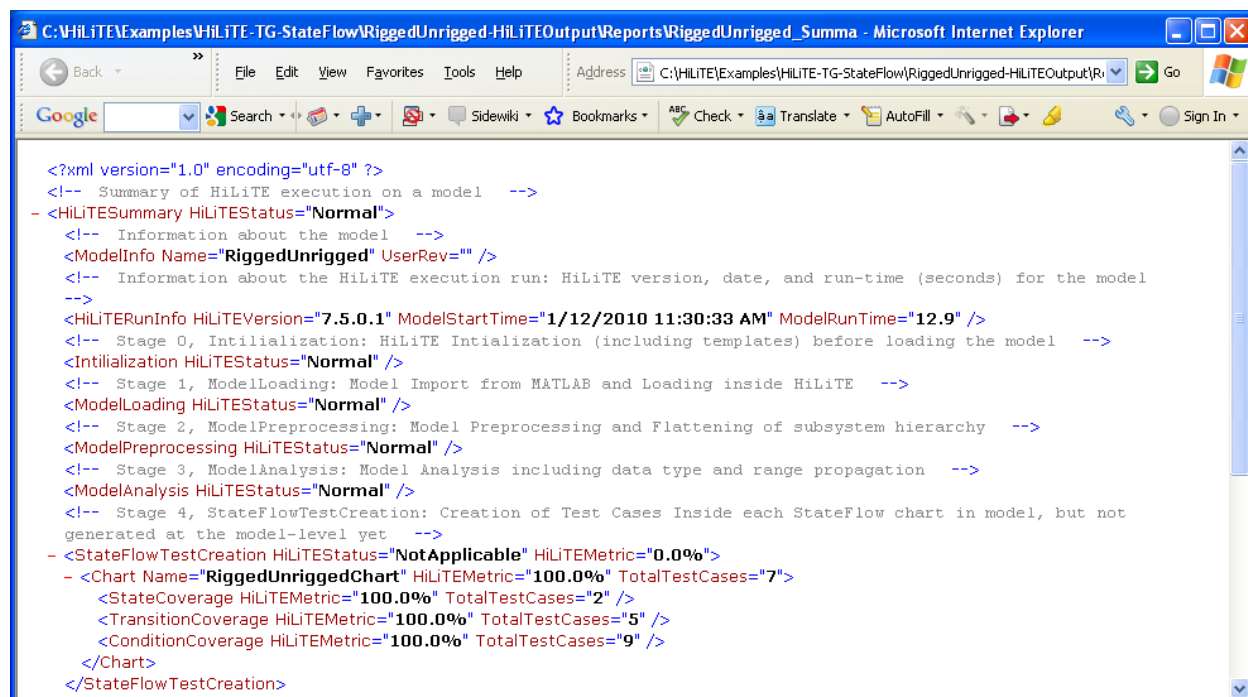


Figure 62 - Model summary for RiggedUnRigged.mdl showing StateflowTestGen status

### 8.2.1.3 Model Defects

As part of test generation, HiLiTE analyzes the model for design/testability defects, which are listed under this element. Figure 63 is an example of a model summary with Defects listed. User should ignore the attributes Status of and HiLiTESuccess; for defects reporting, these attributes have no meaning.

The *Defect* sub-elements list the defects of each type found in the model. In this example, there is a *Defect* sub-element of Type FROZEN, containing a list of 5 defects – 3 as Simulink level defects and 2 as Stateflow defects; a Defect sub-element of Type MAJOR listing 1 defect; a Defect sub-element WASTED, containing listing 1 defect. Please refer to Section 9 for more information on model design defects.

```

<!-- Stage 1, ModelLoading: Model Import from MATLAB and Loading inside HiLiTE -->
<ModelLoading HiLiTEStatus="Normal" />
<!-- Stage 2, ModelPreprocessing: Model Preprocessing and Flattening of subsystem hierarchy -->
<ModelPreprocessing HiLiTEStatus="Normal" />
<!-- Stage 3, ModelAnalysis: Model Analysis including data type and range propagation -->
<ModelAnalysis HiLiTEStatus="Normal" />
<!-- Stage 4, StateFlowTestCreation: Creation of Test Cases Inside each StateFlow chart in model, but not
generated at the model-level yet -->
<StateFlowTestCreation HiLiTEStatus="NotApplicable" HiLiTEMetric="0.0%" />
<!-- Stage 5, SimulinkTestGeneration: Test Case Generation at the Simulink (model) level, including test
cases for any StateFlow charts -->
- <SimulinkTestGeneration HiLiTEStatus="Normal" HiLiTEMetric="100.0%">
  <RequirementsCoverage HiLiTEStatus="Normal" HiLiTEMetric="100.0%" />
  <TotalNumbrOfBlockInstances>22</TotalNumbrOfBlockInstances>
</SimulinkTestGeneration>
<!-- Stage 6, UserSpecifiedVectorFileOutput: Test Vector File output (if selected) in User-Specified Format
-->
<UserSpecifiedVectorFileOutput HiLiTEStatus="Normal" />
<!-- Stage 7, HiLiTEVectorFileOutput: Test Vector File output in HiLiTE Test Vector Format (.csv) -->
<HiLiTEVectorFileOutput HiLiTEStatus="Normal" />
<!-- Stage 8, TestHarnessCodeGen: HiLiTE Test Harness Code Generation (if selected) -->
<TestHarnessCodeGen HiLiTEStatus="NotApplicable" />
<!-- Listing of Model Defects, if any -->
- <Defects>
- <Defect Type="FROZEN" Count="3">
  <MsgText Block="BQT_Math_DivideGain/AfterGainConst" Port="" Index="[0]">All output ports are constant for this
vector index</MsgText>
  <MsgText Block="BQT_Math_DivideGain/GainConst" Port="" Index="[0]">All output ports are constant for this vector
index</MsgText>
  <MsgText Block="BQT_Math_DivideGain/GainConst_Out" Port="" Index="[0]">All output ports are constant for this
vector index</MsgText>
</Defect>
- <Defect Type="MAJOR" Count="1">
  <MsgText Block="BQT_Math_DivideGain/DivByZero" Port="2" Index="[0]">Block DivByZero: Divide by zero possible
since maximum possible range on denominator can not be determined</MsgText>
</Defect>
</Defects>
</HiLiTESummary>

```

Figure 63 - Partial view of Summary for a model with design defects at Simulink Level

## 8.3 Model Status Report

For each model, HiLiTE generates a file named *modelname\_StatusReport.html*, which lists:

- incompleteness/errors in the range and data type values (for model inputs and feedback loops) provided by the user
- errors in range computation for Statechart output ports which are very high
- the low-level requirements that could not be tested by the generated tests
- (if applicable/required by user) the robustness coverage that was not achieved with the generated tests

The Status Report is a “qualified” output of HiLiTE to be used as the sole evidence of HiLiTE execution status for a model. It is reviewed by the user for dispositioning or correcting any errors reported and for obtaining a list which low-level requirements were not tested and robustness coverage that was not obtained. The tester can then decide how to disposition each missing requirement or robustness coverage by creating additional manual test case(s), changing the model design, or dispositioning it as an un-testable situation.

Figure 26 is an example of a status report that indicates that there were no range and data type specification errors and that all requirements were tested.

**Major Status Report Sections.** The Status Report has several sections, as listed below and described in the following subsections.

- Header
- User Specification of Data Types and Normal Ranges

- Errors in Propagation of Data Types and Ranges (only when errors present)
- Test Generation Summary
  - Requirements Coverage
  - Test Generation Directives and other Options Used (only for options used in command file)
- Stateflow Charts Test Generation Summary
  - Stateflow Chart Requirements Coverage (State Coverage, Transition Coverage, Condition Coverage)
  - Stateflow Test Generation Options Used (only for Stateflow options used in command file)
- Test Generation for Blocks' Requirements
- Robustness Coverage of Model Inputs (optional)
- Test Points and Overrides Suggestions (optional, unqualified output)

### 8.3.1 Status Report: Header

The Header section of the status report displays the model name, model revision, HiLiTE version, date and time of report generation along with copyright and use information.

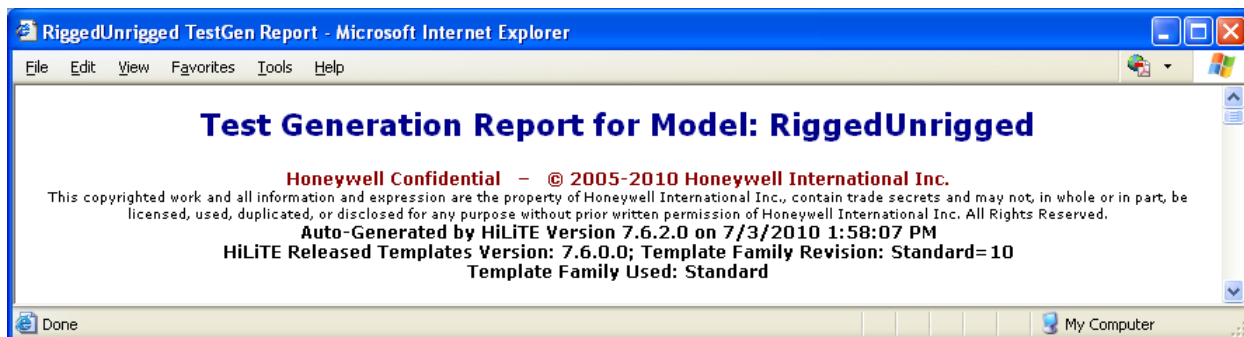
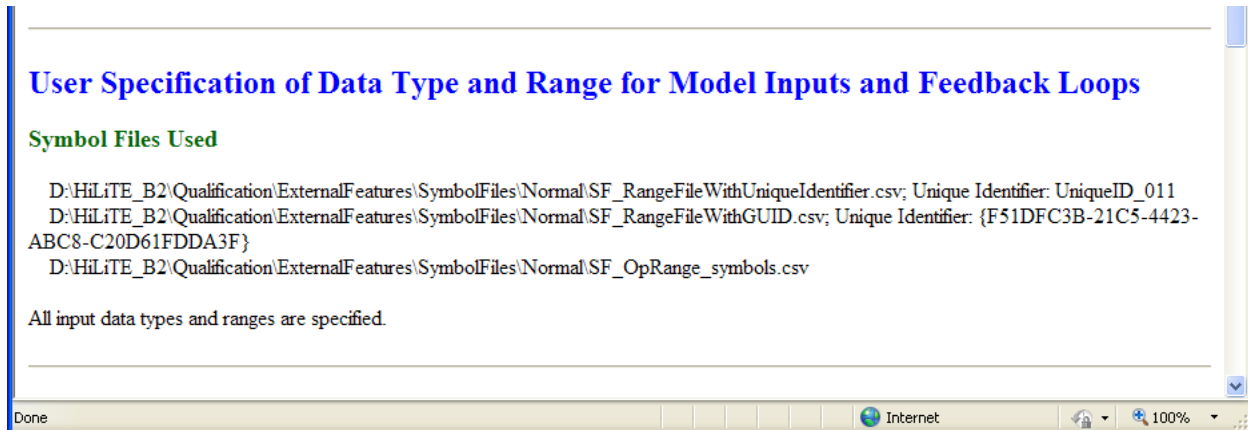


Figure 64 - Status report header

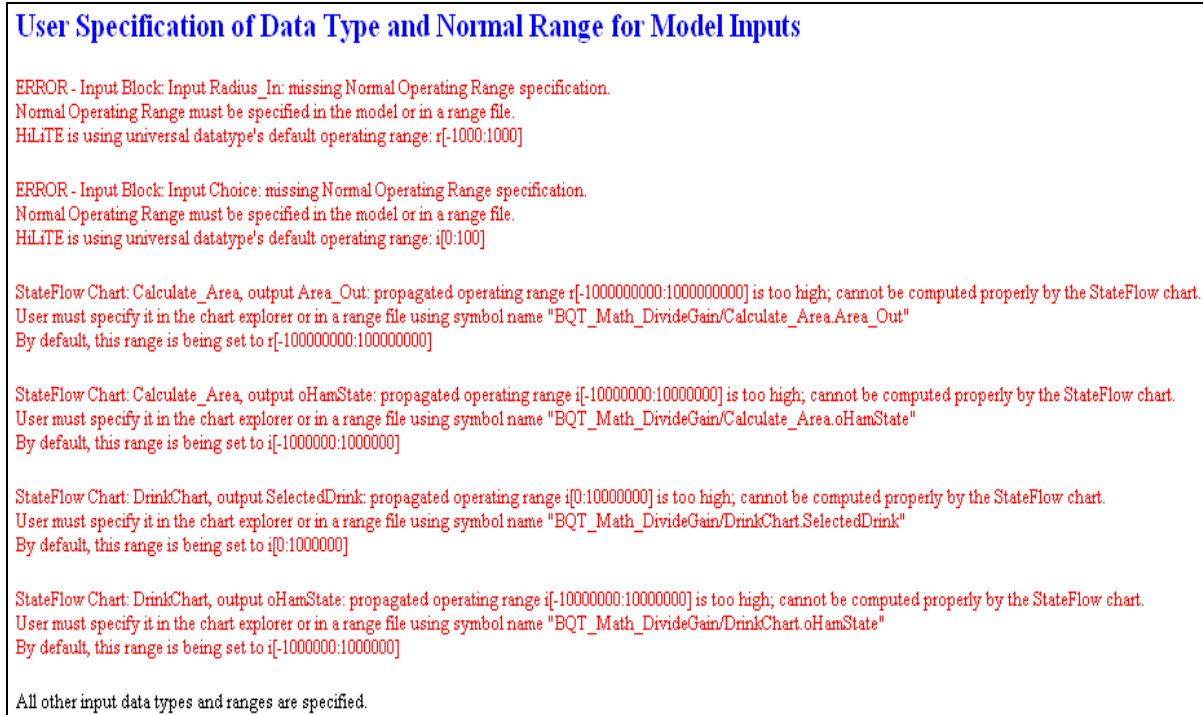
### 8.3.2 Status Report: User Specification of Data Types and Normal Ranges

This section lists the inputs for which data type or normal operating range is unspecified or incorrectly specified – based upon values provided in the model and in range files specified in HiLiTE command input. If data type and ranges are specified for all model inputs and feedback loops, then Status report file says that “All input data types and ranges are specified”, as shown in Figure 65. This area of the report also shows any unique identifiers included in the range files (refer to *Appendix B, Creating Range and Data Type Files* for more information).



**Figure 65 - Status report all data types and normal ranges specified**

If one or more ranges or data types are not specified, then those are listed in this section as errors as shown in Figure 66. Given that the user is notified of the specification errors, HiLiTE will use internal default normal operating ranges and generate tests.



**Figure 66 - Status report example: missing data types and normal ranges and error with range computation for Stateflow chart output ports**

### 8.3.3 Status Report: Errors in Propagation of Data Type and Ranges Thru the Model

HiLiTE will report any errors in data type and range propagation through the model (see Section 9.2). This includes overflow errors as shown below:

#### Errors in Propagation of Data Type and Ranges thru the Model

Block Uint\_Subtraction: Overflow - port Uint\_Subtraction.O1 range i[-2:15]: data type limits exceeded propagated operating range has been reduced to allow downstream computations



### Spurious Error Messages for Propagation of Data Types and Ranges

In some cases, as shown in Figure 67, HiLiTE generates a spurious error message (“Range propagation broken ...”) that you should ignore. For these cases, HiLiTE makes correct interpretations during test generation.

#### Errors in Propagation of Data Type and Ranges Thru the Model

```
[ERROR] Range propagation broken for block Logic/typecast1 v{0}, blocktype: DataTypeConversion
Propagated operating range b[0:1] extends beyond max allowable range {r[0]; r[1]}
```

**Figure 67 - Status report example: spurious range propagation error should be ignored**

### 8.3.4 Status Report: Test Generation Summary

The Test Generation Summary shows which version and family of the templates were used, percentage of simulink block requirements coverage attained, and the directives and options specified in the HiLiTE command file.

#### Test Generation Summary

HiLiTE Released Templates Version: 7.6.0.0; Template Family Revision: Standard=9  
Template Family Used: Standard

**Simulink Blocks Requirements Coverage = 100.000%**

(coverage computation based upon Template Family versions listed above)

#### Test Generation Directives and other Options Used

Functional Requirements Coverage computed for all blocks  
Blocks isolated as separate function units (e.g., StateFlow charts) will be tested using model-level tests  
Use of Unit-Delay Past Value override is not allowed as a secondary stimulus means  
Use of Stimulation Block Override is not allowed as a secondary stimulus means  
Standard Test Case Templates are used for all blocks  
HAM Block Level Test Points are On  
Model Level Test Points are On

**Figure 68 - Status report example: Test Generation Summary**

Simulink Blocks Requirements Coverage displays the percentage of Simulink level requirements that were covered. The computation and reporting of this coverage is in accordance with the HiLiTE command-file options specified for this model, as described in Section 1.1. In the figure above, all requirements test were completed successfully. Requirements coverage is computed as a percentage of successful tests in relation to actual test cases. For example, if ten test cases are present, but only five are completed, the coverage is 50%.

### 8.3.5 Status Report: Stateflow Charts Test Generation Summary

#### Stateflow Chart Requirements Coverage

Figure 69 shows the section of the Status Report that lists the requirement metrics for state, transition, and condition coverage achieved by HiLiTE for Stateflow charts in the model along with the test generation options specified in the HiLiTE command file.



## Stateflow Charts Test Generation Summary

### StateFlow Chart Requirements Coverage - RiggedUnrigged/RiggedUnriggedChart

State Coverage = 100.0% (2 of 2)  
 Transition Coverage = 100.0% (5 of 5)  
 Condition Coverage = 100.0% (9 of 9)

### Stateflow Test Generation Options Used

Backwards Search WILL be used to improve requirements coverage  
 Maximum search depth for backwards search is 5  
 Maximum number of iterations for timers in backwards search is 2000  
 No input vector files will be used  
 Stateflow TPI files will NOT be broken up into multiple files  
 Advanced expression simplification will be applied to expressions of size 10 or smaller

Figure 69 - Status report example: Stateflow charts test generation summary

## 8.3.6 Status Report: Test Generation for Block Requirements

This section of the Status Report displays a summary of the requirements coverage options for the model specified in the HiLiTE command file and the requirements coverage for each block instance in the model. The computation and reporting of this coverage complies with the HiLiTE command-file options specified for this model, as described in Section 1.1. Figure 70 is an example of the layout of this status report section.

### Test Generation for Blocks' Requirements

(Note: Any reporting of potential testability limitations for blocks is intended only as guidance for the user in determining the cause of missing requirements coverage.)

StateFlow Chart RiggedUnriggedChart: All Chart Internal Requirements Tested

StateFlow Chart RiggedUnriggedChart: All Functional Requirements Tested

Note: Requirements for this block are not separately captured; they are denoted by test cases

Figure 70 - Status Report: Test Generation for Blocks' Requirements - Layout

### Potential Testability Limitations

Reporting of potential testability limitations by HiLiTE for blocks is intended only as guidance in determining the cause of missing requirements coverage or exceptions during test case execution. In some cases, HiLiTE may over report model defects. Although some model defects manifest as overflow conditions, all overflow conditions in the model are reported exhaustively in another section (see Section 8.3.3) of the Status Report and the redundant reporting of those conditions as testability limitations should be ignored by the user.

User should apply the following method to address the potential testability limitations reported by HiLiTE:

1. If all test cases for block are generated and there are no exceptions during execution of those test cases, then the user should ignore the potential testability limitations for that block. The only exception are testability defects in certain categories listed in Section 9.3.
2. Otherwise, the user should manually analyze the reported testability limitations for that block to determine whether it is a valid limitation or an over reporting by HiLiTE. If it is a valid limitation, then the user should determine the following:
  - a. If this limitation results in an untestable condition then the missing test case can not be created; resulting in requirement/structural coverage shortfall. This is typically due to model design constraints and may require changes to the model or a disposition of the missing coverage.
  - b. If this limitation does not result in an untestable condition then the limitation can be used to guide manual test creation.

### 8.3.6.1 Block Requirements Categories

Block requirements fall into two categories: functional/interface and robustness. Block requirements are captured using the HiLiTE TemplateManager, as described in Section 0.

**Functional or interface requirements.** For the DO-178B testing objectives, low-level requirements represented by a model need to be tested on the object code for that model. A model's low-level requirements comprise the requirements for all the blocks in the model, where either *functional* or *interface* requirements for a particular block are applicable, but not both. For each block, coverage of generated tests is computed for either functional requirements or interface coverage, based upon the type of blocks and the HiLiTE command-file options.

- Functional requirements. These requirements are applicable to a block whose code is generated as part of a model's code. These represent the full normal-range functionality of a block that subsumes any interface functionality as well.
- Implicit Interface Requirements. If a block is implemented as a separately linked library function and the option *InterfaceCoverageForExtFunctions* is selected, HiLiTE computes interface coverage for that block, rather than functional requirements coverage, using implicit interface requirements as described below:
  - Coverage computation: HiLiTE tallies all tests generated for the model. The implicit interface requirements are that each input and output port of the block must appear in at least one test vector generated for the model. The coverage of different ports may be spread across multiple test cases; it is not mandatory that all ports of the external library block must be used within a single test case.
  - Coverage reporting: If interface coverage is missing for a block, then the ports on which the coverage is missing are listed in the status report. In addition, for guidance, the block's test cases that were tried but not generated are also listed. It is not necessary to recreate the same test cases to obtain this missing coverage.
- Explicit Interface requirements (optional, ignored). These requirements are applicable to a block that is implemented as a separately linked library function (e.g., S-function block, block in run-time library). Explicit interface requirements specification can be done using the Template Manager and is optional. If specified, these represents a subset of data and control coupling of the library function call for documentation purposes only. Explicit interface requirements captured for a block are for user's own documentation benefit and are ignored by HiLiTE.

**Robustness requirements.** In addition to functional or interface requirements, specific robustness requirements can be specified for certain blocks to meet robustness testing objectives. The coverage

achieved of these requirements is computed in addition to the functional or interface requirements, based upon the HiLiTE command-file options.

### 8.3.6.2 Block Requirements Coverage Reporting

For each block (instance) in the model, HiLiTE indicates whether a test case was generated that maps to the requirement for each of the relevant requirements of the block. The Status Reports shows whether any relevant requirement were not covered.

- If a block is vectorized, then each vector instance of the block is listed separately using this notation: *blockname[vectorIndex]*.
- For a particular block, coverage is reported on functional, interface, or robustness requirements (as described above), depending upon HiLiTE command-file options.
- Robustness requirements coverage is reported on a block only if HiLiTE command-file options selects robustness requirement coverage reporting for blocks and this block had one or more robustness requirements.
- Robustness coverage of model inputs is reported for the model inputs only if HiLiTE command-file options select the InputsRobustnessCoverage option. Note that the additional test cases generated for this coverage are named "At or Above Op Range Max" and "At or Below Op Range Min". These test case names indicate that from the perspective of the user program's DO-178B objectives, it is adequate to have the test case either at the boundary value or slightly above/below.
- Chart internal requirements coverage is reported for charts present in the model. It also reports whether the vectors for charts are generated at the model level or in isolation. (When the IsolateBlocks option is On in HiLiTE command file, the vectors are not generated at the model level; the report will show: "Note: model level tests are not generated for this chart.")

Figure 71 illustrates reporting of different types of block requirements coverage.

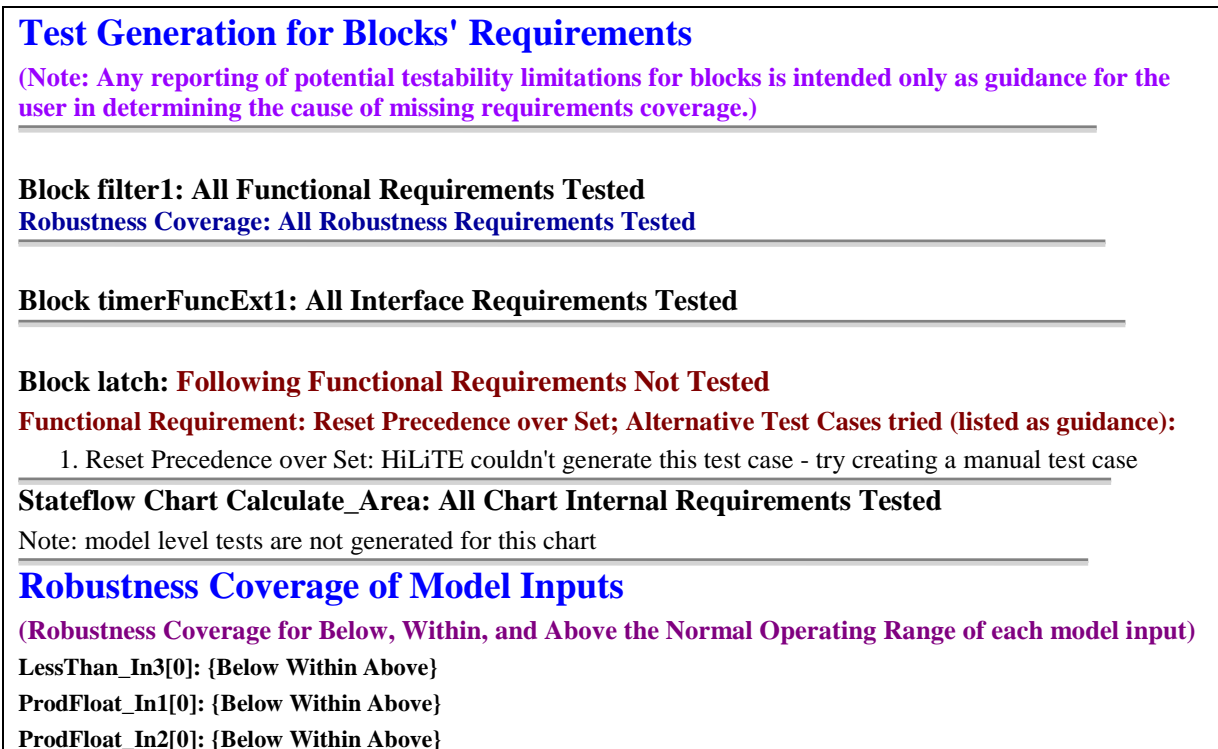


Figure 71- Status Report: Coverage Reporting of Different Types of Block Requirements

If requirements are not specified for a block, then HiLiTE will report this error in the status report indicating that the user must specify the requirements for this block in the template, as shown in Figure 72.

**Block VarLim2Const: All Functional Requirements Tested**  
 No requirements specified for this block - user must specify them; coverage is currently computed based upon test cases captured in template

**Figure 72 - Status Report: where no requirements specified are reported**

### 8.3.7 Status Report: Robustness Coverage of Model Inputs

When *ModelInputsRobustnessCoverage* option is turned “On” in the command file HiLiTE performs robustness coverage analysis of model inputs. Each model input must be used for a value within, below, and above its normal operating range in at least one test vector generated. If a specific robustness value is missing for an input, HiLiTE attempts to generate additional robustness tests for that input. The resulting robustness coverage achieved for each model input is listed in the “Robustness Coverage for Model Inputs” section, as shown in Figure 73.

#### **Robustness Coverage of Model Inputs**

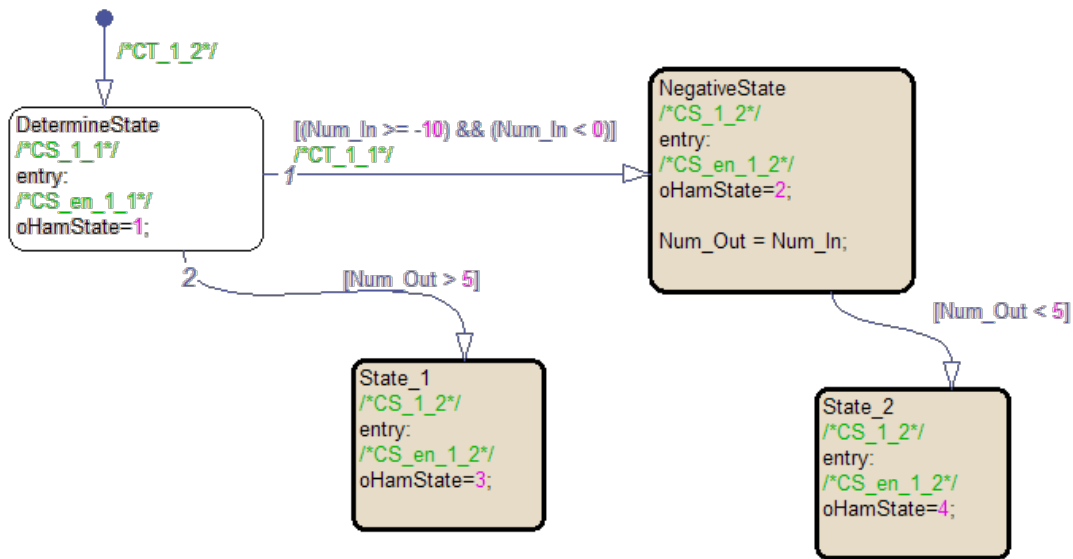
(Robustness Coverage for Below, Within, and Above the Normal Operating Range of each model input)

inputA[0]: {Below Within Above}  
 inputB[0]: {Below Within }  
 inputC[0]: {Within Above}  
 inputD[0]: non-numeric type

**Figure 73 - Status Report: Reporting of Robustness Coverage of Model Inputs**

### 8.3.8 Status Report: Stateflow Errors

The Stateflow test generator produces a writer report named *Chart name\_WriterReport.txt*. This report is a text document that lists every action that sets each local and output variable. It is useful for understanding any uninitialized value errors reported by HiLiTE. For example, in Figure 74 the value of Num\_Out is not initialized in the model explorer.



**Figure 74 - Example Stateflow chart**

The status report contains an error stating, “Tried to read an uninitialized value of Num\_Out. Assuming value of zero.” The writer report (Figure 75) can help find where the value is being read before it is written. Here we see that Num\_Out is written in only one place, NegativeState. Looking upstream from NegativeState, we can see that Num\_Out is read evaluating the transition from DetermineState to State\_1. Since this transition will be evaluated before entering NegativeState, we have identified a model error.

```

Num_Out ( INT32 ) : 1 : OUTPUT_DATA
  NegativeState
    ( Num_Out := Num_In )
  *****
oHamState ( INT32 ) : 2 : OUTPUT_DATA
  DetermineState
    ( oHamState := 1.0 )
  NegativeState
    ( oHamState := 2.0 )
  State_1
    ( oHamState := 3.0 )
  State_2
    ( oHamState := 4.0 )
  *****
  
```

**Figure 75 - Portion of writer report.**



## 9 Model Analysis

HiLiTE's model analysis capability analyzes Simulink/StateFlow models to determine ranges for signals/variables in the model and identifies overflow conditions and testability defects. This analysis capability enables the user to find design problems early in the development cycle and can be used for partial automation of design review checklists<sup>2</sup>. This model analysis capability is complementary to HAM Compliance Checker and other commercially available "style checker" tools. The following are the two main aspects of the model analysis:

1. Range analysis: Comprehensive propagation analysis of the operational ranges of intermediate signals and outputs of the model, given the ranges at the model inputs. This includes determination of hard range constraints (e.g. due to limiters) within the model.
2. Model design defects analysis: Analysis of certain model design defects based upon range analysis – including overflow conditions, frozen signals (constant value), un-testable conditions, and violation of modeling design guidelines or block-specific constraints.

This section describes user steps for analyzing a model, lists possible model defects, and gives simple examples of defective models. The section includes an example HiLiTE command file for analyzing the models and the output information that helps you understand problems.

### 9.1 Creating HiLiTE Command Files for Model Analysis

HiLiTE automatically performs model analysis when generating tests (TestGen command), but it also offers a command to perform model analysis alone. Making use of this command to analyze models, especially complex models, can save time by showing whether the models are ready to be used for generating tests. After HiLiTE performs the model analysis, it includes error information in the

---

<sup>2</sup> The current version of HiLiTE cannot be used to claim DO-178B credit for model design review; this is being addressed in future versions of HiLiTE.

*modelname*\_StatusReport.html. Additionally, based upon user option, HiLiTE generates a file that contains the ranges determined for model outputs.

Both the *TestGen* and *ModelAnalysis* commands will cause HiLiTE to perform model analysis. The command file parameters for model analysis alone are really no different from those for generating tests, although the command files may tend to be less complex. In essence, you replace the *TestGen* command with *ModelAnalysis*. Figure 76 gives an example of the most basic model analysis command file. Figure 77 illustrates the model named in the command file.

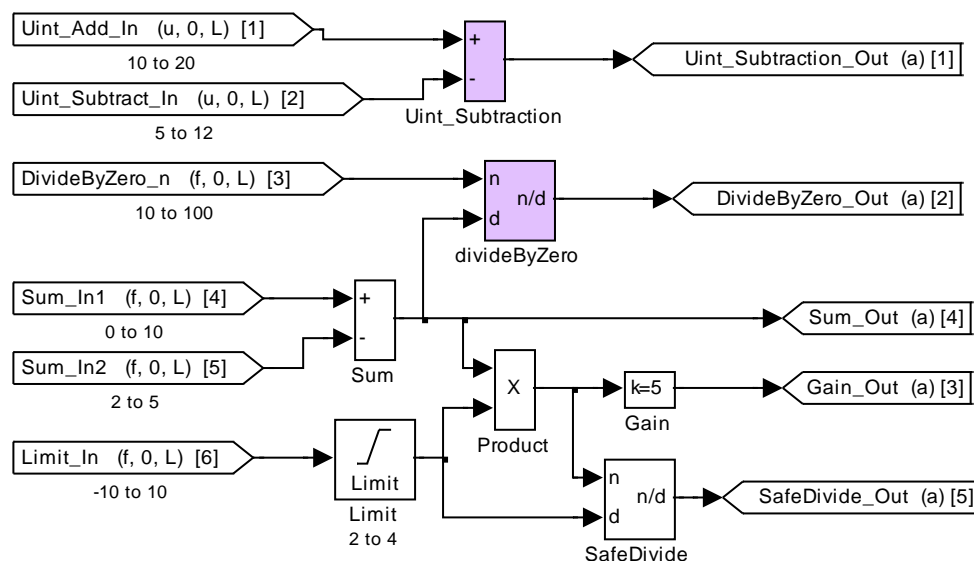
```
<?xml version="1.0"?>
<HiLiTEParameters>
  <ProjectPath>.</ProjectPath>

  <TestOutput>
    <Root><Args>1</Args></Root>
    <Category name="Reports">Reports</Category>
    <Category name="Vectors">Vectors</Category>
  </TestOutput>

  <Extension>HAM</Extension>

  <Command name="Model Analysis">
    <OptionSet name="Model AnalysisDirectives">
      <Option key="GenerateOutputRangeFile">True</Option>
      <Option key="OutputRangeFileInReportsDir">True</Option>
    </OptionSet>
    <Model name="RangePropExample"/>
  </Command>
</HiLiTEParameters>
```

**Figure 76 - HiLiTE command file for model analysis**



**Figure 77 - RangePropExample.mdl**

### Successful completion of Model Analysis

HiLiTE clearly indicates in the *modelname*\_Summary.xml file, in the *modelname*\_StatusReport.htm file, and in the log if there is a problem resolving the ranges or the data types of any block in the model. If no problem is reported then that indicates a successful completion of model analysis.



## 9.2 Data Type and Range Propagation Analysis

During model analysis, HiLiTE propagates data type and operating range information from the model input blocks (specified in the model and/or symbol files) through the rest of the blocks in the model to the model outputs. In this propagation, the data type, operating range, and hard range constraint (if any) is determined for all intermediate signals (all inputs and outputs of each block instance) in the model. The range computation takes into account the specific mathematical and functional effect of each library block. The following are computed for each output of each block:

- **Data Type:** including both the universal data type and language independent representation
- **Normal Operating Range:** the range of possible values the output can have, for all combinations of input values of the block within the normal operating range of each input.
- **Maximum Allowable Range Constraint:** (also called a hard bound or simply range constraint) It denotes a constraint on the feasible values due to specific block computation such as range limiting and/or range constraints on block inputs. Examples are outputs of the constant and range limiter blocks; values on these signals outside the constraint are not feasible in the model. Most signals in the model do not have a range constraint; i.e., all values within the data type limits are feasible for that signal.

Based upon the data type and range propagation results, HiLiTE performs the following analysis and defect checks:

- Detects potential overflow conditions; i.e., situations where the operating range on a signal exceeds the limits supported by the data type. This includes divide-by-zero situations.
- Detect model design and testability defects that are further described in Section 9.3.

### HiLiTE's Usage and Reporting of Range Propagation Results

HiLiTE uses the results of range propagation in generating the test cases – including references to normal operating range values from test cases templates and enforcement of range constraints. In addition, if the command option to generate an output range file is used, as in Figure 76, then a file is generated in CSV format that contains the propagated normal operating ranges for all model outputs (Figure 78).

	A	B	C	D	E	F
1	# Ranges of Model Outputs - Auto Generated by HiLiTE					
2	# Symbol Name	Normal Range Min	Normal Range Max	Data Type	Normal Range in Mathematica	Error Condition (if any)
3						
4	Uint_Subtraction_Out	0	15	LangIndep_UInt32	{i[0:15]Ovr}	Overflow
5	DivideByZero_Out	-500000000	500000000	LangIndep_Float32	{r[-5E+08:-2]; r[1.25:5E+08]}	Overflow
6	Gain_Out	-100	160	LangIndep_Float32	r[-100:160]	
7	Sum_Out	-5	8	LangIndep_Float32	r[-5:8]	
8	SafeDivide_Out	-10	16	LangIndep_Float32	r[-10:16]	
9						

**Figure 78. Output Range File for the example Model**

**Reporting of overflow if a normal range exceeds data type limits:** Any overflow errors are indicated in column L (Error Condition) of output range file if a normal range exceeds data type limits. If the propagated range from a block internal in the model exceeds data type limits, then HiLiTE will detect an overflow and propagate the overflow condition downstream to corresponding model output(s). It will be reported in the output range file in the column L as an upstream overflow, as shown in Figure 79 for the Boolean output GreaterThan\_Out.

	A	B	C	G	J	K	L	M	N
1	# Ranges of Model Outputs - Auto Generated by HiLiTE								
2	# Symbol Name	Normal Range Min	Normal Range Max	Producer Mod Changed for Vector	Error Condition (if any)	Specified	Propagate P		
3									
4	GreaterThan_Out	0	1	CF_RangeProp:Yes	1 Overflow - upstream	r[0:1]	Yes	-	
5									
6									
7									

**Figure 79. Indication of overflow upstream in the model from a Boolean output**

Overflow conditions are also reported in the model status report as results of range analysis, as shown in Figure 80. Please refer to Section 8.3 for more information on the status report. Additionally, original causes/locations of overflow (e.g., a divided-by-zero,  $\tan(90)$ ) are

#### **Errors in Propagation of Data Type and Ranges thru the Model**

Block Uint\_Subtraction: Overflow - port Uint\_Subtraction.O1 range i[-2:15]: data type limits exceeded propagated operating range has been reduced to allow downstream computations

Block divideByZero: Overflow - port divideByZero.O1: data type limits exceeded propagated operating range has been reduced to allow downstream computations

**Figure 80. Overflow error indication in the model status report**

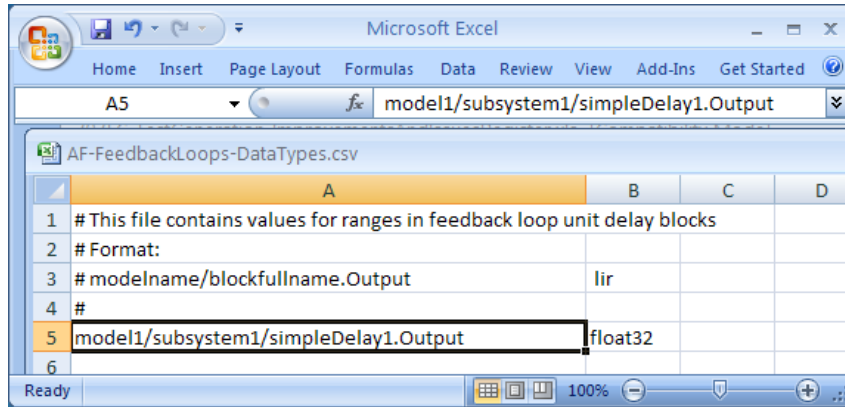
## **9.2.1 User Assistance needed for HiLiTE Data Type and Range Propagation**

### **Specifying Feedback Loop Data Type**

If HiLiTE cannot resolve a data type for a feedback loop then a warning appears in the status report. The user is asked to specify the data type for the feedback loop port similar to the feedback loop range specification. This can be done by adding a column in the feedback loop range file or creating a separate file for the data type. The following example illustrates the latter approach.

- The following is added in the HiLiTE command file:
 

```
<ColumnSpec specName="DataTypes" delimiters="," commentChars="#" titleLines="1">
  <Column name="symbol" col="0"/>
  <Column name="lir" col="1"/>
</ColumnSpec>
<SymbolFile fName="FeedbackLoops-DataTypes.csv" specName="DataTypes" />
```
- The following are the contents of example symbol file FeedbackLoops-DataTypes.csv:



### Using the ExtensiveModelAnalysis Option

For some complex models (especially if the model includes complex loops), it is difficult for HiLiTE to determine range, size, and data type at the output ports of all blocks. Setting the Option key value for *ExtensiveModelAnalysis* in the ModelAnalysisDirectives option set to *True* lets HiLiTE make this determination. If HiLiTE is unable to determine range, size, and data type for blocks in the model, the HiLiTE log file will suggest setting this option key to “True” or “False”, as the case may be for a particular model. You will see a message similar to:

```
[ERROR] ModelName: Data Type and Range propagation failed; test generation aborted.
[ERROR] Attention Hi Li TE Users!... ModelName Enable following option in Hi Li TE
command file...
<OptionSet name="Model AnalysisDirectives">
  <Option key="ExtensiveModelAnalysis">True</Option>
</OptionSet>
```

### Specifying Precise Ranges at the Outputs of StateFlow Charts

In certain models, HiLiTE cannot determine a precise range at an output of a Stateflow chart. HiLiTE uses a very high default value for further analysis and an error message (and related information) is provided in the model status report. User can specify a range for this output in a range file as input to HiLiTE, using the Stateflow chart name and the output port name. The symbol name used in the range file should be in the following format (note: model-name should not include .mdl extension):

<model-name>/<block-hierarchical-name-within-model>.<chart-output-name>

Note that users may have historically used range files containing symbols without the model name prefix, e.g.: <block-hierarchical-name-within-model>.<chart-output-name>. This usage can lead to ambiguities and erroneous application of ranges intended for another chart of the same name in another model. HiLiTE offers deprecated backward compatibility for this usage and will give the following warning messages in the log for models that do not comply with the format:

```
[WARN] Symbol File: Symbol ACE_Tx_Wrap_Test is being used without model name prefix.
User should always use model name as a prefix for symbols relating to block names internal to the model;
specification without model name can lead to ambiguities and is deprecated.
```

## 9.2.2 Special, Optional Range Checks

### Block Inputs Range Limit Check

HiLiTE provides a capability for the user to check if the instances of a certain type of block in a model have their incident input ranges in that model confined within certain limits. The user can specify the inputs range limits for certain block types in a CSV file to HiLiTE which then reports an overflow error if the ranges at inputs of a block exceed the limits for that block type. The following command option is used:

```

<OptionSet name="ModelAnalysisDirectives">
  <Option key="CheckBlockInputRangesAgainstLimits">True</Option>
  <Option key="BlockInputRangeLimitsFileName">BlockInputsRangeLimits.csv</Option>
</OptionSet>

```

The format of the file is as follows:

	A	B	C	D	E	F	G
1	# Mask Type	Num Inputs	Min	Max	Exclude Min	Exclude Max	
2	Sum (HW)	1	-1.00E+12	1.00E+12			
3	Product (HW)	1	-1.00E+10	1.00E+10			
4							
5							
6							
7							

#### Notes:

- If the same limit is to be applied to all inputs of the block, then *Num Inputs* should be 1.
- If different limits apply to different inputs, then all inputs should be counted (in *Num Inputs*) and the four columns (*Min*, *Max*, *Exclude Min*, *Exclude Max*) should repeat for each input.
- The columns *Min* and *Max* are required. The columns *Exclude Min*, *Exclude Max* must be present in all rows but can be left blank if some of the columns do not apply to a block (see the example above). Care must be taken in creating the CSV file to ensure this in Excel.

## 9.3 Detection of Model Design and Testability Defects

During model analysis, HiLiTE performs comprehensive range propagation as described in Section 9.2. Using the results of range analysis, HiLiTE performs a further analysis to detect and report design defects and/or testability defects in the model.

### 9.3.1 Categories of Model Defects

Model defects are loosely classified in the following categories. Note that these categories don't impart any special meaning – the defect message/details need to be examined to assess the impact of the defect.

#### MAJOR

There is a major design issue with this block or preceding logic in the diagram, or constraints on inputs, or uninitialized variables.

#### OVERFLOW

The output of this block has a computed normal operational range that does not fit within the data type limits. This can cause arithmetic exception in the model's code if those range values are reached. Possible divide-by-zero is an example of this.

***DEAD\_OR\_DEACTIVATED***

Some conditions make it impossible to execute parts of the code for a block or a Stateflow construct.

***FROZEN***

The output of this block/port is constant, but the block isn't a constant block. For all blocks/constructs this condition results in unnecessary code and inability to achieve requirements testing coverage. Additionally, for logic, relational, limiters, and switch types of blocks (e.g. blocks with decisions/branches in code), it is impossible to achieve structural coverage (Statement or Decision, MC/DC coverage)

***UNTESTABLE***

It is not possible to test all input values at a port or Stateflow variable used in a condition. This condition makes it impossible to achieve structural coverage (Statement or Decision, MC/DC coverage)

***IGNORED (obsolete)***

This defect code is obsolete now; older versions of HiLiTE may report it.

***WASTED***

This block could be replaced by something simpler, such as a direct connection between inputs and outputs. Wasted blocks don't have to have constant outputs—they just aren't needed.

***TYPE\_OR\_RANGE***

An issue in range or data type specification or propagation has been noted. E.g., illegal range value specified for a model input or an additional range specification is needed for a variable in a feedback loop

***INPUT\_REQ\_VIOLATED***

The normal operational range of an input to the block violates the block's input requirement (e.g., operational range of input to Sqrt is –ive). This can cause abnormal behavior in the operational envelope. Note that some blocks may have built-in protection mechanism to prevent abnormal behavior but the defect should still be addressed.

***INADEQUATE\_CONSTRAINT***

An input to the block is not adequately constrained upstream – i.e., its beyond-normal (abnormal) range can potentially violate a block's input requirement. This defect is less severe than *INPUT\_REQ\_VIOLATED* and may be disposed by a user as acceptable.

***ATTRIBUTE***

A block attribute problem was detected; the block attribute is an illegal value or outside guidelines, e.g., time constant (tau) of filter < 2 \* periodic rate.

***UNSUPPORTED***

This block type or feature isn't currently supported by HiLiTE

***STATEFLOW***

Additional problems were discovered during Stateflow analysis that does not fall into other categories.

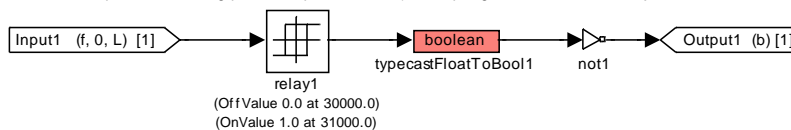
### 9.3.2 Types of Model Defects Currently Reported by HiLiTE

HiLiTE currently reports the following types of defects:

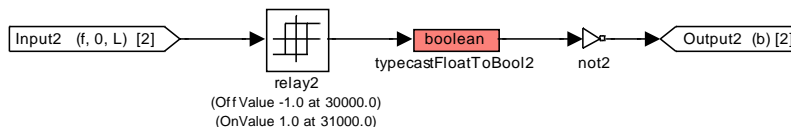
- Potential for overflow when the normal operating range of a signal in the model exceeds the limits of the data type (e.g., divide-by-zero)

- Constant (frozen) signals – i.e., the maximum possible range is constrained to a point value
  - Output of a comparator (e.g.  $\geq$ ,  $<$ ) is always true or false due to range constraints at its inputs
  - Output of a Typecast (float to Boolean) is always true due to floating point error.
  - Output of an and gate is always false when one input is fixed at false.
- Un-testable requirements/conditions (as per DO-178B requirements-based testing and structural coverage guidelines). E.g.,
  - Reset precedence over set condition for a latch cannot be exercised
  - Input to a logic gate is range-constrained to 0 or 1
- The control port of a switch or multiport switch has a constraint such that all code paths cannot be reached (dead/deactivated code)
- Typecast from float data type to Boolean: For this case, MATLAB RTW generates inappropriate code (of the form “input == 0.0”) that tests for floating-point equality. This can be non-deterministic due to floating point error in the input value and a true or false output value cannot be guaranteed deterministically on the target implementation. Note that model analysis, traditionally, is based only upon information in the model and not on unexpected bugs in code generation that violate basic coding guidelines. Therefore previous versions of HiLiTE did not detect this situation since this model scenario by itself cannot be considered defective; the code generator should have generated code that did not test for floating-point equality. However, this is an important situation to detect and therefore HiLiTE model analysis has been extended so that *known* inappropriate implementation (code generation) instances can be factored into model analysis. HiLiTE now generates appropriate model defect messages for the following:
  - Scenario 1: If the input consists of constant values only, including a 0 value, then the non-determinism at the output can be mitigated if the code contains a “0” as literal or the compiler/processor can guarantee no floating point error on assignments. HiLiTE detects this situation using range analysis and generates appropriate error message.
  - Scenario 2: If the input excludes a 0 value, then the output of typecast is always true. HiLiTE detects this situation using range analysis and generates appropriate error message.
  - Scenario 3: If the float input to the typecast block is continuously variable or a result of math operation, the non-determinism at the output is maximum and such defect notification is provided.

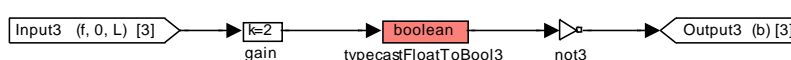
Scenario 1: The range coming into the input of the typecast block is set up (using relay block) to consist of two point values; but that is still prone to floating point comparison error (unless pedigree of constants in object code can be established) and that defect should be identified.



Scenario 2: The range coming into the input of the typecast block is set up to exclude 0; that implies that output of typecast is always true and this defect should be identified.



Scenario 3: The range coming into the input of the typecast block is unconstrained; but that is still prone to floating point comparison error and this defect should be identified.



Note: this is a bug in the MATLAB RTW code generator that HAM invokes. Until the code generator is fixed, the best recourse is to not allow this particular conversion in a model, unless compiler and target-platform code verification is done to provide a determinism guarantee. The HAM team and user groups should be made aware of this. Since it is difficult to detect this situation in the HAM Compliance Checker, user groups should be made aware to rely on HiLiTE to detect such situations.

For StateFlow charts, HiLiTE detects overflow condition, unreachable paths, and un-testable conditions over specification, including:

- Potential for divide-by-zero
- Unreachable states and transitions
- Untreatable conditions for mc/dc coverage
- Over or redundant specification in Truth tables that leads to mc/dc coverage issues in code
- Local data being initialized or written before being read – in all paths

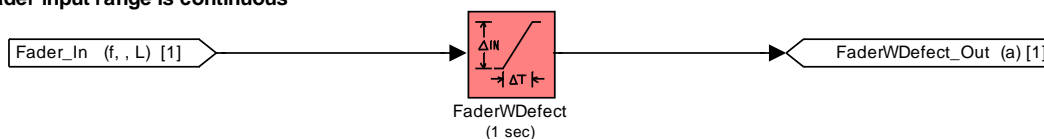
### 9.3.2.1 Checks for Violations of Design Review Guidelines

Violations of following design guidelines related to models and blocks are detected and reported. These items are taken from *Design Requirements Document (DRD) Inspection Checklist Version 10* used in Primus EPIC programs.

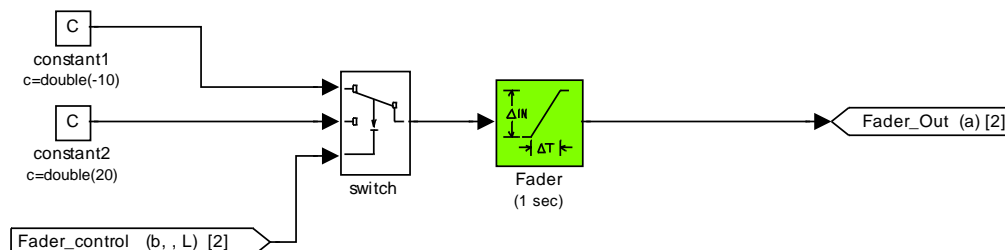
- **Filter blocks:** For Variable Lag and Variable Washout Filters, are the time constants at least twice the sample time of the model?
- **Sqrt block:** Are the square root blocks only taking the root of a positive number?
- **Power block:** Are power blocks, when taking an nth even root, only taking the root of a positive number?
- **Tan block:** For tangent functions, is the range of input angle used is not very near multiples of  $\pi/2$  and thus cause high numerical error and potential downstream overflow?
- **Atan2 block:** Atan2 inputs are not both zero?
- **Asin, Acos blocks:** Is input angle in the range of  $[-1 \leq \text{angle} \leq +1]$ ?
- **Model Outputs:** Are two or more model outputs not tied together?
- **Fader and Switched Fader blocks:** (Checklist item 3.14) Are fader blocks only used on continuous signals (i.e., only signals that change in discrete increments should be input to the fader)?

HiLiTE checks the maximum allowable range at the input(s) of Fader and Switched Fader blocks to ensure that it comprises only of discrete values. The rationale behind this check is that the functionality of a fader block is to convert discrete changes in the input to continuous changes at the output. If the input itself can vary continuously, the behavior of the block cannot be reasoned from this perspective and the use of the block would be unnecessary. Example scenarios are:

Fader input range is continuous



Fader input range comprises of discrete values

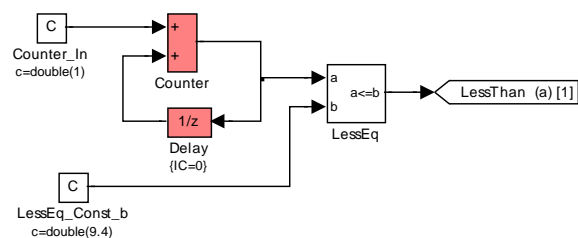


- **Counter Pattern Limits Check:** (Checklist item 3.25) Have all counters been limited (i.e., avoiding the use of non-limited counters)?

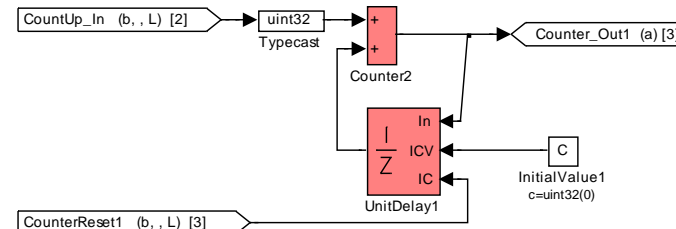
In the initial version of this capability, HiLiTE should analyze combination of blocks for simple counter patterns that can be with or without "limiter" types of blocks and can contain a reset input coming into the pattern. Future versions of this capability will detect several other more complex counter patterns based upon generalized analysis of feedback loops, which is a very difficult computational problem.

The following are examples of counters for which current version of HiLiTE can detect as being limited or not:

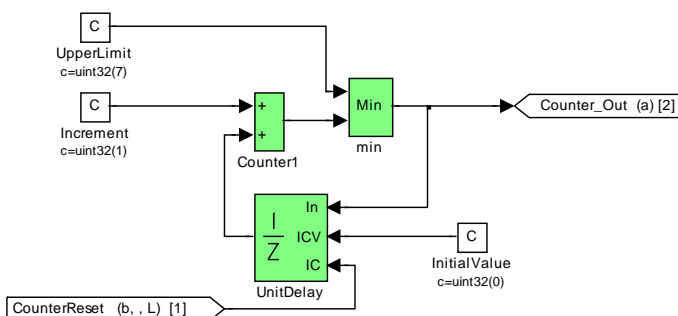
Example 1: Counter Pattern without Limits



Example 2: Resettable Counter Pattern without Limits



Example 3: Resettable Counter Pattern with Limits





### 9.3.3 Model Defect Reporting and Dispositions

#### HiLiTE Reporting of Model Testability Defects

HiLiTE reports model testability defects in the following places; identical messages are provided in each:

1. Model Status Report: This is the “qualified” output of HiLiTE that should be used/reviewed for formal test generation runs. Defects are reported as “potential testability limitations”. See also Section 8.3. (note: if the ModelAnalysis command is used, then defects aren’t reported in the status report, only in the log and model summary report).
2. HiLiTE Log: Log messages are generated for most model defects; note some defects may not be reported in the log.
3. Model Summary Report: Defects are reported in XML text format that can be read by scripts that create automated extraction/summarization of this information for diagnostic purposes.

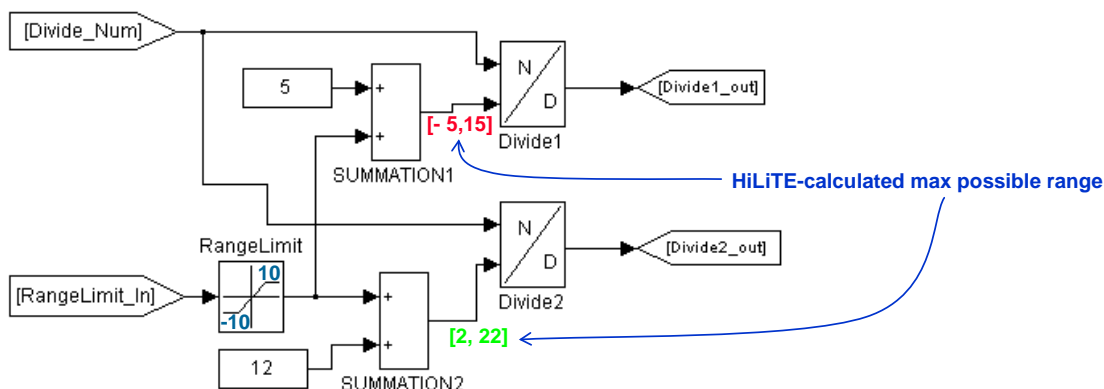
#### User Dispositions of Defects and Potential Over Reporting of Defects by HiLiTE

Model defects reported are for user guidance only to perform further diagnosis in cases where some requirements could not be tested. Since HiLiTE can sometimes over report models defects, the defects reported for a block should be ignored by the user if the requirement coverage for that block is complete. The exception is the following categories of defects which should always be further analyzed by the user: MAJOR, OVERFLOW, DEAD\_OR\_DEACTIVATED, FROZEN.

The model must be manually analyzed by the user for these above categories of defects (especially for a divide-by-zero defect) to ascertain whether it is a real defect or an over reporting by HiLiTE. If it is an over reporting then the defect report can be disposed off. Additional discussion on this can be found in Section 8.3.6.

### 9.3.4 Model Defect Examples

Figure 81 illustrates a model with a possible divide-by-zero defect that is detected by HiLiTE based upon the analysis of range constraints computed at the denominator input of a divide block.



**Figure 81 - Simulink Range Analysis Example: Detecting Divide-by-Zero Possibility**

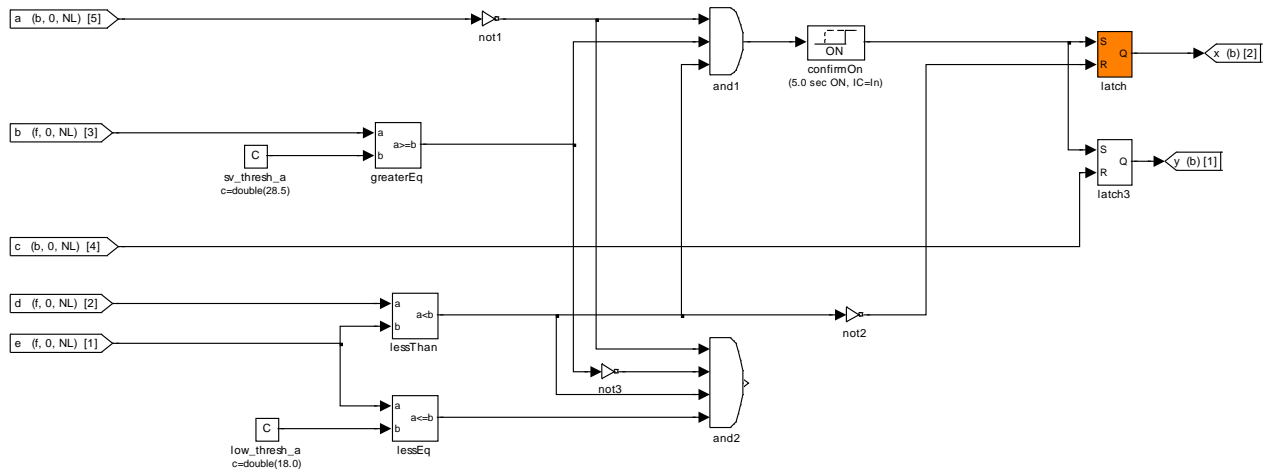
For this model, HiLiTE generates the following messages:

Block divideByZero: Overflow - port divideByZero.O1: data type limits exceeded  
propagated operating range has been reduced to allow downstream computations

[ERROR] Block divideByZero: Divide by zero possible since maximum possible range on denominator can not be determined

HiLiTE does not warn about Divide2, since its denominator's maximum range is [2,22].

Figure 82 illustrates an untestable condition.



**Figure 82 - Example of Untestable Condition (mutually exclusive conditions)**

HiLiTE reports that:

Reset Precedence over Set cannot be tested for Block named "latch" because of preceding logic starting at block "greaterEq". Both values can't be True at the same time.

## 9.4 Generating Output Range Files

You can generate a range file by including the `Model AnalysisDirectives` OptionSet with appropriate option-key values in your command file. HiLiTE will create a range file containing model output names and their operational ranges. The file also includes information about: HiLiTE version, User ID, mathematical format for the range value, and any range that was specified in a range file for that output. The precision (i.e., number of significant digits) of the range min and max values is based on the single or double precision floating point data type specified in the model or the HiLiTE command file. Figure 78 shows an output range file and some of its columns.

The basic OptionSet information to include in the command file looks like this:

```
<OptionSet name="Model AnalysisDirectives">
  <Option key="GenerateOutputRangeFile">True</Option>
</OptionSet>
```

This creates a file named `OutputRanges_model name. csv`.

You can give a specific name to the file using the `OutputRangeFileName` option key:

```
<OptionSet name="Model AnalysisDirectives">
  <Option key="GenerateOutputRangeFile">True</Option>
  <Option key="OutputRangeFileName">MyFile.csv</Option>
</OptionSet>
```

If you do not specify an output range file directory, HiLiTE will create the file in the same directory as the model. You can specify it to be put in the same directory as the *Reports* directory containing other HiLiTE reports by using the `OutputRangeFileInReportsDir` option key:

```
<OptionSet name="Model AnalysisDirectives">
  <Option key="GenerateOutputRangeFile">True</Option>
  <Option key="OutputRangeFileInReportsDir">True</Option>
</OptionSet>
```

The `UpdateOutputRangeFileInPlace` option key allows HiLiTE to read an intermediate range file that contains the outputs of many models, but to update only the values of symbols produced by the model being processed:

```
<OptionSet name="Model Analysis Directives">  
  <Option key="UpdateOutputRangeFileInPlace">True</Option>  
  <Option key="OutputRangeFileName">MCP_IntermediateRanges.csv</Option>  
</OptionSet>
```

## 10 HiLiTE Template Manager

One of the critical inputs to HiLiTE is a set of block and test case definitions that tell HiLiTE how a block in MATLAB must be stimulated to be fully tested and how to determine expected results. The HiLiTE Template Manager provides the ability to develop and maintain this data for HiLiTE. You can use the HiLiTE Template Manager to add templates and block definitions or modify existing templates to meet your needs.

The HiLiTE Template Manager is installed with the main HiLiTE application. Installation places an icon on your Windows desktop.

### Starting the Template Manager

To launch the Template Manager, double-click the icon on your desktop or choose menu option

**Start → Programs → HiLiTE → HiLiTE TemplateManager.**



The Template Manager is displayed with a single main window that lists all the supported block types in a scrollable list on the left.

### Closing the Template Manager

When you are done working with the Template Manager, click the  in the upper right corner of the main window.

If you made changes but did not save them, the Template Manager will display a message asking if you wish to save changes before closing.

- Click **Yes** to make changes permanent.
- Click **No** to ignore changes and close
- Click **Cancel** to not close the Template Manager.

## 10.1 Viewing/Editing Information in the Template Manager

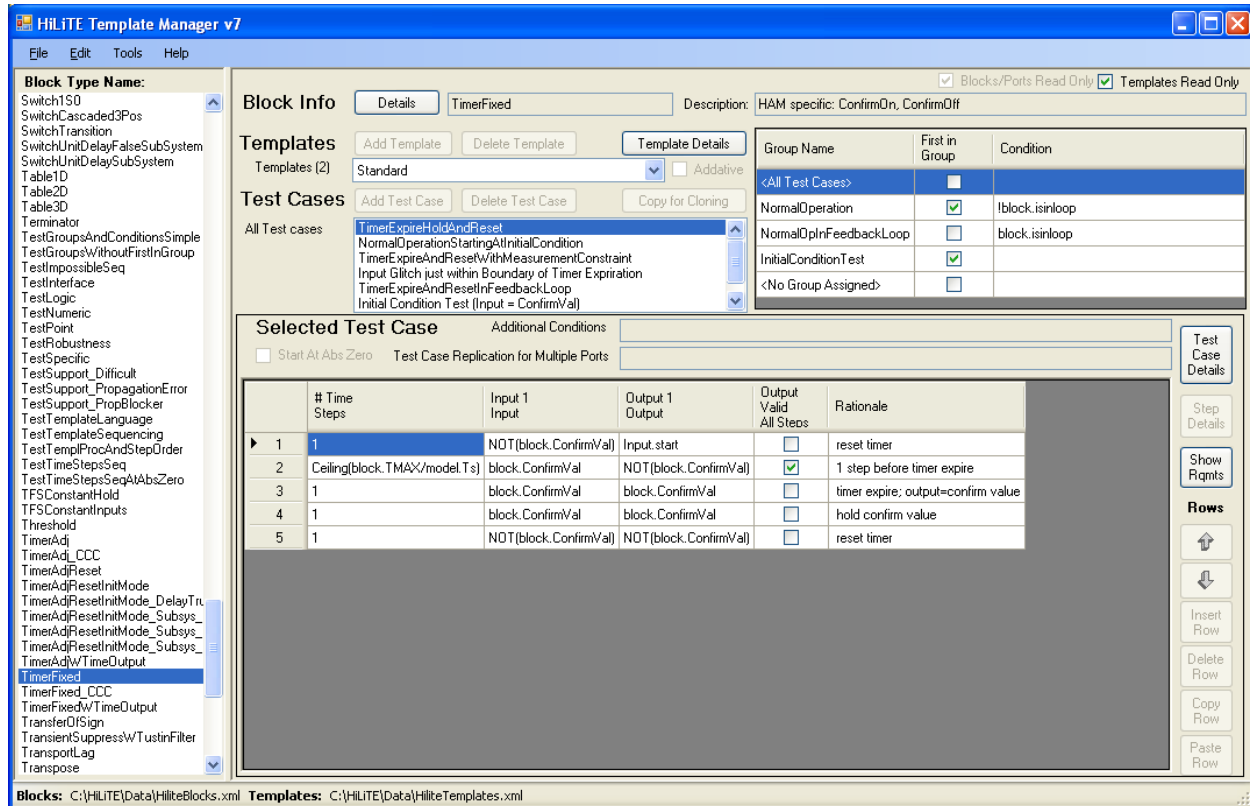
All users can view data in the Template Manager. Records are based on *Block Types* and show basic block type data along with the testing templates and procedures related to each block type.

### What is a Block Type?

Block Type refers to a particular “block” support provided by HiLiTE that represents certain functionality. A Block Type can support corresponding blocks from one or more block libraries – i.e. blocks from multiple block libraries can map to the same Block Type and more than one block from a single block library can map to a Block Type with different block attributes. To determine which HiLiTE Block Type name corresponds with a particular block, refer to the tables in Appendix E.

#### 10.1.1 Viewing a template for a Block Type:

Select a **Block Type Name** from the list on the left of the main Template Manager display. The Template Manager displays associated data on the right.



**Figure 83 - Template Manager Main Window**

You can display further details about the block, a template, or a test case. Details are often displayed in a separate dialog window. You must close the dialog before you can display another dialog. To close any dialog, click the X in the upper right corner or click the Cancel button.

- To display detailed information about the block, click the **Details** button next to **Block Info**. The Template Manager displays a separate BlockDetails window.

- To display more information about a template:
  - If the **Templates** label indicates that this block has templates in more than one family, select the template family from the drop-down list. The example in Figure 83 shows (2) next to the Templates label, indicating that this block has two templates. (For more information about template families, see Section 10.2.3 )

Click the **Template Details** button to display a ManageTemplate window. Close this window when you are done reviewing the information.
- To display test case details, select the test case in the **All Test Cases** box. The Template Manager displays steps in the Selected Test Case area at the bottom of the main window. Note that the list of test cases depends on the group that is selected in the
  - Click the **Test Case Details** button on the right to display a dialog with further test case details.
  - Select a step in the list and click the **Step Details** button to see more information about the selected step.
  - Click the **Show Rqmts** button to display any test case requirements in a Requirements Map to the left of the steps.

The following subsections describe the data available in each detail dialog or display area.

### 10.1.2 Viewing Block Type Details

In the main Template Manager window, click on **Details** button to see the details of the Block Type selected in the list on the left. The BlockDetails window pops up as shown in Figure 84. It shows the general Block Type data, detailed data for the ports, and the requirements for the Block Type. Port specifications are shown in a table at the bottom of the window. By clicking “**More >>**” button on the left side of the window, additional fields are displayed as shown in Figure 85.

*Note that much of the Block Type Details specification is HiLiTE internal configuration data and is not pertinent to HiLiTE user. The user need only pay attention to the information that is relevant to specification of block type requirements and test cases templates. The information is presented here only for completeness purposes.*

A brief description of all the Block Details fields is presented in Table 4. The description of the Port Type fields is given in Table 5. (Note: User will need to Click “**PortType Details**” button in the bottom-right corner to see all the fields.)

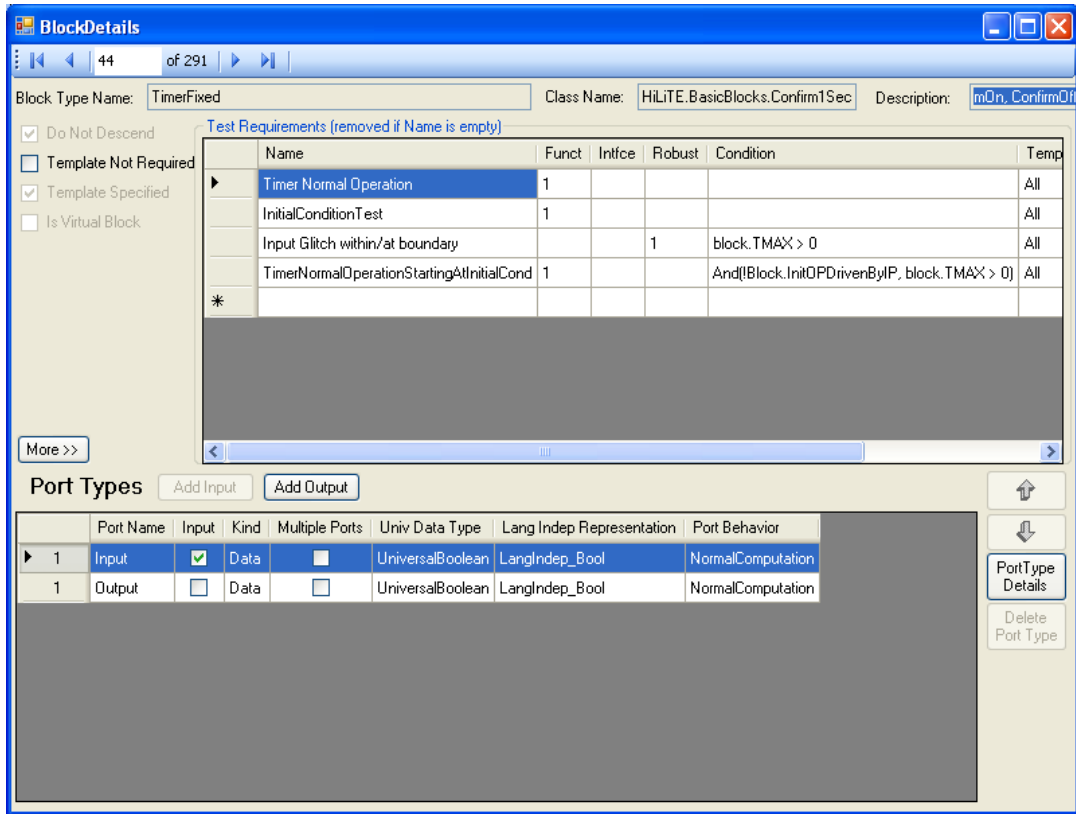
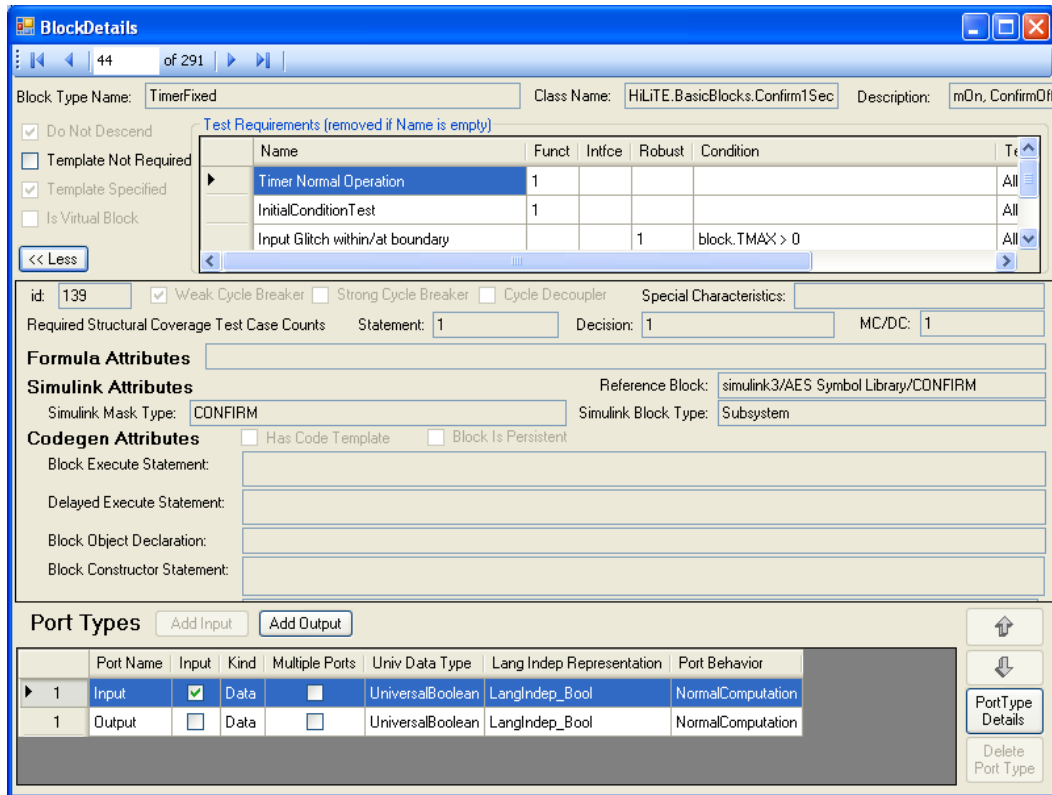


Figure 84 - Block Details window for Block Type TimerFixed

Figure 85 - Block Details Window with *More* Information Displayed

**Table 4. Fields in Block Details**

Field	Description	Format
Id	unique id (number) the block type	AutoNumber
Block Type Name	Text ID for block type	Text
<b>Check boxes:</b> Do Not Descend Template Specified Template Not Required Is Virtual Block Cycle Decoupler  <b>Reserved for Future Use:</b> Has Code Template Block is Persistent Weak Cycle Breaker Strong Cycle Breaker	If checked mean: - do not hierarchically descend into this block - this block type has a test template - a template is not required for this block - this is a virtual block (e.g. mux, demux) and may not require a test template - this block is a cycle decoupler (e.g. a unit delay) – one must be present in each cycle (feedback loop) in a model  - this block has a code generation template - Yes implies a persistent object must be allocated for this block; all outputs will use this; i.e. no separate variables for any output link or dest. model output	Yes/No
Description	Brief description of the block type.	Text
<b>Reserved for Future Use:</b> Required Structural Coverage Test Case Counts - Statement - Decision - MC/DC		
Class Name	Name of the class within HiLiTE source code used for this block type	Text
Special Characteristics	Comma-separated list of special block characteristics	Text
Reference Block	For block types that come from a library, the library block referred to	Text
Simulink Mask Type	The default Mask Type	Text
Simulink Block Type	The default Simulink “Block Type”	Text
Formula Attributes	Block and Port attribute keywords allowed in requirements or test case template formulae	Text
Block Execute Statement	Block Execute Statement	Memo
Delayed Execute Statement	This statement is applicable when the unit delay block gets finally executed in the end	Memo
Block Object Declaration	Denotes the corresponding class that implements this particular block...	Memo
Block Constructor Statement	Constructor for the block	Memo
Block Attribute Names	The standard attributes for this block type	Text
<b>Requirements</b> Name  Functional (Funct)	This table allows you to define (name) requirements for the block type and add a condition for it. Make sure the name is unique and reflects the functionality being tested.	Name is alpha-numeric



Field	Description	Format
Interface (Intfce) Robustness (Robust) Condition Template Family Untestability	<p>A requirement can be Functional, Interface, or Robustness.</p> <ol style="list-style-type: none"> <li>1. <i>Funct</i> captures the functional requirements of a block</li> <li>2. <i>Intfce</i> captures the Interface requirements of a block. Applies only to blocks with a separate function implementation.</li> <li>3. <i>Robust</i> captures the robustness requirements of a block</li> </ol> <p><i>Condition</i> describes an additional condition. HiLiTE will not consider the requirement in calculating requirement coverage and will not report the requirement Untested if the additional condition is not met for that requirement.</p> <p>You can indicate different requirements for different families. Use the <i>Template Family</i> column to select the appropriate family for a requirement in Requirement field. The default selection is All.</p>	<p><i>Funct</i> is an integer (number of times it's tested)</p> <p>Enter 1 for interface or robustness requirements or leave blank for no requirement.</p> <p>For <i>Condition</i>, enter a template manager language expression, as shown in Figure 87.</p>

Table 5. Fields of Port Type

Field	Description	Format
PortName	Name of the Port...used for port identification...	Text
Kind	the kind of information passed through the port - not the same as data type	Text
MultiplePorts	If true, this port can support zero - many lines instead of exactly 1	Yes/No
UniversalDataType	Defines what data types can be applied to a block's port instances in a model. (For more information about UniversalDataType, see subsection below)	Text
LangIndepDataType	The specific language independent data type, if one is required for this block type.	Text
PortBehavior	The behavior (typically) of an input port	Text
<b>Reserved for Future Use:</b>		
Overrideable	Indicates that code for this port must support overrides if true	Yes/No
Overridability	Indicates whether the port type is overridable, and whether it depends on the block context	Text
MustAllocateVariable	A variable must be allocated at this port, either persistent or temporary. Typically for blocks that are not persistent objects.	Yes/No
DoNotAllocateTempVariable	A temp variable must not be allocated at this port. The block must be a persistent object for this to be checked Yes.	Yes/No
BlockAssignsToVariable	This block's execute statement makes an assignment to any variable allocated to this	Yes/No

Field	Description	Format
	port, temporary or persistent	
OutputValueTemplate	template string that denotes the output value (usually an expression) for an output port.	
AllocatePersistentVariable	Whether to allocate a persistent variable...	
ShortCircuit	If true, this port can be a short circuiting port (if it is an input port) or can show the effects of a short circuit (if it is an output port)	Yes/No

### About UniversalDataType Specification

The UniversalDataType gives the type definitions without specific bits representation (magnitude, precision) on the target. The UniversalDataTypes are represented as a hierarchy of sets as shown in Figure 86. TA Type higher in the hierarchy is a superset of the lower types. For example, a UniversalNumber can be either a UniversalInteger or UniversalReal. Note that the actual data types applied to the ports of a specific block instance is specified in the MATLAB model. The UniversalDataType specification in Block Details is just a compatibility check with the type in the model. For example, a type definition of UniversalNumber in the Details window is compatible with both data types UniversalInteger and UniversalReal for a block instance. Earlier model compliance checks should have already established the correctness of the type in the model, so the compatibility check by HiLiTE is not mandatory.

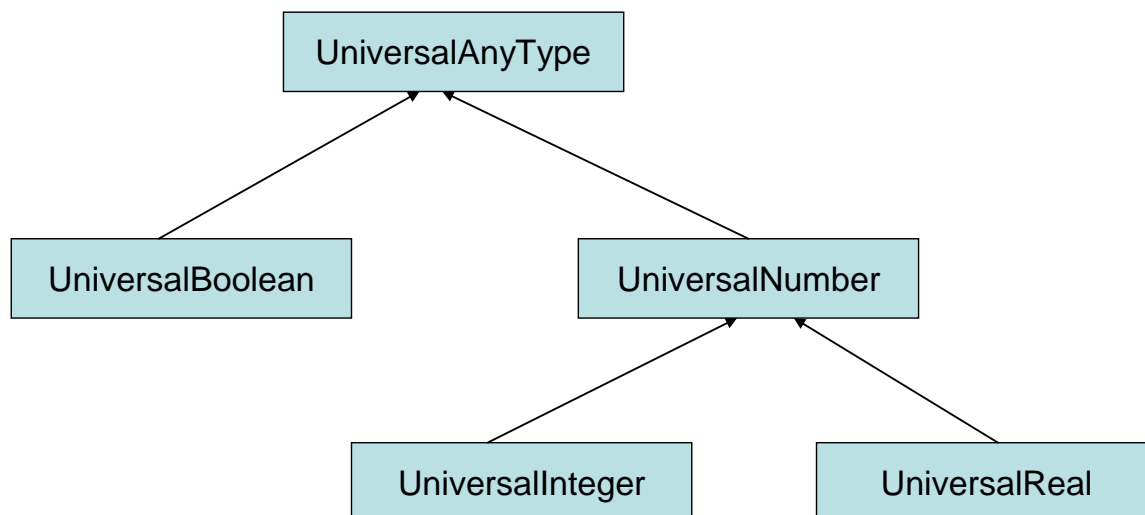


Figure 86. Universal DataType Sets Hierarchy

### 10.1.3 Viewing Block Requirements

The requirements for a block can be categorized as Functional, Interface, or Robustness requirements, as described in detail in Section 0.

HiLiTE provides a capability for users to view/modify the Requirements table for a block within the Block Details window, as shown in Figure 87. (To see the Requirements table: in Template Manager, select a Block Type on left and click on *Details* button.)

Test Requirements (removed if Name is empty)

	Name	Funct	Intfce	Robust	Condition	TemplateFamily	Untestability
►	Timer Normal Operation	1				All	▼
	InitialConditionTest	1				All	▼
	Input Glitch within/at boundary			1	block.TMAX > 0	All	▼
	TimerNormalOperationStartingAtInitialCond	1			And([Block.InitOPDrivenByIP, block.TMAX > 0])	All	▼
*						All	▼

Figure 87 - Requirements Table within Block Details Window

### 10.1.4 Viewing Template Details

When you click the **Manage Template** button, the Template Manager displays a dialog that gives basic template information (name, block type, rationale), shows test case/group relationships, and displays group conditions.

The dialog box 'Manage Template "Standard"' contains the following sections:

- Template Section:**
  - Family: Standard (dropdown)
  - ID: 88 (text box)
  - Block Type: 139 (text box)
  - Rationale: (text area)
  - ☐ Additive
- Test Cases and Groups Section:**
  - ☒ Show Test Cases for Selected Group
  - ☐ Show all Test Cases
- Table:**

	Group Name	First in Group	Condition
►	NormalOperation	<input checked="" type="checkbox"/>	!block.isinloop
	NormalOpInFeedbackLoop	<input type="checkbox"/>	block.isinloop
	InitialConditionTest	<input checked="" type="checkbox"/>	
*		<input type="checkbox"/>	
- Test cases for group "NormalOperation"** (List box):
  - TimerExpireHoldAndReset
  - TimerExpireAndResetWithMeasurementConstraint
- Test Cases with no Valid Group Association** (List box):
  - NormalOperationStartingAtInitialCondition
  - Input Glitch just within Boundary of Timer Expiration
- Buttons:** Add, Remove, Up arrow, Down arrow, OK, Cancel.

Figure 88 - Manage Template Dialog

Table 6. Fields in Manage Template

Field	Description	Format/Comments
ID	unique identifier for the template	AutoNumber
Block Type	numeric identifier for the block type	Number.
Name	Name of template	Text
Rationale	Description of the template procedure or of a specific step.	Use this field to tell why the procedure or step is important.
<b>Test Cases and Groups</b>		

Field	Description	Format/Comments
Show Test Cases for Selected Group	When this button is clicked, you will see a table of groups and conditions on the right as well as the test case lists below.	Option button
Show all Test Cases	When this button is clicked, the groups and conditions table is not visible. The test case lists show the names of all test cases.	Option button
<b>Groups and Conditions table</b>		
Group Name	Descriptive identification of a group. Test cases in a group can share common conditions.	Group names are alphanumeric strings (without punctuation) If you name a group that is not applied in the Group column for any procedure, HiLiTE will ignore it.
First in Group	Test cases in a group can be alternates. If First in Group is checked, HiLiTE will run test cases from the group until one test case passes, then ignore the remaining cases.	Check/uncheck
Condition	Boolean expression that describes the condition under which test cases in this group should be executed. If the condition is true, HiLiTE will perform the test case steps. Alternate test cases can cover the context for a specific instance of this block in a model.	Expression. See Template Language
<b>Test Case Lists</b>		
Test cases for group "name"/ All Test Cases	When the first option button is selected (default), this box lists all the test cases assigned to the selected group. When Show all Test Cases is selected, this box lists all test cases that are assigned to any group.	Text. List of test case names.
Test Cases with no Valid Group Association	This box lists all test cases that are not assigned to any group.	Text. List of test case names.

### 10.1.5 Viewing Test Case Details

Test cases are often assigned to a group that gathers related cases together. The table in the upper right of the main display shows whether a template has groups and the conditions that define each group.

- To see a list of all test cases for a template, click the <All test cases> row in the table in the upper right of the main display. The list is displayed in the Test Cases box.
- To see a list of test cases assigned to a specific group, click the row for that Group Name. The test cases are listed in the Test Cases box.

- To view details of a specific test case, select its name in the Test Cases box, then click the Test Case Details button. The steps that make up the test case procedure are displayed in the Selected Test Case Steps table at the bottom of the display.

The screenshot shows a dialog box titled "Test Case 'Single False'". It contains the following fields and controls:

- id:** 607
- template Id:** 186
- Test Case Name:** Single False
- ☒ Auto Adjust Input Values
- ☐ Dont Merge With Other Test Cases
- ☒ Check Untestable Condition
- Structural Coverage Count Satisfied:**
  - Statement:** 1
  - Decision:** 1
  - MC/DC:** 1
- Rationale:** walk a 0 thru each input while others are 1
- Buttons:** OK, Cancel

Figure 89 - Test Case Details Dialog

Table 7. Fields in Test Case Details

Field	Description	Format
Id	unique identifier (number) of this test case	Autonumber
template Id	unique identifier (number) of the template to which this test case belongs	Autonumber
Test Case Name	Descriptive name for this test case	Text
AutoAdjustInputValues	Not required for most test cases. If this option is checked, it indicates that HiLiTE can auto-adjust (change) an input value in the test case to be within the max allowable range constraints (if any) of that input on the specific block instance in the model. Acceptable usage for many primitive Math blocks but do not use for blocks with more complex functionality. <i>Check this flag only after a careful review of the test case.</i>	check/uncheck
Don't Merge With Other Test Cases	Checked indicates that this test case shouldn't be merged with any other test case	check/uncheck
Check Untestable Condition	Checked indicates that this template procedure can have an un-testable condition	check/uncheck
<b>Future Usage</b> Structural Coverage Count Satisfied Statement Decision MC/DC	Tells the count value with respect to how this procedure satisfies the block test requirement for each item. E.g., if Statement here=3 and the Statement coverage requirement=12, the requirement will be completely satisfied if 4 test cases with Statement=3 are run.	Number
Rationale	Description of the Procedure (Test Case) This is displayed in the report file. comment about rationale/assumptions/-	Text.

Field	Description	Format
	limitations of this procedure and its step values	

### 10.1.6 Viewing Test Case Steps

To see a list of all the steps in a test case procedure, select the appropriate template and test case. The Template Manager displays a row for each step in the table at the bottom of the display. View details for each step by selecting its row and clicking the **Step Details** button on the right.

Note that the number of columns in a step table varies, depending on the number of ports in the block type description.

The screenshot shows the 'Selected Test Case' window. At the top, there are fields for 'Additional Conditions' and checkboxes for 'Start At Abs Zero' and 'Test Case Replication for Multiple Ports'. Below this is a table with 7 columns: # Time Steps, Input 1 Input, Input 2 TimerInput, Output 1 Output, Output Valid All Steps, and Rationale. The table contains 5 rows of data. To the right of the table is a sidebar with buttons: 'Test Case Details', 'Step Details', 'Show Rqmts', 'Rows' (with up/down arrows), 'Insert Row', 'Delete Row', 'Copy Row', and 'Paste Row'.

	# Time Steps	Input 1 Input	Input 2 TimerInput	Output 1 Output	Output Valid All Steps	Rationale
▶ 1	1	0	MAX(self.op.min, model.Ts)	0	<input type="checkbox"/>	latch is set, timer initialized
2	Round(TimerInput.start/model.Ts)	1	MAX(self.op.min, model.Ts)	0	<input type="checkbox"/>	one cycle before timer expires
3	1	0	MAX(self.op.min, model.Ts)	0	<input type="checkbox"/>	latch is set, timer initialized
4	Round(TimerInput.start/model.Ts) + 1	1	MAX(self.op.min, model.Ts)	1	<input type="checkbox"/>	timer expires
5	1	0	MAX(self.op.min, model.Ts)	0	<input type="checkbox"/>	timer disabled, latch is set

**Figure 90 - Selected Test Case Details and Step Table**

The screenshot shows the 'Selected Test Case' window with a different table structure. It includes the same top controls as Figure 90. The table has 9 columns: # Time Steps, Input 1 Input1, Max: Input1, Input 2 Input2, Max: Input2, Output 1 Output, Output Valid All Steps, and Rationale. The table contains 1 row of data.

	# Time Steps	Input 1 Input1	Max: Input1	Input 2 Input2	Max: Input2	Output 1 Output	Output Valid All Steps	Rationale
▶ 1	1	self.op.max	self.allow.max	Input2.Allow.Min	Input1.StartMin - self.op.mdvd	block.getoutput	<input type="checkbox"/>	

**Figure 91 - Selected Test Case Details with min and max values for Inputs**

Double-click on the **Input** cell. A dialog window pops up where you can enter the minimum and maximum values for input port. You can enter a point value in the first text box and an entire range in the 2<sup>nd</sup> text box.

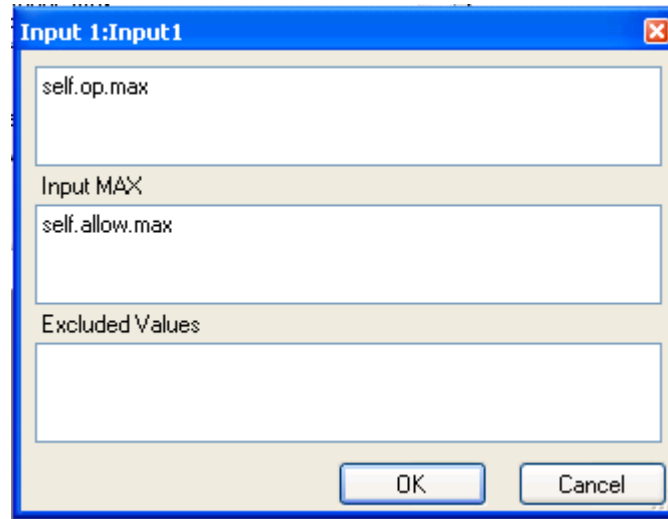


Figure 92 - Input dialog box

Table 8. Selected Test Case Fields and Step Table

Field	Description	Format
Additional Conditions	These conditions must be satisfied to execute the test case. A formula that can be evaluated to determine if this test case should be used. The result of this condition is ANDed with the result of evaluating conditions for the group this test case belongs to.	Boolean expression – see Template Formula Language.
Start At Abs Zero	When enabled, indicates that this test case needs to start at the “absolute zero” time step – i.e., the <code>modelname_initialize()</code> function must be called before the first step of this test case. Note that this does not imply initialization of the model inputs.	check/uncheck
Test Case Replication for Multiple Ports	A formula that is used to generate a class of test cases from one test case template (with new keyword such as <i>foreach</i> and <i>combination</i> )	Expression
# Time Steps	A time step is one step (frame) of execution of the model. For a model with a rate of 10 Hz, there are 10 time steps per second, each of .1 second duration.	value (number) or expression (see Template Language)
Input	Describes the input for the named port. The table will include a column for each input port defined for the block type. Inputs can be Input and MaxInput	value (number) or expression (see Template Language)
Output	Describes the output for the named port. The table will include a column for each output port defined for the block type.	value (number) or expression (see Template Language)

Field	Description	Format
Output Valid All Steps	If true, the value of output remains the same through all the steps specified in “# Time Steps” column. If false, the value is valid only for the last step of the “# Time Steps”; it is don't care for previous steps.	check/uncheck
Rationale	Text description of this test step	Text

The image shows a Windows-style dialog box titled "Step Details". It contains the following fields and controls:

- id:** A text box containing the value "162".
- Dependent Override:** A checkbox that is currently unchecked.
- Special Vector Type:** An empty text box.
- Short Circuit:** A checkbox that is currently unchecked.
- Rationale:** A large text area containing the text "latch is set, timer initialized".
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

Figure 93 - Step Details Dialog

Table 9. Fields in Step Details

Field	Description	Format
Id	Unique identifier (number ) for this test step	Autonumber
Rationale	Text description of this test step	Text
<b>For Future use:</b>		
Dependent Override	If true, this step overrides all dependent override outputs on the block under test	check/uncheck
Special Vector Type	Special mark on a vector that may affect how it is processed (e.g., colored in the test report)	
Short Circuit	If true, then this step can be affected by short circuiting	check/uncheck

### 10.1.7 Mapping Test Cases to Block Requirements

A test case can be mapped to one or more requirements of a block, implying that this test case tests all those requirements sufficiently. You can also map alternative test cases to the same requirement to



improve requirements coverage, since alternative test cases can address different conditions and contexts for the instances of a Block Type in various models.

In Figure 94, the selected test case is mapped to a single requirement: “ $a > b$ ”. Similarly, other test cases are mapped to corresponding requirements.

Test Cases		Add Test Case		Delete Test Case		Copy for Cloning	
All Test cases							
A > B, Using A Normal Range Max		A > B	<input checked="" type="checkbox"/>	AND(!Block.IsBoolComparator, In1.Allow.Max > In2.Allow.Min)			
A > B, Using B Normal Range Min		A < B	<input checked="" type="checkbox"/>	AND(!Block.IsBoolComparator, In1.Allow.Min < In2.Allow.Max)			
A > B, Using A & B Normal Range Max		A == B	<input checked="" type="checkbox"/>	!Block.IsFloatComparator			
A > B, Using A & B Normal Range Min		A ~ B Bool	<input checked="" type="checkbox"/>	Block.IsBoolComparator			
A < B, Using B Normal Range Max							
A < B, Using A Normal Range Min							

Selected Test Case		Additional Conditions		!Input2.Allow.IsPt	
<input type="checkbox"/> Start At Abs Zero		<input type="checkbox"/> Test Case Replication for Multiple Ports			

#	Time Steps	Input 1	Max	Input 2	Max	Output 1	Output Valid	Rationale
1	1	self.op.max	self.allow.max	Input2.Allow.Min	Input1.StartMin - self.op.mdvd	block.getoutput	<input type="checkbox"/>	

Requirement Map	
<input checked="" type="checkbox"/>	a > b
<input type="checkbox"/>	a < b
<input type="checkbox"/>	a == b
<input type="checkbox"/>	a ~ b

**Figure 94 - Mapping a test case to requirements**

1. Click the **Show Rqmts** button to see how requirements are mapped.
2. If you want to change the mappings, check or uncheck the box next to each map you want to change.

**Many-to-many mapping from test cases to requirements.** A test case can be mapped to more than one requirement which implies that the test case satisfies all those requirements. When the test case is generated, one count of each of the mapped requirements is satisfied and counted in the computation of requirements coverage. Conversely, a requirement can be mapped to from more than one test case, which implies that if any one of those test cases is generated, the requirement is satisfied.

## 10.2 Adding and Changing Block Requirements and Templates

If you are authorized to change data in the Template Manager, use the *Read Only* fields in the upper right corner of the Template Manager main window.

☒ Blocks/Ports Read Only    ☒ Templates Read Only

**Figure 95 - Checkboxes to make Template Manager read only**

**Blocks/Ports Read Only:** Un-checking this field allows changes to all Block Type details, including addition and detection of blocks. Only HiLiTE developers are authorized to do this.

**Templates Read Only:** Un-checking this field allows changes Block Type Requirements and Templates.

### 10.2.1 Adding/Changing Block Requirements

The test requirements for a block can be categorized as Functional, Interface, or Robustness requirements, as described in detail in Section 0

HiLiTE provides a capability for users to view/modify the Requirements table for a block within the Block Details window, as described in Section 10.1.2.

Authorized users can add new requirements or edit the existing requirements. *Note that the users are not allowed to edit any information in Block Details other than the Requirements.*

Test Requirements (removed if Name is empty)							
	Name	Funct	Intfce	Robust	Condition	TemplateFamily	Untestability
▶	MinOut	Input.Count				CCC	▼
	MaxOut	Input.Count				CCC	▼
	AllMax	1				CCC	▼
	AllMin	1				CCC	▼
	SumOperation	2				Standard	▼
*							▼

Figure 96 - Editing block test requirements

Each row in Figure 96 represents a requirement of the Block Type.

#### Requirement Columns.

- Name: the name of the requirement
- Funct | Intfce | Robust: Only one of these columns can have a value for a particular requirement. The value is a number that denotes that the named requirement is of the type (Functional, Interface, or Robustness) and has a count denoted by the value. The count is usually “1” for most requirements except for certain requirements of blocks that have a variable number of ports (e.g. an AND block) and there is a requirement for every port. In these cases, an expression of the form “PortName.Count” can be used in a column, where Count is a supported keyword.
- Cond: This is short for Condition. It is a Boolean expression that can use any operators or keywords of the Template Formula Language (*see Section 10.6*). If a Condition is present on a requirement, this requirement exists only for those block instances (of this Block Type) for which the Condition evaluates to true. This is useful when a single Block Type denotes blocks with different types, as shown in the example of Figure 96. For many block’s requirements, this column is left blank; i.e. no condition exists for this requirement.

**Modifying a requirement:** Any column can be edited as described above.

**Adding a requirement:** Use the scrollbar on the right of the Requirements table to show the bottom-most row with a “\*” on left. Start typing in the Name column and a new row will be created. Add the data in other columns appropriately. Then, select any other row in requirement window and close the requirement window.

**Deleting a requirement:** Select the text in the Name column and delete it. The requirement will be deleted when the Block Details window is closed.

**Exiting Requirements Table:** Close the Block Details window. Your changes to the requirement will be retained within the Template Manager. You must save these changes to an appropriate when exiting the Template Manager.

**Important Note:** When adding a requirement, select a requirement row other than one you have added, before closing the Block Detail window. Otherwise, the newly added requirement will not be remembered by the Template Manager.

### 10.2.2 Adding/Changing Templates

Authorized users can create new templates or change or delete existing templates. They can change existing templates by adding, deleting or modifying Template Details, Groups, Test Cases, or Test Case Steps.

A complete template consists of basic template information such as name and groups, the test case or cases to be run, and the steps for each test case. Most blocks will be associated with only one test template.

**To create a new template for a block:**

1. Make sure the box next to **Templates Read Only** is unchecked.
2. From the **Block Type Name** list, select the block type to which you wish to add a template. The Template Manager displays data on the right.
3. Click the **Add Template** button. The Template Manager displays a confirmation message.
4. Click **yes**. The Template Manager displays the TemplateDetails dialog.

**Figure 97 - Template Details Dialog**

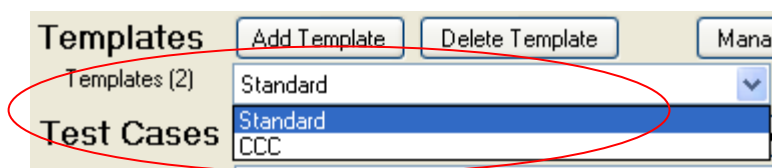
5. Enter a template **Name** and **Rationale**.
6. From the **Family** drop-down select the template family you are adding to.
7. Click **OK** to save the information and close the dialog.
8. Create a test case.
  - a. Click the **Add Test Case button**. Template Manager displays a confirmation message; click Yes to display the Test Case Details dialog.
  - b. Complete all fields for which you have information. For information about completing fields, refer to Table 7.
  - c. Close the dialog when you are done.
9. Add one or more steps to the test case.
  - a. Click the **Insert Row** button on the right side of the Selected Test Case steps table. Template Manager adds a row to the table. Complete fields in the row. (For descriptions of the fields, refer to Table 8.)
  - b. Create and complete one row for each procedure step.
10. Choose menu option **File → Save** to save your work.
11. Repeat Steps 5 through 10 as needed.
12. If the test cases should be grouped according to defined conditions, create the groups and assign test cases to them using instruction in Section 10.2.4, Creating Groups and Conditions.

### 10.2.3 Supporting Alternate Template Families

HiLiTE lets you maintain multiple “Families” of templates on a block-by-block basis. Different requirements can be given to test cases in different families. Use a template family to select the appropriate family for a requirement in Requirement field. The following Template Families are allowed:  
::

- **Standard Template Family:** This represents the test case templates that are common to various programs. The name of the Standard template (in Template Manager) is typically the name of the block or just “Standard.”
- **Program-Specific Template Families** If, for a particular block, a certain program finds the Standard test case template inadequate, then an alternative template can be specified for that block. This alternative template must have one of the following names in the Template Manager GUI (the name of the template identifies the family): CCC, 787FC, 787FC\_Rlx, 787FC\_SF, A350, UserDef\_A, UserDef\_B, UserDef\_C.

Example:



**Figure 98 - Templates drop-down**

Figure 98 shows that the block has two templates: Standard and CCC. It means that HiLiTE supports two sets of templates for this block.

#### Running HiLiTE with a particular template family:

By default, HiLiTE uses the Standard templates when generating tests for a model. If you want to use a different template family, specify the name of the family with this option in the Command file:

```
<OptionSet name="TestGenDirectives">
  <Option key="UseAlternateTemplateIfPresent">CCC</Option>
</OptionSet>
```

If a template family is specified, HiLiTE uses the template of the specified family for a particular block. If no family-specific template is provided for a block, then HiLiTE uses the Standard template for that block. (Note: the Standard template can be identified by its name being a string other than the set of family names listed above. The HiLiTE Status Report will indicate which family was used.)

#### Capturing block requirements specific to a template family:

HiLiTE supports mapping requirements for the test cases to different template families. Select the template family in the TemplateFamily drop down for the test case. The default is “All,” meaning the test case will apply to every template family.

Example:

Test Requirements (removed if Name is empty)							
	Name	Funct	Intfce	Robust	Condition	TemplateFamily	Untestability
▶	MinInput	1				All	▼
	MaxInput	1				All	▼
	Out of range input			1		CCC	▼
*							▼

**Figure 99 - Selecting TemplateFamily test case requirement**

In the above example:

- Requirement “MinInput” is applicable to all template families.
- Requirement “Out of range input” is applicable only to the CCC family.

### 10.2.4 Creating Groups and Conditions

Creating groups of similar test cases will test similar functionality of a block. These test cases can be in "FirstInGroup or not" based on your requirements. For examples of groupings, examine the details of the ArcTan2 and Product block type templates.

#### To create a group:

1. Display the template to which the group will belong and click the **Manage Template** button.
2. Make sure the option **Show Test Cases for Selected Group** is selected and a table with columns for Name, First in Group and Condition appears on the right.
3. In an empty row (indicated with an \*), type a group name in the first column.
4. If a test will be complete with the first successful test case, check the **First in Group** indicator and enter an expression in the **Condition** column.

See Section 10.6, Template Formula Language for information about writing condition expressions.

5. Assign test cases to the group:
  - a. Make sure the row for the group is selected.
  - b. In the list **Test Cases with no Valid Group Association**, highlight a test case that you want to add to the group.
  - c. Click the left-point **Add** arrow. Template Manager moves the test case into the list Test cases for group “*groupname*”
  - d. If you add a test case by mistake, select it in the left hand list and click the Remove arrow to move it back to Test Cases with no Valid Group Association.
  - e. Repeat Steps b and c for each test case that should belong to the group.
6. When you are done, click **OK** to make the changes permanent and close the dialog.

### 10.2.5 Modifying Templates

You can change any aspect of an existing template simply by selecting the template, opening, the appropriate dialog or selecting an appropriate field, and typing new information. For templates that have been in use for awhile, the most likely changes will include:

- Adding test cases
- Adding, rearranging, or changing steps in existing test cases
- Organizing the test cases into different groups

#### When you are ready to make changes to a template:

1. Click the box next to **Templates Read Only** to remove the check mark.
2. Select the block type for the template you wish to change.
3. Select the item you wish to change and follow the instructions given in the appropriate procedure below.

- When you have made all changes, choose menu option **File → Save** to make all changes permanent. If you do not save changes, Template Manager will prompt you to do so when you select a different block type or when you close the application.

### 10.2.6 Adding Test Cases

Each template can include any number of test cases.

**To add a new test case:**

- Display the template to which it will belong.
- Click the **Add Test Case** button. The Template Manager displays a confirmation message.
- Click **Yes**. The Template Manager displays the New Test Case dialog.

**Figure 100 - New Test Case Window**

- Enter a **Test Case Name** and **Rationale**. Complete all other fields for which you have data.
- Click **OK** to close the dialog. The data will be saved the next time you save the Template Manager file.

### 10.2.7 Deleting Test Cases

You can remove test cases from a template.

**To delete a test case:**

- Display the template to which the test case belongs.
- From the Test Case list, select the test case you wish to delete. You may wish to display its details (click the Test Case Details button) to verify that you are deleting the correct test case.
- Click the **Delete Test Case** button. The Template Manager displays a confirmation message.
- Click **Yes**. The Template Manager removes the test case from the list. This change will become permanent the next time you save the Template Manager file.

### 10.2.8 Changing Test Cases

You can change the details (basic description) of a test case. You can also add, change, delete and reorder the steps for a test case procedure.

**To change details of an existing test case:**

1. Select its name from the list for the appropriate template. If the test case you want to change is not listed, make sure that you have selected either All Groups or the correct name of the group to which it belongs.
2. Click the **Test Case Details** button to display detail fields.
3. Type over or delete information in the fields you want to change.
4. Close the dialog.

To make changes to the steps for the test case, follow the instructions for one of the following procedures.

### 10.2.9 Adding Steps to Test Cases

You can add new steps either by inserting a blank row and completing the row or by copying and pasting an existing row, then changing the data as needed.

**To copy and paste a new row:**

1. Select the test case name from the list for the appropriate template. If the test case you want to change is not listed, make sure that you have selected either All Groups *or* the correct name of the group to which it belongs.
2. In the **Selected Test Case** area of the display, select the row you want to copy.
3. Click the **Copy Row** button. (no reaction from interface)
4. Click the **Paste Row** button. Template Manager pastes the row above the copied row.
5. Make changes to the row as needed. Table 8 describes the fields.
6. Click the **Step Details** button and make changes to the Step Details dialog as needed. Table 9 describes the fields.
7. Choose menu option **File → Save** to permanently save your changes,.

**To add new steps to a test case:**

1. Select the test case name from the list for the appropriate template. If the test case you want to change is not listed, make sure that you have selected either All Groups or the correct name of the group to which it belongs.
2. In the Selected Test Case area of the display, select the row after the step you want to add.
3. Click the **Insert Row** button, Template Manager displays a new row above the current row. The new row has a colored background (you can select this color in your preferences).
4. Complete information in the row. Table 8 describes the appropriate information for each column.
5. Save the row (choose **File>Save**. Note that the background color may not return to white even though the changes are saved).
6. Select the new row and click the **Step Details** button to display the Step Details dialog. Complete the Step Details data. Table 9 describes the appropriate information for each field.
7. Click **OK** to close the dialog.
8. To permanently save your changes, choose menu option **File>Save**.

### 10.2.10 Deleting Steps in a Test Case

To delete steps from a test case:

1. Select the test case name from the list for the appropriate template. If the test case you want to change is not listed, make sure that you have selected either All Groups or the correct name of the group to which it belongs.
2. In the Selected Test Case area of the display, select the row you want to remove.
3. Click the **Delete Row** button. Template Manager removes the row from the table.
4. To permanently save your changes, choose menu option **File>Save**.

### 10.2.11 Reordering Steps in a Test Case

You can change the order in which test case steps occur.

To change step order:

1. Select the test case name from the list for the appropriate template. If the test case you want to change is not listed, make sure that you have selected either All Groups or the correct name of the group to which it belongs.
2. In the Selected Test Case area of the display, select the row you want to move.
3. Click the button for the direction you want to move the row (↑ or ↓). . Template Manager moves the row in that direction.
4. Repeat Step 3 until the row is in the proper position.
5. Follow Steps 3 and 4 for each row you want to move.
6. To permanently save your changes, choose menu option **File →Save**.

### 10.2.12 Changing Group Relations (assigning test cases to groups)

HiLiTE runs test cases in defined groups. The groups and the conditions that define them are created as part of the initial template creation using the Manage Template dialog (Figure 88). You can change the group to which a test case belongs by disassociating it from its current group and associating it with a different group. If a test case is not associated with any group, you can give it an association.

To change test case/group relations:

1. Select and display the appropriate template.
2. Click the **Manage Templates** button to display the Manage Templates dialog.
3. Select the group to which the test case you want to change currently belongs.
4. Find and select the test case name in the **Test cases for group “name”** list box on the left.
5. Click the **Remove**/right-pointing arrow button. Template Manager moves the test case name to the list for Test Cases with no Valid Group Association.
6. Select the group to which you want to assign the test case.
7. Select the test case name from the **Test Cases with no Valid Group Association** list.
8. Click the **Add**/left-pointing arrow button. Template Manager moves the test case to the Test cases for group “name” list box on the left.
9. Click **OK** to close the dialog.
10. Save your changes.



## 10.3 Tips for Creating Test Cases

- Design test cases such that each test case is independent. The output of a test case should not depend on the previous test case.
- Use the keyword “<portname>.previous” only if the value set at that port in the immediately previous step is *not* don’t care (X). .
- Use block.getoutput() only if it is supported and inputs are not Don'tcare (X).
- If you select "Start At Abs Zero" or "Auto Adjust Input values," ensure that these selections are exactly the intention of the test case design. (By default, these check boxes are unchecked.)
- Allowed values for Boolean data type in templates are: 0, 1, X, self.op.min, self.op.max, self.Allow.min, or self.Allow.max. Arithmetic expressions such as “self.op.min + self.op.mdvd” and “self.op.max – self.op.mdvd” are not allowed. mdvd should never be used on Boolean data types. (For information about the template formula language, see Section 10.6)
- In a group that includes a FirstInGroup condition for its test cases, an additional condition may not be required if you are certain that at least one test case from the group will be generated.

## 10.4 Application Menus

The following subsections describe the menu options available in the Template Manager interface.

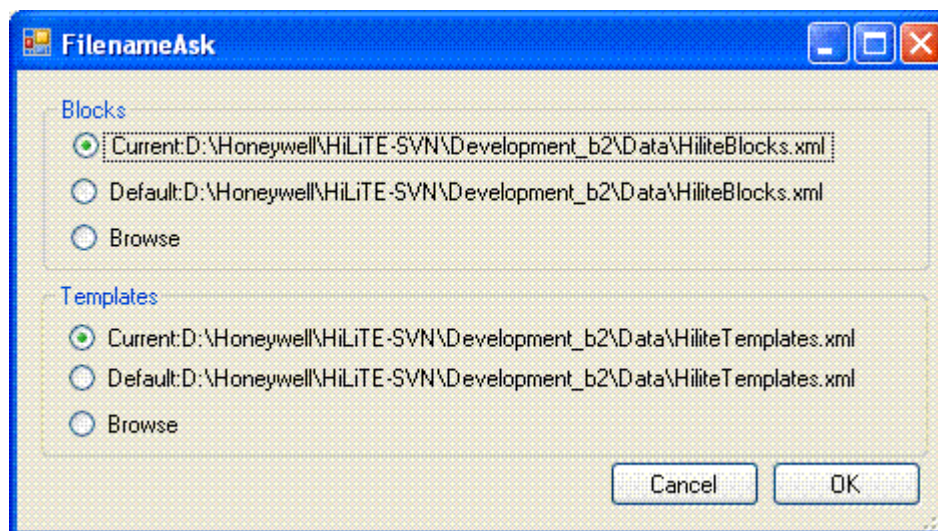
### 10.4.1 File Menu

The File menu includes four options.

<b>Open</b>	Dialog box lets you redisplay the default blocks and template data (XML) files or browse for a different copy/version.
<b>Save</b>	Makes any changes you've made permanent in the open version of the data files.
<b>Save as</b>	Dialog box lets you save the changes to blocks and templates data to XML files with a different name than the open one.
<b>Export</b>	Dialog box lets you export template or test case of any block
<b>Import</b>	Dialog box lets you import template or test case of any block
<b>Exit</b>	Closes the Template Manager. If you have not saved changes, it will display a message asking if you wish to do so.

### Open and Save As Dialog Box

The Open and Save As options allow a user to manage their changes to blocks and templates data and save them in specifically named files that can be later loaded by HiLiTE instead of the standard blocks and templates xml files supplied with a HiLiTE release. Both these options use the same dialog box that the user can use to open and save specific files, as shown in Figure 101.

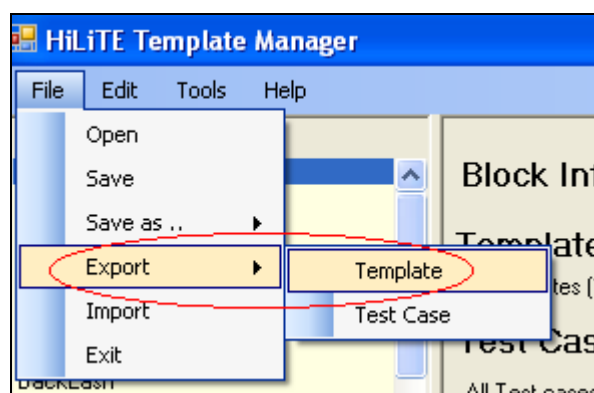


**Figure 101 - Dialog Box for Open and Save As**

Note that there are two parts to the blocks and templates data – contained in separate xml files. The **Blocks** part of the data contains the Block Type specifications and details, including block requirements. The **Templates** part of the data contains all template specifications, including mapping of test cases to block requirements. In this dialog, the user can choose the Current or Default option or Browse to a specific filename.

### Export and Import Dialog Box

Template manager has option to export/Import a test case/template. This option is useful while merging the changes done by multiple people into a single database file.



**Figure 102 - Dialog Box for Export and Import**

Template can be exported by selecting any block and select option File -> Export -> Template. Similarly, test case can be exported by selecting any test case of any block and select option File -> Export -> Test case. Save the exported file in any location. The exported files will be in xml format.

Exported template or test case can be imported by selecting option File -> Import. Browse to the path where exported xml files are present and select the xml files. Template or test case will be imported to the same block type.

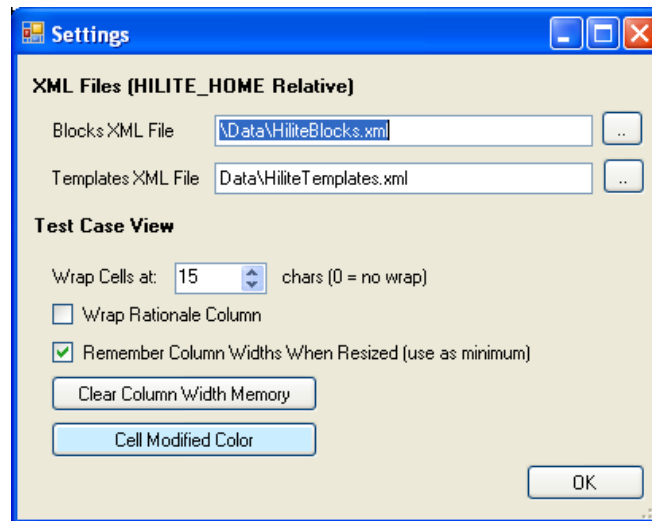
### 10.4.2 Edit Menu

The Edit menu includes two options: Settings and Revisions.

#### Settings

Use the **Edit → Settings** option to display a dialog where you can set some default behaviors of the Template Manager to suit your needs.

To use the dialog, make new selections in any fields, then click OK. To make the changes permanent, choose the menu option **File → Save**.



**Figure 103 - Settings dialog box**

Fields in the XML Files section of this dialog show where the data files that the Template Manager reads currently reside. If your default XML files are in a different location, click the browse [...] button to display an Open dialog and browse to that location.

Fields under Test Case View let you customize some settings in the Selected Test Case area of the display.

- To have cells in the step table wrap differently, use the up and down arrows at the end of the **Wrap Cells** field to change the number of characters to show on each line. If you select 0, the cell will not wrap.
- Select/deselect **Wrap Rationale Column** to change the behavior of just that column.
- Select **Remember Column Widths...** for the Template Manager to use the column widths you set in the table as minimum widths. (To set the widths in the table, let the cursor hover over a column border. When it changes to a two-headed arrow, drag left or right.)
- Click **Clear Column Width Memory** to set all columns back to the size you entered in Wrap Cells.
- **Cell Modified Color** shows the color that Template Manager uses to indicate when you've changed the contents of a table cell (or inserted a row). Click this button to display a color chart from which you can select a different color.

#### Revisions

Use the **Edit → Revisions** option to see which version of the Template Manager and Family Templates you are using, as shown in the left part of Figure 104. The **Base Revision** for a template family is the revision for that family as released in the HiLiTE release. If you (or another user with access to your

copy) have made changes to specific templates, increment the **User Revision** number appropriately. Note that you can only increment the revision number when Templates Read Only is not checked. If you are in read-only mode, you will not see the options for resetting or incrementing the revision numbers (see example on right in Figure 104)

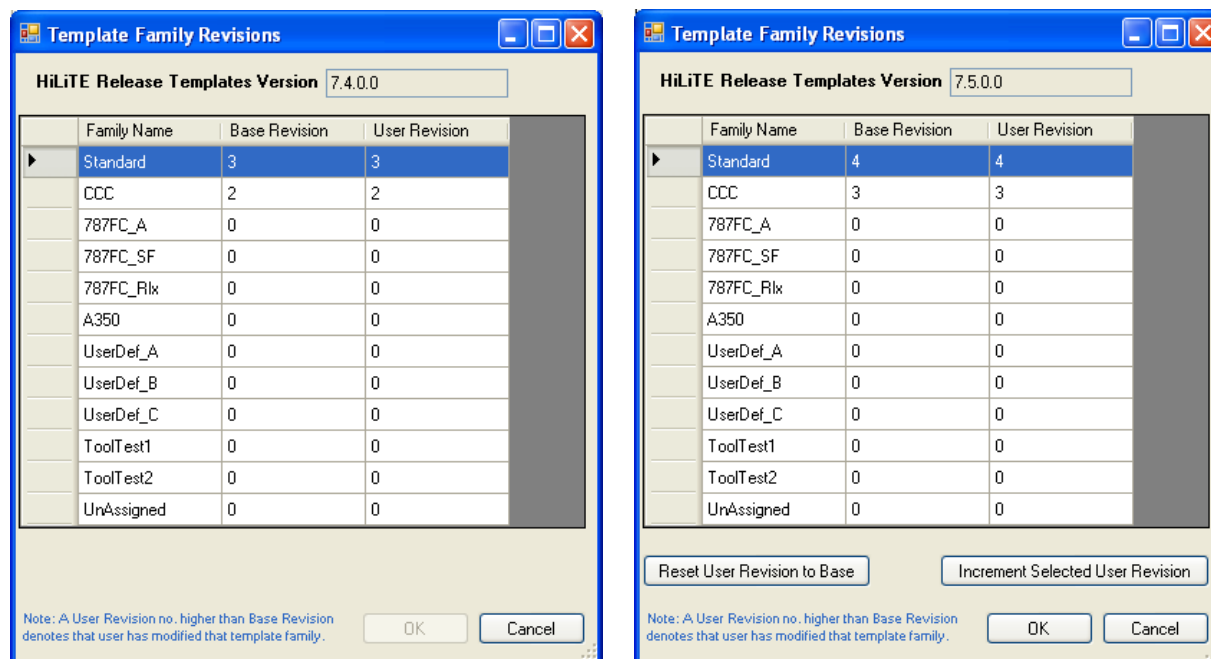


Figure 104 - Template Family Revisions dialog

### 10.4.3 Tools Menu

The tools menu has two options

**Block Types W/O templates**

**Check Formulas**

### 10.4.4 Help Menu

The Help menu includes two options

**Contents** Displays the help file and its table of contents

**About** Version information about your copy of the Template Manager.

## 10.5 Managing Configuration of Changes to Blocks and Templates

As noted earlier, one type of HiLiTE input data that users can control is the blocks and templates specification given as XML files. Using the methods described above, the user can modify and save this data in new files. This section describes the procedures for reviewing the modified data and using the modified data when executing HiLiTE.

**Note:** The process of using modified blocks and templates data, as described here, is supported by a qualified version of HiLiTE. No other changes to the tool configuration data by the user are permitted.

### 10.5.1 Using Modified Blocks and Templates Data when running HiLiTE

Using the capabilities described in Sections *10.2 Adding and Changing Block Requirements and Templates* and *10.3 Tips for Creating Test Cases*, you can modify the block and template data and save the changes in new files.

By default, HiLiTE will use the standard version of the blocks and templates data supplied with HiLiTE release. In order to provide HiLiTE a user's version of the blocks and templates data, the `<BlockDBFile>` and `<TemplateDBFile>` elements in the HiLiTE command file must be used.

Figure 105 shows an example of specification of user-modified versions of blocks and templates data in a command file

```
<?xml version="1.0"?>
<HiLiTEParameters>
  <HiLiTEMgmt pauseOnCompletion="no" />
  <ProjectPath>./ProjectPath>
  ...
  <Extension>HAM</Extension>
  <BlockDBFile>HiliteBlocks_mine.xml</BlockDBFile>
  <TemplateDBFile>HiliteTemplates_mine.xml</TemplateDBFile>
  ...
  <Command name='TestGen'>
    ...
  </Command>
</HiLiTEParameters>
```

**Figure 105 - User specification of blocks and templates data in HiLiTE Command File**

The value of the XML element `<BlockDBFile>` or `<TemplateDBFile>` is the name of the corresponding XML file. The following kinds of filenames are supported:

- Filename relative to the ProjectPath directory, e.g. as shown in Figure 105.
- Filename that specifies a full absolute path; e.g.:  
D:\MyProject\HiliteBlocks\_mine.xml
- Filename that starts with an environment variable. E.g., the following specifies a file in the *HiLiTE\_HOME* directory structure:  
%HiLiTE\_HOME%\Data\HiliteBlocks\_mine.xml

HiLiTE lists the user-specified files in the *modelname\_StatusReport.htm* file, as shown in Figure 106.

```
Test Generation Summary

HiLiTE Released Templates Version: 7.6.0.0; Template Family Revision: Standard=9
Template Family Used: Standard

User Specified Blocks and Templates files were used
  Blocks: d:\TestGeneration\Examples\HiliteBlocks_mine.xml
  Templates: d:\TestGeneration\Examples\HiliteTemplates_mine.xml
```

**Figure 106 - Model Status Report showing use of User-Specified Blocks and Templates Files**

### 10.5.2 Reviewing User-Modified Block and Template Data

HiLiTE uses the blocks and templates data described above as an input when generating test cases for a model. Care must be taken to ensure that undesired/incorrect changes are not made; this is especially

important when using a qualified version of HiLiTE. The following procedure is recommended to be followed by users:

1. Only make changes to requirements and test case templates of specific blocks, using methods as described in previous sections.
2. Review the changes within the Template Manager before saving them and make a list of all changes. It is also advisable to diff the changed blocks and templates XML files with the HiLiTE Release standard versions using a diff program such as BeyondCompare to ensure no unintended changes were made.
3. Testing modifications: Since it is possible to make mistakes in modifications/addition of test cases that may not be detected in reviews, the users should test the modifications:
  - For all the blocks with requirements/template modifications, execute HiLiTE on the test models supplied with HiLiTE release for those blocks, using the modified version of blocks and templates data. (scripts are provided to execute HiLiTE on these models and run the HiLiTE Test Harness on the code for the models.)
  - Verify that HiLiTE does not report any errors in the interpretation of the modified requirements and templates.
  - Verify that no test vector errors are reported by HiLiTE Test Harness when applying the generated test vectors on the models' code.
  - As necessary, compare with the standard HiLiTE output for these models, supplied with the release.
4. Perform configuration management of modified blocks and templates data, the list of modifications, and test results, required for configuration management (CM) procedures of the user's project.

## 10.6 Template Formula Language

Formulae in the templates are arithmetic expressions in C syntax that evaluate to *double* (64-bit floating point) numbers. The value of an individual term or sub expression is also maintained as a *double*.

- Floating point values are represented as doubles (64-bit floating point)
- Integer values are represented as whole numbers; e.g. 1, 2, 3, -1, -5, etc.  
If the desired value is an integer, make sure that the formula will evaluate exactly to an integer.
- Boolean values are represented by True or False. *True* is represented by 1, *False* by 0.

Use Operators, Predefined Constants, Math and Logic Functions, and Keywords to create formulae for group conditions, time steps, port values, and multiple port elaboration rule.

### 10.6.1 Infix Operators

The following infix operators (shown in the order of precedence) can be used:

Operator Type							Remarks
arithmetic	+		–				
arithmetic	/		*				
relational	>	<	==	>=	<=	<>	return 1 for True and 0 for False
arithmetic	^						$x^y$

Parentheses '(' and ')' can be used to any depth of nesting to override the default precedence or to disambiguate expressions.

Examples:

```
2 * .01 * (5 + 3)
in1.op.max - in2.op.min
in1.op.max > 5
```

### 10.6.2 Predefined Constants

The following constants can be referenced directly using their names:

`pi`, `e`, `true`, `false`

(*true* and *false* evaluate to 1 and 0 respectively)

Examples:

```
2 * pi
e ^ 2
```

### 10.6.3 Math and Logic Functions

The following math and logic functions are provided. *Note that function names are case insensitive.*

Note that expressions and function calls can be nested; thus an argument to a function can be an arbitrary expression involving operators or other functions.

**Table 10. Math and Logic Functions**

Function <i>Function names are case insensitive</i>	Arguments, Return Value	Usage Examples / limitations
sin(x), cos(x), tan(x), sinh(x), cosh(x), tanh(x)	x : angle in radians Returns: the trigonometric function value	sin(in1.op.min)
asin(x), acos(x), atan(x)	Returns: radians	asin(.75)
deg(x), degrees(x)	x: angle in radians Returns: angle in degrees	deg(Block.Heading)
rad(x), radians(x)	x: angle in degrees Returns: angle in radians	
compare(x,y)	x,y : two values (numeric or string) Returns: -1 if x < y ; 0 if x == y ; +1 if x > y	
log(x), logn(base, x)	x, base : numeric expression Returns: value of logarithmic function	
sqrt(x), abs(x), exp(x) fact(x)	x: numeric expression Returns: value of mathematical function	sqrt(Input.Start) abs(Block.LoLimit)
round(x), floor(x), ceiling(x)	x: numeric expression (possibly floating point) Returns: integer result of the mathematical operation	<b>Limitation</b> due to inherent floating point representation error: (ip = integer part) If x is very close to ip.5 <i>round</i> can yield ip or ip+1 If x is very close to ip.0 <i>floor</i> can yield ip or ip -1 <i>ceiling</i> can yield ip or ip +1
max(x1, x2, x3, ...)	x1, x2, ...: numeric expressions	max(in1.op.max, in2.op.max,

Function <i>Function names are case insensitive</i>	Arguments, Return Value	Usage Examples / limitations
min(x1, x2, x3, ...)		5.2)
and(x1, x2, x3, ...) or(x1, x2, x3, ...) Note: This format for Boolean expressions works as though the operator in front of the parentheses is inserted between values (e.g. x1 and x2 and x3)	x1, x2, ...: Boolean expressions (two or more arguments) Returns: logical operation Boolean result	and(Block.IsInt, Input.Op.Min > 2)
not(x)	x : a Boolean expression Returns: Boolean negation	not(Block.IsBoolComparator)
Bitwise operators BitNot(x) BitAnd(x, y)	These operators perform bitwise operations on their operands: x, y: a number, preferably in hex format, or an expression	BitNot(Block.BitMaskHex)  BitAnd(Block.BitMaskHex, 2 ^ 10)
hex(x)	s: an int value, returns the value represented in hex	
if (cond, true_exp, false_exp)	if <i>cond</i> is true then return the <i>true_exp</i> , else return <i>false_exp</i>	If (in1.op.max > 5, in1.op.max / 2, 2)
combinations(port_name, num_at_time)	Steps through each port named port_name, picking num_at_time ports at a time	combinations(input, 2) Used with a port where MultiplePorts is enabled.
foreach(port_name)	Steps through each port named port_name	foreach(input) Note that foreach is equivalent to combinations(input, 1) Used with a port where MultiplePorts is enabled. Used in conjunction with the current keyword.

### 10.6.4 Keyword References

Include values associated with model objects (e.g., ins, block, model) using keyword references. These references use the form:

obj ect. val ue

*Objects* are ports of the block being tested, the block itself, or the model in which the block occurs. Each object takes specific *value* types.

The following subsections describe keywords for port objects and values, block objects and values, and model object values.

**Note 1:** Gray rows in any table indicate potential future extensions. If you need to be able to use any of these, contact the HiLiTE team to schedule development.

**Note 2:** Keywords are case-insensitive; however, mixed case spellings increase the readability of long keywords.

#### Port Objects and Values

Ports are referenced as **in***x* or **out***x*, where *x* is a positive integer, or as **self**. Port value keywords are: **op**, **allow**, **IsEvenPort**, **IsOddPort**, **IsInteger**, **IsReal**, **IsBool**, **IsMultiplePtVals**, **start**, **startmin**, **startmax**,



**previous, expected, tolerance.** Op and allow can have subparts: **min, max, mid, ispt.** Value keywords are defined in the table below.

**Examples:** i n1. op. max; sel f. op. mi n; i n2. start; i n3. al l ow. mi d

**Table 11. Port Objects and Values Keywords**

<b>Keyword</b>	<b>Definition</b>
Op, Allow	Op: Signifies either the normal operating range at this port Allow: Signifies the maximum allowable range (e.g., a constraint) at this port
MDVD	subpart of op. The minimum distinct value difference (MDVD) is the minimum difference between two values in the template that guarantees that the actual values in the code under test are/will be different. MDVD is explained further in Section 10.6.6.1.
	gets the maximum allowable range for a port (ex: out1.allow.mid)
Max	subpart of op or allow that specifies the maximum of the range on this port
Min	subpart of op or allow that specifies the minimum of the range on this port
Mid	subpart of op or allow that specifies the mid point of the range: (max + min)/2
IsPt	Sub-part of op or allow that returns true if the range is a point value. ispt only checks to see if the range is a point value; it doesn't check the constrained range.
Examples	out1.Allow.Mid : the maximum allowable range for port out1 in1.Op.Min : the minimum operating range for port in1
Op.Contains(Value)	Returns Boolean value '1', if the operating range of the port contains the given value.
Allow.IsMultiplePtVals	Returns Boolean value '1', if max allow range consists of multiple point values.
Allow.Contains(Value)	Returns Boolean value '1', if the maximum allowable range of the port contains the given value.
IsEvenPort	Returns Boolean value '1' if the port is even port (If the block is having n input ports, then 2 <sup>nd</sup> 4 <sup>th</sup> etc. ports are treated as even ports and other ports are treated as odd port (ex: if(self.IsEvenPort, 0,1)).
IsOddPort	Returns Boolean value '1' if the port is odd port.
IsInteger	Returns Boolean value '1', if the data type at the port is integer (signed and unsigned integer).
IsReal	Returns Boolean value '1', if the data type at the port is Real.
IsBool	Returns Boolean value '1', if the data type at the port is boolean.
Start, StartMin	References the formula (only column or <i>InputX</i> column) for this port in the same test case template step. This allows you to reference the formula specified at one port from within the formula at another port.
Startmax	References the formula ( <i>Max</i> : column) for this port in the same test case template step. This allows you to reference the formula specified at one port from within the formula at another port.
Previous	value on the port in the immediately previous step
Expected	value expected on an output port (not the current port)
Tolerance	amount of variation within which values will be considered the same
Current, First,Second, ...	Used in conjunction with foreach and combinations. Is true (1) if this port is the one of the foreach group selected by the function and false (0) if the port is not of the group. This is actually an abbreviation for the expression "(Index == CurrentIndex)"
Index, CurrentIndex, FirstIndex,	When foreach expressions are evaluated against a multiple port set, these keywords take on values. "Index" gets the index of the port being considered, "CurrentIndex" and "FirstIndex" get the index value of the first / top level iterator

Keyword	Definition
SecondIndex, ....	walking through the array of ports. When using the “combinations” keyword, “SecondIndex” gets the value of the second iterator walking through the array of ports. Up to “TenthIndex” is currently supported, but these only have values if the combinations take that many elements at a time (a combination taking 3 elements at a time would have “FirstIndex”, “SecondIndex”, and “ThirdIndex” defined, but “FourthIndex” and on would not be defined.)

### Block Object and Values

Each formula has only one associated block object—the block under test. The block object reference is **block**. It can have values of **getoutput** or *parameter\_name* (actual parameter name).

**Examples:** block.GetOutput

block.maximum (for fixed range limit block type)

**Table 12. Block Objects and Values Keywords**

Keyword	Definition
GetOutput	the output that the block will produce with the given formulae at the input ports in this test case template step. It can be used only in the formula at an output port. It returns an array of all outputs in order of output port number.
Unique	This is a block-level function that works in conjunction with <i>foreach</i> . It takes two parameters: a port reference and a range reference. (example: block.unique (Input[CurrentIndex], op) ) In this example, the port reference is “port[CurrentIndex]” and the range reference is “op”, for the operating range. This means that the port “Input [CurrentIndex]” must have a unique value among all ports in the array of ports in that array of ports named “Input” and that value must be within the range specified by the operating range of that port.
mdvl, mdvh	These are adjuncts to the port reference in the Unique clause described above. When used, these direct HiLiTE to skew the unique value obtained within that range to the lowest unique value (mdvl) or the highest unique value (mdvh). These are used in situations where the unique value chosen can make a difference in whether the test can be generated or not. If these are not specified, the unique value chosen will be close to the midpoint of the range.
<i>block attributes</i>	Allows you to test for (with?) specific attributes of a block. See Appendix G for lists of available block attributes.
AllInputsInteger	Returns Boolean value ‘1’, if all inputs of the block has integer (signed and unsigned integer) datatype.
AllInputsReal	Returns Boolean value ‘1’, if all inputs of the block has Real datatype.
AllInputsBool	Returns Boolean value ‘1’, if all inputs of the block has Boolean datatype.
AllInputsOpContainZero	Returns Boolean value ‘1’, if the normal operating range of all inputs contain zero.
AllInputsAllowContainZero	Returns Boolean value ‘1’, if the maximum allowable range of all inputs contain zero.
AllInputsContainZero	Returns Boolean value ‘1’, if all inputs of the block can have a zero value. A test case for all 0 inputs should be enabled only when this is true.

Keyword	Definition
SomeInputsRelatedNotTied	This indicates that some inputs of the blocks are related to each other but not tied together. If this is true then a test case for specific or same values at the inputs or all max (or all min) values at the inputs is not possible.
SomeInputsTied	Returns Boolean value '1', if some inputs of the block are tied together. If this is true then a test case for different values at the inputs (hard coded, one max and one min) values at the inputs is not possible.
AllInputsTied	Returns Boolean value '1', if all inputs of the block are tied together.
AllInputsConstant	Returns Boolean value '1', if all inputs of the block are constants (hard wired)
AllInputsInteger	Performs an operation only if inputs are integers and the operation can not be carried out for float inputs.
TotalInPortCount	Returns total count of <b>data</b> port and not the control port
TotalOutPortCount	Returns the count of output ports
InputsRelated	When the inputs of a block get their values from the same source (when there is a fork join), this attribute will be set to TRUE. Otherwise, it will be set to FALSE
ICV_Input_Tied	When the input port and ICV port of a block are tied together, this attribute will be set to TRUE. Otherwise, it will be set to FALSE

### Model Object and Values

Each formula has only one associated model object—the model under test. The model object reference is **model**. A model object can take only two values: **periodms**, which is the period of the model execution frame in milliseconds; and **Ts**, which is the period of the model execution frame in seconds.

**Example:** `model . Ts`

### 10.6.5 Test Case Fields Reference

Multiple Port Elaboration Rule	A formula that is used to generate a class of test cases from one test case template (with new keyword such as <i>foreach</i> and <i>combination</i> )	Expression
Group	Grouping of two or more similar procedures differentiated by special conditions.	Text. Name of an existing group.
Additional Condition	The Condition must be satisfied for procedure to be executed A formula that can be evaluated to determine if this procedure should be used. The result of this condition is ANDed with the result of evaluating conditions for the group this templ proc belongs to.	Expression. See Template Language.
Start At Abs Zero	This option resets the whole model and executes the test case. When enabled, indicates that procedure requires an Initial Condition. Indicates whether the procedure needs to start from absolute step zero.	check/uncheck

## 10.6.6 Definitions and Explanations of Specific Constructs

### 10.6.6.1 MDVD

The minimum distinct value difference (MDVD) is the minimum difference between two values in the template that guarantees that the actual values in the code under test are/will be different. The need for MDVD arises due to the floating point representation error in computations – if floating point values are applied at model inputs and undergo mathematical computations, there can be significant numerical error in the value of at an intermediate signal in the model. MDVD is a default estimate of this numerical error; for a relational operator, if the value of two inputs being compared is apart in the test case template at least by MDVD (e.g.,  $a_{\text{templ}} - b_{\text{templ}} \geq \text{MDVD}$ ), one can have confidence that the actual propagated values in the code will be such that  $a_{\text{code}} > b_{\text{code}}$ . The following formula is used to compute MDVD (note that MDVD is always a positive number):

#### Integer Data Type

$$\text{MDVD} = 1$$

#### Boolean Data Type

MDVD is not defined and should not be used in templates for Boolean data types.

#### Floating Point Data Type

$$\text{MDVD} = 2 * \textit{Tolerance}$$

*Tolerance* is calculated using the formula defined in Section 5.6.1, using the span (max – min) of the operational range in the formula. Note that MDVD is not defined for maximum allowable range (e.g., port.Allow.MDVD is not supported; only port.Op.MDVD is supported)

# 11 Using the HiLiTE Test Harness

The HiLiTE Test Harness (HTH) is used for unit testing of a model's executable object code. It uses the test vectors generated by HiLiTE for the model. HTH is a set of C++ and C source-code files linked with the code generated for a MATLAB Simulink/Stateflow model using MATLAB Real-Time Workshop Embedded Coder (RTW/EC)<sup>1</sup>. Figure 107 illustrates HTH usage and the flow of files.

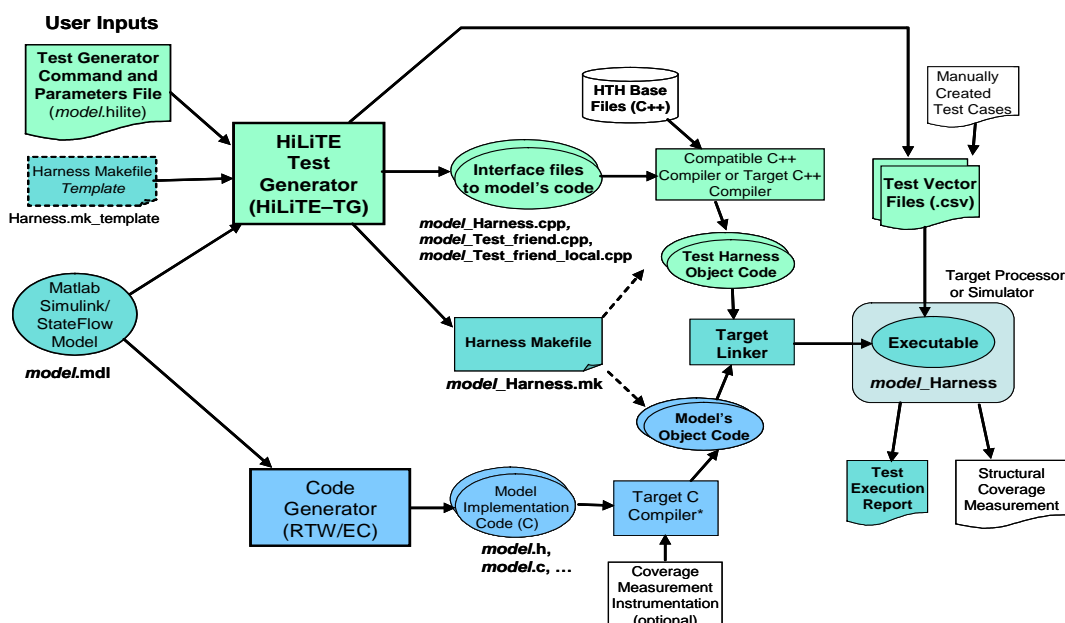


Figure 107 - HiLiTE Test Harness usage and flow of code files

<sup>1</sup> HAM code generation uses the underlying RTW/EC and sets some default options.

The Simulink/Stateflow model “*model.mdl*” on the left side of the figure is input to the HiLiTE test generator. (For this illustration, the name of the model is “*model*,” since it is used as the base name of many generated files.) Note that a separate test harness executable must be created for unit-testing the code for each model in the project.

The following summary gives the sequence to properly execute a unit test. Later subsections give more detail

1. **Generate model code.** Generate code for the model as you normally would use MATLAB RTW-EC or HAM. Code generation creates several C files, including *model.h*, *model.c*, *model\_private.h*, *model\_data.c*, etc.
2. **Generate tests and harness interface code.** Run the HiLiTE Test Generator on the model to produce two sets of artifacts for the harness:
  - a. Files containing the generated test vectors in HiLiTE’s test vector format. Test vector files are named *model\_Vectors.csv*.
  - b. Interface files for the model’s code generated using RTW. These files interface to variables in the model’s code that correspond to model inputs, outputs, and test points. These files are: *model\_Test\_friend.h*, *model\_Test\_friend.cpp*, *model\_Harness.cpp*.
3. **Compile and link model code and harness code.** Use the make files generated by HiLiTE for this purpose.
  - a. Compile model code. The model’s generated code (C) is compiled using the target processor compiler with exactly the same options that will be used for the target executable object code.
  - b. Compile harness code. The harness code consists of harness base code and model-specific code generated by HiLiTE to interface to the model’s code (see Figure 107). Compile this code with the same compiler used for the model’s code or with a compatible C++ compiler. When using the target compiler, include options to enable the harness to use exceptions, run-time libraries, etc.
  - c. Link model code and harness code to build the harness executable. Use the target linker for this purpose.
4. **Run the test harness.** The test harness executable reads the test vectors generated by HiLiTE and executes the tests on the model’s code. Each row of a test vector is a time step with values to be applied at model inputs and expected values of model outputs after the function named *model\_step()* on the model’s code is called. The test harness calls the code and compares the measured values of model outputs with the expected values listed in the test vector. If the measured and expected values are the same for all vectors (time steps), then the unit test passes; otherwise, the unit test fails. The harness generates a test execution report of the overall pass/fail status and diagnostic information for any failures.

## Reference Example

A reference example built around a model named *SimpleExampleModel* is provided with the HTH code. The discussion in the following subsections refers to this example.

## 11.1 Generating Model Code

To generate model code, use the standard MATLAB Simulink/RTW code generation features or the appropriate feature provided by HAM. Place the generated code in the same directory as the model on which you will run HiLiTE.

MATLAB RTW header files are supplied with MATLAB. You must modify the file *PlatformDefinitions.mkinc* to include the path to these files in the compile command. Do *not* use the

*RTW\_Headers* folder supplied with the HiLiTEstHarness. It is included only as an example and may not be compatible with your MATLAB code-generation environment.

## 11.2 Generating Test Vectors and Harness Interface Code

Generate test vectors and harness interface code in HiLiTE using an OptionSet element for HTH under the TestGen command. The following example shows how to include the element in your command file :

```
<Command name=' TestGen' >
  <OptionSet name="HTH">
    <Option key="GenerateHarnessCode">True</Option>
  </OptionSet>
  <Model name=' SimpleExampleModel ' />
</Command>
```

The HiLiTE test generator will first generate the test vector files normally. It then generates an interface code file for each harness. Make sure the model's code files generated using RTW are in the directory before running HiLiTE, since the content of the harness code files depends on the content of the model's code files.

**Table 13. Keys and values available for the *HTH* OptionSet**

Key	Definition and Values
<Option key="HarnessCodePath"> <i>Path</i> </Option>	<i>Path</i> where HiLiTE generates code for the test harness. Default is the project path, e.g. where the model is. Note that keyword substitution is performed on the path string—the name of the model is substituted for a substring "{model}".
<Option key="UY_IO_Style">	<i>True</i> value denotes that model's code was generated by MATLAB where all model inputs are declared in a global structure named *_U and outputs in structure named *_Y. The default is <i>False</i> , which denotes that the model inputs and outputs are declared in the code directly as individual global variables.
<Option key="GenerateFromTemplate"> <i>Tailname.ext_template</i> </Option>	The value is the name of a template that is keyword substituted (see 11.3.4) to create a file with a name <i>model_Tailname.ext</i> , where <i>model</i> is the name of a model. Note that the value must end with the string "_template".

## 11.3 Compiling and Linking Model and Harness Code

You are now ready to compile the code for the model and the harness, then link them before running the test harness.

### 11.3.1 Compiling Harness Code

The test-harness code consists of base code supplied in the HiLiTE installation and model-specific code generated by HiLiTE:

Harness base code consists of these files:

```
HTH_Platform.h
HTH_Utils.h
HTH_Typed_data.h
HTH_Test_driver.h
HTH_Test_friend_base.h
HTH_Test_types.h
HTH_Platform.cpp
HTH_Utils.cpp
HTH_Test_driver.cpp
```



```

HTH_Test_friend_base.cpp
HTH_Typed_data.cpp

```

Model-specific harness code generated by HiLiTE consists of these files:

```

modelname_Test_friend.h
modelname_Test_friend.cpp
modelname_Harness.cpp

```

**Contents of file *HTH\_Platform.h*.** In the base harness code, the file *HTH\_Platform.h* handles dependencies on the target compiler environment and platform. You may need to add to this file for a new platform; *please send all changes to the HiLiTE team so that we can fold those in the next release*. Any additional support functions for a platform can be put in *HTH\_Platform.cpp*. There is currently no content in these files.

**Compiling Base Harness Code.** Compile base harness code once for all the models in the project. The object files for the base classes can be arranged in a library archive so that they can be linked with multiple harness executables for different models in the project. *Makefile* in the *HiLiTETestHarness* directory builds the base harness code and creates a library archive file. This *Makefile* in turn includes another file that contains target-platform-specific definitions (see *A makeinclude File for Target-Platform-Specific Definitions*).

**Compiling Model-Specific Harness Code.** This code is compiled for each model separately. Note that a specific makefile to compile the model-specific harness code can be automatically generated by supplying a harness-makefile template to HiLiTE during test generation.

**Harness Compiler and Options.** All harness code is in ANSI-Standard C++ and uses the C++ *std* library. Thus a C++ version of the compiler must be used with options (if necessary) to include the runtime library for *std*. No other special compiler options are needed, unless required by a specific target compiler.

#### 11.3.1.1 A makeinclude File for Target-Platform-Specific Definitions

Use the makeinclude file named *PlatformDefinitions.mkinc*, found in %HILITE\_HOME%\HiLiTETestHarness, to create a makefile for building the test harness for target platforms<sup>2</sup>. This file contains the definitions for the platform-specific compilers, linkers, and compiler flags to use. It also contains definitions for locations of project-specific directories. The default settings in the supplied version of this file are to use the GNU compiler for PC/Desktop platform. ***You must edit this file before doing any compiles.*** Comments in the file describe how and what needs to be modified.

#### 11.3.2 Compiling Model Code

Generate code for the model with MATLAB Simulink RTW/EC, either independently or via HAM controls. Compile the code using the target processor compiler with exactly the same options you will use for the airborne executable object code for the model. Certain RTW header files (for RTW run-time libraries) are referenced by this code, so make sure you include appropriate references to the correct version of these files—the same as those for the target-system version of the code. Note that HAM/RTW may generate a model-specific makefile (named *model.mk*) to compile this code; we do not recommend using this makefile. Any makefile to compile target environment needs to have target-platform-specific information. Note that a specific makefile to compile the model's code can be automatically generated by supplying a target-makefile template to HiLiTE during test generation. See Section 11.3.4.

---

<sup>2</sup> The initial example for this was provided by the EAJ7/AGT1500 users.

### 11.3.2.1 Separating Model Code from Harness Code

We recommend that you compile the model's code separately from the harness code so that the separation between airborne code and harness code is clear during unit testing.

HiLiTE Test Harness does not require the model's code to be compiled at the same time as the harness code. If object files (or an object library) for the model's code are already created through the build process, they can be appropriately referenced when linking the test harness executable. Section 11.3.4 describes how to create a single or multiple model-specific makefiles using HiLiTE when you supply a project-specific makefile template to HiLiTE.

### 11.3.3 Linking all Object Code to Create Test Harness Executable

The target linker is used to link the base harness code, model-specific harness code, and model's code into an executable named *model\_Harness*. For the reference example, this executable is called *SimpleExampleModel\_Harness*. The recommended directory location for this executable is the same directory as the model and associated source code files. For the reference example, HiLiTE automatically generates a makefile using the supplied template. This makefile compiles all model-specific code and builds the harness executable.

### 11.3.4 Creating Model-Specific Makefiles for the Compile and Link Steps

You can generate any number of makefiles (or other files) by supplying templates (file names end in *\_template*, e.g. *File1.mk\_template*) for these files to HiLiTE. HiLiTE substitutes keywords in each file and creates an output file of the same name as the template file (without the trailing "*\_template*"). In the following example, HiLiTE will create two files named *MyModel\_File1.mk* and *MyModel\_File2.mk* using the templates.

```
<Command name=' TestGen' >
  <OptionSet name="HTH">
    <Option key="GenerateHarnessCode">True</Option>
    <Option key="GenerateFromTemplate">File1.mk_template</Option>
    <Option key="GenerateFromTemplate">File2.mk_template</Option>
  </OptionSet>
  <Model name=' MyModel ' />
</Command>
```

The HiLiTE installation supplies the makefile template *Harness.mk\_template* with the reference example. This template generates the makefile named *SimpleExampleModel\_Harness.mk* when HiLiTE is run. You can modify this template as needed.

The following keywords are substituted in the template file (note: each keyword starts with "%%"):

Keyword	Substitution
%%ModelName	Name of the model
%%CodeGenVersion	Version of HiLiTE
%%DateTime	Date and time of generation

## 11.4 Running the Test Harness Executable

The linked test harness executable is a command line application or a target application, depending upon the target platform used for testing. The application is named after the model in this format: *model\_Harness*. It reads the test vector file(s) generated by HiLiTE and applies the test vectors to the model's code. An output file containing the results of verification is created whose name defaults to: *test-vector-file<extension removed>\_results.extension*.

To run the harness for the supplied reference example on the desktop platform, use this command line:

```
> simpleExampleModel_Harness TestVectors\SimpleExampleModel_Vectors.csv
```

(for platforms that do not support command line usage, *see note at end*)

This command produces a test result file named TestVectors\SimpleExampleModel\_Vectors\_Results.csv.

## Usage

```
model_Harness [options] [<test-vector-file> [<base-dir-name>]] [std i/o redirection]
```

## Options

Use this option:	To:	Where:
-h	print help	
-n	suppress file I/O; use StdInput to read test vectors and StdOutput for reports	
-a<prefix>	derive as-built vector file(s) using answers from this run	As-built is a (set of) test vector file(s) identical to the one being executed, except the expected results are updated to match those actually measured during the test, and with file names permuted by <prefix>. This is useful for correcting typos when manually generating scripts.
-l	activate internal state machine logging	
-o[opts]	create Output file info:(t)race, (r)esults, input (l)ines. Append '-' to disable options, e.g. "-or-l"=lines, no results default is "results"	-o sends any combination of several classes of information to the output file and/or stdout respectively.
-s[opts]	create Stdout info (see Output file info)	-s sends any combination of several classes of information to stdout.
-b<name>	override output file base name	The output (results) file defaults to test-vector-file<extension removed>_result.csv. -b permits an alternate base output file name to be specified.

## Note for test environments not supporting normal command line usage:

Command line usage will vary depending upon the target environment. If the test harness was compiled for a target processor and the resulting executable is run in a simulator, it may not be possible to specify command line options or read/write specific files. In this case, the test harness will default to stdio and stdout for reading the input and writing output. In such instances it may be necessary to specify to the simulator where to read the input file from and where to direct the output to.

For instance, with *SingleStep* simulator, you would use the following commands to specify input/output files:

```
targetio -i model_vectors.csv
targetio -o model_output.txt
```

## 11.5 Tolerance on Measurement of Output Values

An expected output value is specified in an output column in the *Vector* file. This value is compared with the actual (measured on code under test) value of this output variable by the test harness as described in Section 5.6. As described in the HiLiTE Test Vector File Format document [8], a built-in default value of *0.0001 relative tolerance (0.01% tolerance)* is used. To override this, a specific individual-column *relative tolerance* values can be present in a test vector file to denote a relative tolerance to be applied to actual output values for that particular output column.

Regardless of how the tolerance value is determined for an output column, the HiLiTE Test Harness uses the following algorithm for comparison of expected output values with actual output values in that column. (Note: in the following algorithm, the keyword *Tolerance* denotes the relative tolerance determined for a column):

$$\text{AbsoluteTolerance} = \text{Max}(\text{Abs}(\text{ActualOutputValue} * \text{RelativeTolerance}), \text{RelativeTolerance})$$

**For an actual output value pass, the following must be true:**

$$\begin{aligned} &\text{ActualOutputValue} - \text{AbsoluteTolerance} \leq \text{ExpectedOutputValue} \\ \text{AND} \\ &\text{ExpectedOutputValue} \leq \text{ActualOutputValue} + \text{AbsoluteTolerance} \end{aligned}$$

### 11.5.1 Overriding the Built-In Default Tolerance

Section 5.6 describes how a tolerance value can be specified as an input to HiLiTE. The HiLiTE Test Generator creates a corresponding entry in a *Tolerance* row in the model's test vector file as shown in Figure 108. The file element *Tolerance* specifies the tolerance for each vector-aligned output column.

ColumnContent	Repetition	Input:1	Input:2	Input:3	Input:4	Input:5	Input:6	Output:1	Output:2
ColumnName		table1_in	table2_in	sum1_in	product1	product_ir	sum_in	product1	product_out
Comment	Tests for limit								
Comment	Test Case AboveMax								
Tolerance									0.001
Vector	1	-112	85.9596	1098.871	X	X	-129.453	X	X

**Figure 108 - Specifying tolerance in test vector file (Modelname\_Vectors.csv)**

The HiLiTE Test Harness processes information in this way: 1) If the value in a Tolerance column is blank, the current value of DefaultTolerance is used. 2) If the DefaultTolerance element is not specified in the test vector file, the built-in default tolerance of .0001 is used for all output columns. Note that a subsequent assignment of the DefaultTolerance value will have no effect unless followed by a tolerance file element.

When a Tolerance file element is encountered, the column tolerance assignments are recomputed. In the event that a prior tolerance file element has already been processed, any compatibility validation regarding the configuration changes must not occur until a vector file element is encountered.

In the test vector file shown in Figure 108, a tolerance of 0.001 is specified for Output: 2. All other columns are left empty, which implies that the built-in default tolerance is used for those columns.

#### Example1:

If the output value is 0.5, and specified tolerance is 0.001. Then

AbsoluteTolerance = Max (0.5 \* 0.01, 0.001) = 0.005. So any actual output value between (0.5 – 0.005) and (0.5 + 0.005) will pass.

**Example2:**

This is a case where a test vector failure is reported by HiLiTE Test Harness as shown below

Vector	1	1098.8721	2.5	-129.4542	X	0.4	Failure product_out (Output:2) expected 0.4 found 0.0.3967	FAILED
--------	---	-----------	-----	-----------	---	-----	---	--------

Here the actual output value = 0.3967. In this case, you can specify the tolerance for this output column in such a way that 0.4 passes. If you specify tolerance = 0.01, then Product =  $0.3967 * 0.01 = 0.003967$ . AbsoluteTolerance = Max (0.003967, 0.01) = 0.01. Now all actual values between 0.3867 and 0.4067 will be considered valid for the expected output of 0.4.

## 12 Using the TPI Converter

When HiLiTE cannot achieve 100% test coverage of flight code, it generates additional test cases by turning on test points in the model. You can use the test point-enabled models to generate additional test cases, but those test cases must be executed against the flight code (which does not have test points enabled) and externally observable results must be verified before the additional test cases can be used. For more information about TPI converter usage, refer to [1]

When HiLiTE generates test cases:

- Two sets of test vectors will be generated. One set is the normal set; the other is the augmented set that results from turning on Test Points in the model.
- These two sets (i.e. files) of vectors will be combined into a single set.
- The observation points that resulted from turning on test points will be removed from the test vectors (as they do not exist in the flight code).
- The actual outputs corresponding to the model's outputs for the Test Points ON set of test vectors are replaced with the outputs from the CSV file which is a result of TPI run on CTP Generator (either of PPC or MIPS processor).

This set of test vectors can be run against the real flight code

TPI Converter is a tool that converts a test point ON set of tpi to test point OFF set of tpi. TPI Converter imports TP ON, TP OFF, and CTP Output files and generates a final TPI file.


### Files required to use the tool

- **exe file:** This is the file used to generate the converted TPI file.
- **ini file:** This is the initialization file used by the tool to know the format of the TPI file and where the input files are present along with the path where and output should be generated. It also includes information on tolerance. For more details, Ref 12.4

## Starting the TPI Converter

To launch the TPI Converter, double-click the icon (exe file).

## Closing the TPI Converter

Click the  button at the right top of the window. Or select menu option File->Exit.

## 12.1 Inputs

### TPI generated with Test Points ON Option

The input variables in the TPI file generated with the test points ON are Model Level Inputs. If the Unit Delay or Stim override options are used, the inputs will also include the override inputs. The output variables are the model level outputs. The test points in the TPI file generated with the Test Points ON comprise of Block Level Test Points and Model Level Test points.

### TPI generated with Test Points OFF Option

The input variables in the TPI file generated with the test points OFF are Model Level Inputs. The output variables are the model level outputs. The test points in the TPI file generated with the Test Points OFF comprise of only Model Level Test points.

### CSV file

When the CTP tool is run on a TPI generated with test points ON a .csv file is created. This file contains values for model outputs and test points. It should have the same number of test cases as the TPI file generated with test points ON. For example, if the number of test cases in TPI\_ON file is 100, the number of rows in the .csv file should also be 100.

## 12.2 OUTPUT

### Final TPI

- In the final TPI generated, inputs remain the same for the TP\_ON TPI files as the model and override inputs.
- The output variables comprise only model outputs and model level test points and are similar to the TP\_OFF file.
- Test case counts and the test cases should be same as in the TP\_ON file.
- All the set outputs should be set as in CSV file.
- Final TPI output and tolerance are decided as follows:
  - If a test case in TP\_ON TPI has all outputs as don't cares, then the converted TPI file will have all output as don't care for the same test case.
  - If all measurable variables are common in TP\_ON and TP\_OFF files, then TPI Converter will not expand the test case and all the values will be picked up from the TP\_ON TPI.
  - If none of the measurable variables are common in TP\_ON and TP\_OFF files, then test cases will be expanded. If output variables common between TP\_ON and TP\_OFF vary, the test case will not be expanded.
  - If some (but not all) of the measurable variables are common, then the values for common variables will be picked up from the TP\_ON TPI and those that are not in common will be picked up from CTP-generated .csv file.
  - If the magnitude of the output value is less than  $2 \times 10^{31}$ , then the converter renders it in a normal number format; if the magnitude is more than  $2 \times 10^{31}$ , then it will be given in scientific notation. This provides proper number format for integers and Booleans.

## Log.csv

A **log.csv** file is created every time you run the TPI Converter tool from the user interface. Whatever is logged in the UI's list control will be logged in log.csv. The file includes successful completion status of commands as well as errors. The log.csv will be created in the same folder as TpiConverter.exe.

## 12.3 Using the TPI Converter

### 12.3.1 Command Line Usage

Command line usage for the TPI Converter (file or folder wise) requires four arguments separated by a single space.

- First argument: TP\_ON tpi file path (file/folder)
- Second argument: TP\_OFF tpi file path (file/folder)
- Third argument: CTP\_Output csv file path (file/folder)
- Fourth argument: Final TPI path where the final toi should be generated

Keyword associated for command line usage with file and folder is **-file:** and **-folder:**

**Note: Tool accepts only absolute path and no relative path (For both folder and file wise)**

#### Example folder paths:

```
D: \>D: \Hi Li TE_B2\Development\Tools\TPI_Converter\Tpi Converter. exe  
- folder: D: \TPI_Converter_Testing\TP_ON\  
D: \TPI_Converter_Testing\TP_OFF\ D: \TPI_Converter_Testing\CTP_Output\  
D: \TPI_Converter_Testing\Converter_Output\
```

Note: Back slash must be included at the end when specifying the path to the folder

#### Example file paths:

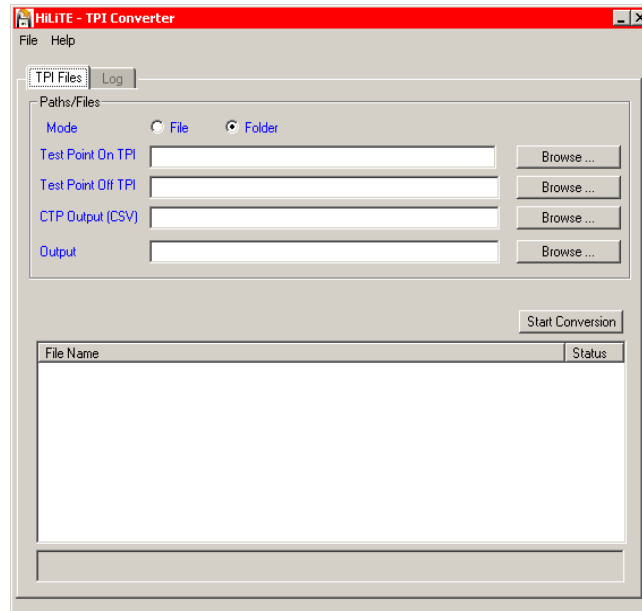
```
D: \>D: \Hi Li TE_B2\Development\Tools\TPI_Converter\Tpi Converter. exe -file:  
D: \TPI_Converter_Testing\TP_ON\ PF_TPI_Convert_Test_ctp. tpi  
D: \TPI_Converter_Testing\TP_OFF\ PF_TPI_Convert_Test_ctp. tpi  
D: \TPI_Converter_Testing\CTP_Output\ PF_TPI_Convert_Test_ctp_RESULTS. CSV  
D: \TPI_Converter_Testing\Converter_Output\ PF_TPI_Convert_Test_ctp. tpi
```

### 12.3.2 Using the TPI Converter Interface

The TPI Converter graphical user interface (GUI) consists of two tabbed pages:

- **TPI Files** is used to select the names of the files to use as inputs to the TPI and to perform the actual conversion.
- **Log** displays the errors generated during TPI conversion.





**Figure 109 - GUI for TPI Converter**

### Creating TPI Files

Use the TPI Files page of the TPI Converter interface to select the files to convert. You can enter specific paths or accept default values.

Note: The TPI Converter accepts only absolute folder and file paths; do not use relative paths.

Complete the fields at the top of the page, then click the **Start Conversion** button to complete the process.

### Field descriptions

**Mode** The selection you make here determines how you will browse to select paths in the next four fields. Click the **File** button to browse for files. Click the **Folder** button to browse for folders.

**Test Points ON TPI** Complete this text box with the path to the ON TPI file. Click the **Browse** button to find and select the path.

**Default Value:** Taken from the ini file section for the Default value: [Path And File]

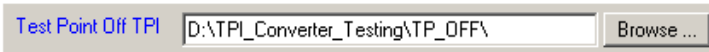
For example:

Through Tool	INI File
<div> Test Point On TPI <input type="text" value="D:\TPI_Converter_Testing\TP_ON\"/> <input type="button" value="Browse ..."/> </div>	[Path And File] <b>Test Points On</b> = D:\TPI_Converter_Testing\TP_ON\ Test Points Off = D:\TPI_Converter_Testing\TP_OFF\ CTP Output = D:\TPI_Converter_Testing\CTP_Output\ Output = D:\TPI_Converter_Testing\Converter_Output\

**Test Points OFF TPI** This text box accepts the path of the folder for the Test Points OFF TPI file. Click the **Browse** button to find and select the path.

**Default Value:** Taken from the ini file section for the Default value: [Path And File]

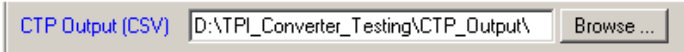
For example:

Through Tool	INI File
	[Path And File] Test Points On = D:\TP_Converter_Testing\TP_ON\ <b>Test Points Off = D:\TPI_Converter_Testing\TP_OFF\</b> CTP Output = D:\TP_Converter_Testing\CTP_Output\ Output = D:\TP_Converter_Testing\Converter_Output\

**CTP Output (CSV)** This text box accepts the path of the folder for the CTP\_Output file. Click the **Browse** button to find and select the path.

**Default Value:** Taken from the ini file

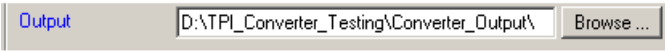
For example:

Through Tool	INI File
	[Path And File] Test Points On = D:\TPI_Converter_Testing\TP_ON\ Test Points Off = D:\TP_Converter_Testing\TP_OFF\ <b>CTP Output = D:\TPI_Converter_Testing\CTP_Output\</b> Output = D:\TPI_Converter_Testing\Converter_Output\

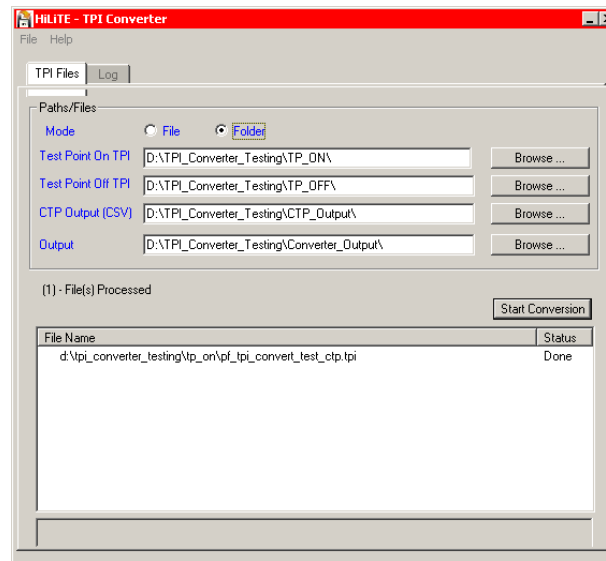
**Output** This text box accepts the path of the folder in which the Output file is generated. Click the **Browse** button to find and select the path.

**Default Value:** Taken form the ini file

For example:

Through Tool	INI File
	[Path And File] Test Points On = D:\TPI_Converter_Testing\TP_ON\ Test Points Off = D:\TPI_Converter_Testing\TP_OFF\ CTP Output = D:\TPI_Converter_Testing\CTP_Output\ <b>Output = D:\TPI_Converter_Testing\Converter_Output\</b>

**Start Conversion** button. After selecting all input files, click Start Conversion to generate output files. The TPI Converter displays the names of the output files and their status in the list box below the button.



**Figure 110 - TPI Files Tab**

### 12.3.3 Example

Input 1 – TPI File with TP ON set of vectors

Input 2 – TPI File with TP OFF set of vectors

Input 3 – csv File which is a result of TPI run on CTP Generator (either of PPC or MIPS processor) with Test Points ON set of test vectors

Input 4 – Output Directory Path

Output 1 – TPI file containing the combined set of test vectors to be run against the real flight code.

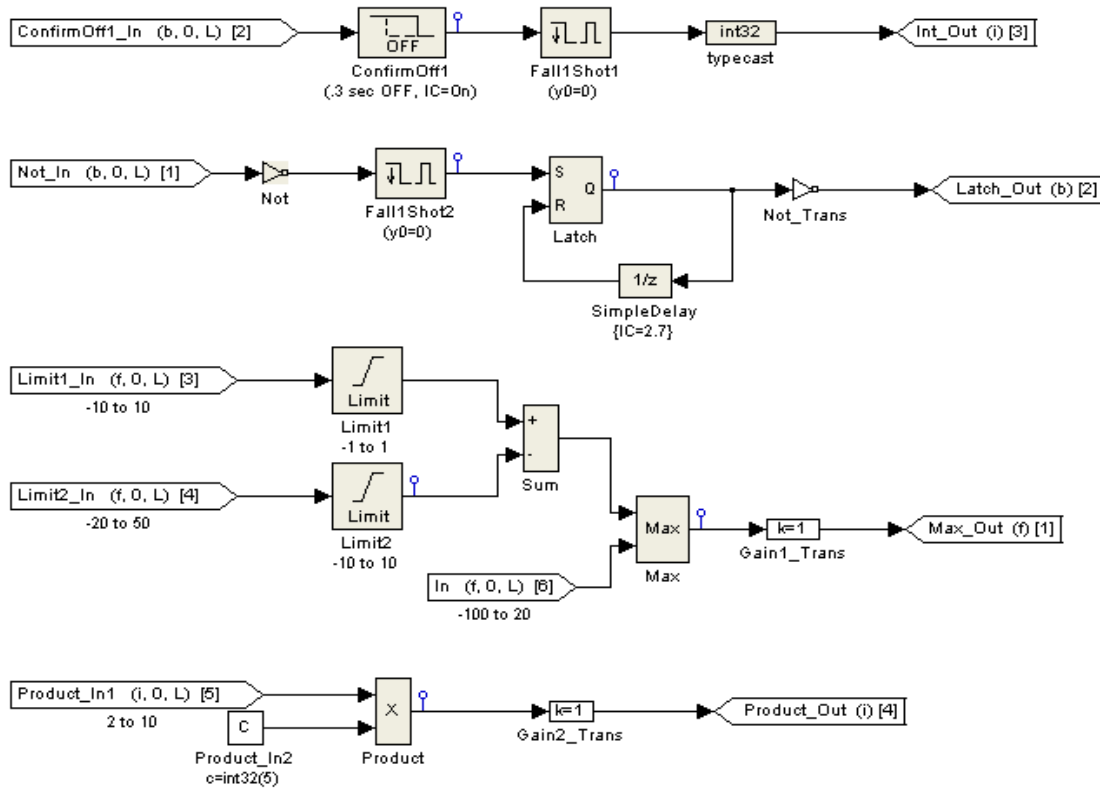


Figure 111 - Simulink Model

```

/* Block signals (auto storage) */
typedef struct {
    real_T Limit_1;           // BLOCK LEVEL Test Point
    real_T n_Limit_1;        // MODEL LEVEL Test Point
    real_T Sum_1;            // BLOCK LEVEL Test Point
    real_T Max_1;            // MODEL LEVEL Test Point
    real_T Gain_1;          // BLOCK LEVEL Test Point
    int32_T tCast_1;         // BLOCK LEVEL Test Point
    int32_T Product_1;       // MODEL LEVEL Test Point
    int32_T o_Gain_1;        // BLOCK LEVEL Test Point
    uint32_T cOffSum_1;      // BLOCK LEVEL Test Point
    boolean_T Not_1;         // BLOCK LEVEL Test Point
    boolean_T Fall_1;        // MODEL LEVEL Test Point
    boolean_T SimpleDelay_1; // BLOCK LEVEL Test Point
    boolean_T Latch_1;       // MODEL LEVEL Test Point
    boolean_T e_Not_1;       // BLOCK LEVEL Test Point
    boolean_T confirmOFF_1;  // MODEL LEVEL Test Point
    boolean_T d_Fall_1;      // BLOCK LEVEL Test Point
} BlockIO_TPI_Convert_Test_Tolerance;

```

Figure 112 - Test Points in the generated code.

**Example :** Table 14 contains the variables associated with TP\_ON tpi and TP\_OFF tpi. Similar variables in two files are matched with the same color code.

**Table 14. Matching of test points in TPI\_ON and TPI\_OFF files**

TestPoints OFF (Outputs + Model level Test Points )			
TYPE	Data type	Name	Count Sequence
Outputs	Float64	TPI_Convert_Test_Tolerance_Y.Max_Out	1
	Bool	TPI_Convert_Test_Tolerance_Y.Latch_Out	2
	Int32	TPI_Convert_Test_Tolerance_Y.Int_Out	3
	Int32	TPI_Convert_Test_Tolerance_Y.Product_Out	4
	Float64	TPI_Convert_Test_Tolerance_Y.Double_Out	5
TestPoints: (Only model Level Test Points)	Bool	TPI_Convert_Test_Tolerance_B.confirmOFF_1	6
	Bool	TPI_Convert_Test_Tolerance_B.Fall_1	7
	Bool	TPI_Convert_Test_Tolerance_B.Latch_1	8
	Float64	TPI_Convert_Test_Tolerance_B.n_Limit_1	9
	Float64	TPI_Convert_Test_Tolerance_B.Max_1	10
	Int32	TPI_Convert_Test_Tolerance_B.Product_1	11
TestPoints ON ((Outputs + Model level Test Points +Blocks level ) )			
TYPE	Data type	Name	Count Sequence
Outputs	Float64	TPI_Convert_Test_Tolerance_Y.Max_Out	1
	Bool	TPI_Convert_Test_Tolerance_Y.Latch_Out	2
	Int32	TPI_Convert_Test_Tolerance_Y.Int_Out	3
	Int32	TPI_Convert_Test_Tolerance_Y.Product_Out	4
	Float64	TPI_Convert_Test_Tolerance_Y.Double_Out	5
Test Points ( Model level Test Points+ Block Level Test Points)	Bool	TPI_Convert_Test_Tolerance_B.confirmOFF_1	6
	Bool	TPI_Convert_Test_Tolerance_B.d_Fall_1	7
	Bool	TPI_Convert_Test_Tolerance_B.Fall_1	8
	Bool	TPI_Convert_Test_Tolerance_B.Latch_1	9
	Float64	TPI_Convert_Test_Tolerance_B.Limit_1	10
	Float64	TPI_Convert_Test_Tolerance_B.n_Limit_1	11
	Float64	TPI_Convert_Test_Tolerance_B.Max_1	12
	Bool	TPI_Convert_Test_Tolerance_B.Not_1	13
	Int32	TPI_Convert_Test_Tolerance_B.Product_1	14
	Bool	TPI_Convert_Test_Tolerance_B.SimpleDelay_1	15
	Float64	TPI_Convert_Test_Tolerance_B.Sum_1	16

In Table 15, the original CSV file has been edited and color codes have been used to match the test points in TP\_ON and TP\_OFF files.

**Note:** We have modified the CSV with values that can be easily identified in the TPI.

**Table 15. An excerpt from the CSV files containing only model outputs and model level test points**

1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6	8	9	11	12	14
Max_Out	Latch_Out	Int_Out	Product_Out	Double_Out	confirmOFF_1	Fall_1	Latch_1	n_Limit_1	Max_1	Product_1
1	2	3	4	5	6	8	9	2.3	3.4	5.6
11	22	33	44	55	66	88	99	23	34	56
111	222	333	444	555	666	888	999	232	343	565
1111	2222	3333	4444	5555	6666	8888	9999	2323	3434	5656
11111	22222	33333	44444	55555	66666	88888	99999	23232	34343	56565

Table 16 shows columns that were removed for better understanding.

**Table 16. An excerpt from the CSV file containing the moved columns**

NA	NA	NA	NA	NA
7	10	13	15	16
d_Fall_1	Limit_1	Not_1	SimpleDelay_1	Sum_1
7	1.2	4.5	6.7	7.8
77	12	45	67	78
777	121	454	676	787
7777	1212	4545	6767	7878
77777	12121	45454	67676	78787

### Converted TPI:

If a test case in TP ON TPI has all outputs as don't cares, then in converted TPI file will have all output as don't care for that particular test case.

If all measurable variables are common in TP On and TP Off file, then TPI Converter will not expand the test case and all the values will be picked up from TP ON TPI.

If none of the measurable variables are common in TP ON and TP OFF file, then test Case will be expanded. If any of the output variable (common between TP ON and TP Off) varies otherwise the test case will not be expanded.

If some (not all) of the measurable variables are common then the values for common variable will be picked up from TP ON TPI and for those which are not common will be picked up from CTP generated CSV file.

**Example1: TP On Test case**

```

*****
;
; Test Case #    1
; Testing       : TP_Convert_Test_Tolerance/ConfirmOff1
; Description   : TimerExpireHoldAndReset
; Inputs        : 2)TPI_Convert_Test_Tolerance_U.ConfirmOff1_In
; Outputs       : 6)TPI_Convert_Test_Tolerance_B.confirmOFF_1
; TestReference : Test full normal operation sequence of fixed timer
; Type         :
; Requirements  :
*****
;Loop 1
; INPUT VARIABLES for testcase 1
  X   1 X   X   X   X   X
; OUTPUT VARIABLES for testcase 1
  X X X X X   X X X X X X X X X X
; TOLERANCE VALUES for testcase 1
  X X X X X   X X X X X X X X X X

```

**Converted TPI of example1.**

```

*****
;
; Test Case #    1
; Testing       : TP_Convert_Test_Tolerance/ConfirmOff1
; Description   : TimerExpireHoldAndReset
; Inputs        : 2)TPI_Convert_Test_Tolerance_U.ConfirmOff1_In
; Outputs       : 6)TPI_Convert_Test_Tolerance_B.confirmOFF_1
; TestReference : Test full normal operation sequence of fixed timer
; Type         :
; Requirements  :
*****
;Loop 1
; INPUT VARIABLES for testcase 1
  X   1 X   X   X   X   X
; OUTPUT VARIABLES for testcase 1
  X X X X X   X X X X X X
; TOLERANCE VALUES for testcase 1
  X X X X X   X X X X X X

```

### Example2: TP ON TPI Test case

```

.*****
;
; Test Case #    2
; Testing       : TP_Convert_Test_Tolerance/ConfirmOff1
; Description    : TimerExpireHoldAndReset
; Inputs        : 2)TPI_Convert_Test_Tolerance_U.ConfirmOff1_In;
; Outputs       :2) TPI_Convert_Test_Tolerance_Y.Latch_Out, 4)
TPI_Convert_Test_Tolerance_Y.Product_Out, 5) TPI_Convert_Test_Tolerance_Y.Double_Out,9)
TPI_Convert_Test_Tolerance_B.Latch_1,14) TPI_Convert_Test_Tolerance_B.Product_1
; TestReference : Test full normal operation sequence of fixed timer
; Type          :
; Requirements   :
.*****
;
;Loop 1
; INPUT VARIABLES for testcase 2
X   1 X   X   X   X   X
; OUTPUT VARIABLES for testcase 2
X   1 X   77  22   X X   X   0   X   X   X   X   12   X   12.44
; TOLERANCE VALUES for testcase 2
X   0.04   X   0.12  0.023   X X   X   00.14X   X   X   X   X   0.005   X   X

```

### Converted TPI of example 2:

```

.*****
;
; Test Case #    2
; Testing       : TP_Convert_Test_Tolerance/ConfirmOff1
; Description    : TimerExpireHoldAndReset
; Inputs        : 2)TPI_Convert_Test_Tolerance_U.ConfirmOff1_In;
; Outputs       :2) TPI_Convert_Test_Tolerance_Y.Latch_Out, 4)
TPI_Convert_Test_Tolerance_Y.Product_Out, 5) TPI_Convert_Test_Tolerance_Y.Double_Out,8)
TPI_Convert_Test_Tolerance_B.Latch_1,11) TPI_Convert_Test_Tolerance_B.Product_1
; TestReference : Test full normal operation sequence of fixed timer
; Type          :
; Requirements   :
.*****
;
;Loop 1
; INPUT VARIABLES for testcase 2

```



```

X   1 X   X   X   X   X
; OUTPUT VARIABLES for testcase 2
11   1 33   77   22   66   88 0    23 34 12
; TOLERANCE VALUES for testcase 2
0.22 0.04 0.66 0.12 0.023 1.32 1.76 00.14 0.26 .068 0.005

```

**Example3: TP ON TPI test case**

```

,*****
;
; Test Case #    2
; Testing       : TP_Convert_Test_Tolerance/ConfirmOff1
; Description    : TimerExpireHoldAndReset
; Inputs        : 2)TPI_Convert_Test_Tolerance_U.ConfirmOff1_In;
; Outputs       : 2) TPI_Convert_Test_Tolerance_Y.Latch_Out, 4)
TPI_Convert_Test_Tolerance_Y.Product_Out, 5) TPI_Convert_Test_Tolerance_Y.Double_Out,9)
TPI_Convert_Test_Tolerance_B.Latch_1,14) TPI_Convert_Test_Tolerance_B.Product_1
; TestReference : Test full normal operation sequence of fixed timer
; Type          :
; Requirements   :
,*****
;
; Loop 2
; INPUT VARIABLES for testcase 2
X   1 X   X   X   X   X   X
; OUTPUT VARIABLES for testcase 2
X   1 X   77   22   X X   X   0   X   X   X   X   12   X   12.44
; TOLERANCE VALUES for testcase 2
X   0.04 X   0.12 0.023 X X   X   00.14X   X   X   X   X   0.005 X   X

```

**Converted TPI of example 3:**

```

,*****
;
; Test Case #    2
; Testing       : TP_Convert_Test_Tolerance/ConfirmOff1
; Description    : TimerExpireHoldAndReset
; Inputs        : 2)TPI_Convert_Test_Tolerance_U.ConfirmOff1_In;
; Outputs       : 2) TPI_Convert_Test_Tolerance_Y.Latch_Out, 4)
TPI_Convert_Test_Tolerance_Y.Product_Out, 5) TPI_Convert_Test_Tolerance_Y.Double_Out,8)
TPI_Convert_Test_Tolerance_B.Latch_1,11) TPI_Convert_Test_Tolerance_B.Product_1
; TestReference : Test full normal operation sequence of fixed timer
; Type          :
; Requirements   :
,*****
;

```

```
;Loop 1
; INPUT VARIABLES for testcase 2
  X   1 X   X   X   X   X
; OUTPUT VARIABLES for testcase 2
  11   1 33   77   22   66   88 0   23 34 12
; TOLERANCE VALUES for testcase 2
0.22 0.04 0.66 0.12 0.023 1.32 1.76 00.14 0.26 .068 0.005

;Loop 1
; INPUT VARIABLES for testcase 3
  X   1 X   X   X   X   X
; OUTPUT VARIABLES for testcase 3
  11   1 33   77   22   66   88 0   23 34 12
; TOLERANCE VALUES for testcase 3
0.22 0.04 0.66 0.12 0.023 1.32 1.76 00.14 0.26 .068 0.005
```

### Tolerance:

Tolerance is calculated as per the ini file entry. By default, for Boolean and Integers, it is 0

Float 32/Float 64 it is 0.02 based on the ini file entry.

## 12.4 INI file

This is the initialization file where the format of the TPI file can be specified. Also the default path of the input and output files can also be specified here.

### [Common Options]

- **Model Initialization = Initialize** - This entry takes the keyword appearing in the tpi file to initialize test cases. Currently it is **Initialize**
- **Append Text For CTP Output File = \_RESULTS** - This entry takes the string appended in the file name which is generated by the CTP output generator that has the values for all the output variables values of the model in a CSV format. Currently it is **RESULTS**
- **TPI Comment Character = ;** - This entry takes the comment character appearing in the tpi file. The current comment character is a semicolon ( ; )
- **Variable Initialization Section Mandatory = 1** - This entry allows to configure the Variable Initialization section to be Mandate or not. If this option is set to 1, then TPI Converter will not generate the converted TPI if the Variable Initialization section is not present in the TP ON TPI file. If this options is set to 0, then TPI Converter will generate the converted TPI but log a warning saying the Variable Initialization is not present hence this might result in some vector failures.

### [Tolerance]

- **Tolerance Calculation Enabled** - This entry takes two values
  - 1** - Tolerance values are calculated as per below mentioned ini file entries
  - 0** - Tolerance values are not calculated and default values are used

- **Data Type Count for Tolerance** -This entry takes the count of the data types for which the user wants the tolerance to be calculated. If the count is 2, we can provide the Data Type for both by appending the count to end of the keyword **DataType**. For ex – **DataType1**, **DataType2** and so on. We should assign the data types for them as

**DataType1 = Float32** -This is the first data type for which the user want the tolerance to be calculated

**DataType2 = Float64** -This is the second data type for which the user want the tolerance to be calculated

... so on

- **Tolerance in percentage of absolute value for DataType1** - This entry takes the value in terms of percentage with which the tolerance needs to be calculated. We should specify this field for all the data types specified in the previous entry.
- **Minimum tolerance value for DataType1** - This entry allows to configure the minimum tolerance value for the data type specified. The tolerance will never go below this minimum value specified

For example –

```

Mi ni mum tolerance value for DataType1 = 0. 0002
Mi ni mum tolerance value for DataType2 = 0. 002
Mi ni mum tolerance value for DataType3 = 0. 002
... so on

```

- **Default Tolerance Format** - This is the default format for the tolerance values to be displayed in the TPIfile.

**Decimal** - Tolerance will be displayed in Decimal format

**Scientific** - Tolerance will be displayed in Scientific format

- **Tolerance Range For Decimal Output Format** -This entry takes the range of the tolerance beyond which tolerance value appears in scientific format. For ex – we can specify it to be (1.0E-3,11.0E0) which means if tolerance value is less than 0.001 or greater than 11 it appears in the scientific format.

#### [TPI File Header]

- **Variables Counts Starts After = #Inputs** - This entry takes the keyword appearing in the tpi file after which the count for inputs, outputs, test points, and test cases are listed. Currently it is **#Inputs**
- **Data Types and Variables Starts After = all variable data types** - This entry takes the keyword appearing in the tpi file after which the data types of variables are listed. Currently it is **all variable data types**
- **Data Types and Variables Ends After = end data types** - This entry takes the keyword appearing in the tpi file before which data types of variables are listed. Currently it is **all variable data types**
- **Variables Starts After = all variable names** - This entry takes the keyword appearing in the tpi file after which the names of the variables inputs, outputs and test points without the data type are listed. Currently it is **all variable names**
- **Variables Ends After = end variables** - This entry takes the keyword appearing in the tpi file before which the names of the variables inputs, outputs and test points without the data type are listed. Currently it is **end variables**

- **Listing Inputs And Data types Starts After = Inputs:** - This entry takes the keyword appearing in the tpi file after which the inputs are listed along with their data type. Currently it is **Inputs:**
- **Listing Outputs And Data types Starts After = Outputs:** - This entry takes the keyword appearing in the tpi file after which the outputs are listed along with their data type. Currently it is **Outputs:**
- **Listing Test Points And Data types Starts After = Test Points:** - This entry takes the keyword appearing in the tpi file after which the test points are listed along with their data type. Currently it is **Test Points:**
- **Listing Inputs Starts After = Inputs:** - This entry takes the keyword appearing in the tpi file after which the inputs are listed without their data type. Currently it is **Inputs:**
- **Listing Outputs Starts After = Outputs:** - This entry takes the keyword appearing in the tpi file after which the outputs are listed without their data type. Currently it is **Outputs:**
- **Listing TestPoints Starts After = TestPoints:** - This entry takes the keyword appearing in the tpi file after which the test points are listed without their data type. Currently it is **TestPoints:**
- **Input Variable initializations Starts After = variable initializations** - This entry takes the key word appearing in the tpi file after which the variables initialization is done.
- **Input Variable initializations Ends After = end initializations** - This entry takes the key word appearing in the tpi file before which the variables initialization is finished.

#### [Test Case Header]

- **Test Case Number = Test Case** - This entry takes the keyword in the Test case header for the number of the test case in the test case header for the test vector. Currently it is **Test Case**
- **Testing Case Block = "Testing"** - This entry takes the keyword in Test case header for the specifying as which is the Block Under Test for the test vector. Currently it is **"Testing"**
- **Test Case Description = "Description"** - This entry takes the keyword in Test case header for the specifying as which is the Test Case Template for the test vector. Currently it is **"Description"**
- **Test Case Inputs = "Inputs"** - This entry takes the keyword in Test case header for the specifying the inputs which are not don't care in the test vector. Currently it is **"Inputs"**
- **Test Case Outputs = "Outputs"** - This entry takes the keyword in Test case header for the specifying the outputs which are not don't care in the test vector. Currently it is **"Outputs"**
- **Test Case TestReference = "Test Reference"** - This entry takes the keyword in Test case header for the specifying the rationale. Currently it is **"Test Reference"**
- **Test Case Type = "Type"** - This entry takes the key word for test case type after which the test case type appears.
- **Test Case Requirements = "Requirements"** - This entry takes the key word for requirements after which the test case requirements are listed
- **List Output Variable names** - This entry takes two values
  - 1** - If set to "1", names of the output variables appear in the test case header in the output file
  - 0** - If set to "0", names of the output variables do not appear in the test case header in the output file
- **Comment For Test Case Header =\*\*\*\*\*** - This entry takes the keyword in for commenting the test case header. Currently it is \*\*\*\*\*

**[Test Case Values]**

- **Input Variable Values = INPUT VARIABLES for test case** - This entry takes the keyword in the tpi file for specifying the input variables for the specific time step. Currently it is **INPUT VARIABLES for test case**
- **Output Variable Values = OUTPUT VARIABLES for test case** - This entry takes the keyword in the tpi file for specifying the output variables for the specific time step. Currently it is **OUTPUT VARIABLES for test case**
- **Tolerance Values = TOLERANCE VALUES for test case** - This entry takes the keyword in the tpi file for specifying the tolerance for the specific time step. Currently it is **TOLERANCE VARIABLES for test case**
- **Comment For Test Case Values=\*\*\*\*\*** - This entry takes the comment character for the test case values

**[Path And File]**

- **Test Points On** - This takes the path of the folder which contains TPI file generated with test points ON. This will appear in the text box for “Test point On TPI” when we launch the gui.  
By default it is **Test Points On = D:\TPI\_Converter\_Testing\TP\_ON\** - This is the default path to the folder which contains TPI file generated with test points On
- **Test Points Off** - This takes the path of the folder which contains TPI file generated with test points ON. This will appear in the text box for “Test point Off TPI” when we launch the GUI.  
By default it is **Test Points Off = D:\TPI\_Converter\_Testing\TP\_OFF\** - This is the default path to the folder which contains TPI file generated with test points Off
- **CTP Output** - This takes the path of the folder which contains CTP Output file. This will appear in the text box for “CTP output (CSV)” when we launch the GUI.  
By default it is **CTP Output = D:\TPI\_Converter\_Testing\CTP\_Output\** - This is the default path to the folder which contains CTP Output file
- **Output = D:\TPI\_Converter\_Testing\Converter\_Output\** - This is the default path to the folder in which the output file is generated. This will appear in the text box for “Output” when we launch the GUI.  
By default it is **Output = D:\TPI\_Converter\_Testing\Converter\_Output\**

**Note:** A backslash ‘\’ character is required at the end of the path.

## Appendix A. Troubleshooting

### Incorrect or Missing Parameter Elements and Attributes

The following table lists messages you may see in the command window when running a HiLiTE command file. Most of these messages indicate a problem in the construction of the command file. When you see Error or Warn messages in the command window, review Hilite.log; it may contain further explanations. If you observe problems in the generated tests or code, but no errors are displayed in the command window, review the Hilite.log.

**Table 17. Configuration File Errors Reported in the Command Window and HiLiTE.log**

Message/Indicator	Problem/Explanation
[ERROR] *** Validation Error *** : The element 'HiLiTEParameters' cannot contain text. An error occurred at file:///C:/Projects/testthis/testme/testmin.hilite, (1, 20).	This message and the runtime error will occur if you forget to include a <ProjectPath> element in the .hilite file. HiLiTE.log will show the same error.
[ERROR] *** Validation Error *** : The required attribute 'commentChars' is missing. An error occurred at file:///C:/HiLiTE/ models/example.hilite, (9, 5).	Both the command window and the HiLiTE.log will show a similar message if the <ColumnSpec> element is missing one or more attributes. In this case, the element did not include commentChar, delimiters, or TitleLines. If only one attribute is missing, it will be named in the message.

Message/Indicator	Problem/Explanation
ERROR: Could not find ColumnSpec named CombinedRangeDataType ... skipping file C:\AADocuments\HiLiTE\models\Req_Symbols_1-10.csv	The [ERROR] message is displayed in both the command window and the HiLiTE.log if you fail to include column specifications, but did include a <SymbolFile> element.
ERROR:.\example01.mdl could not be processed because it cannot be found... skipping	This message indicates that HiLiTE could not find the model you named in the <Model> element. Make sure the spelling is correct and that the model is in the project path.
ERROR: cycle is trying to run from both ends: an input port already in masterPRSL as an attached Port	This message usually indicates a flaw in the model design, but may show up for other reasons. The error will prevent tests from being generated.
ERROR: Block divide: Divide by zero possible since Max possible range on denominator can not be determined	This message describes a problem in the model. Make sure division blocks don't allow 0 as a denominator. Use a range file and allowmin and allowmax columns to provide range constraints.
WARN: Port simpleDelay1.O1: forward propagated operating range [-192.073170731707,192.073170731707] is larger than estimated (or specified) range [-75,75]. Ignore warning since this is perhaps a counter pattern; keeping the estimated (or specified) range.	If the MATLAB model includes a loop, you can ignore this message. It indicates that range bounds cannot be established at any point of the loop. In fact, they do not need to be accurately established since this is just an intermediate point in the model and the bound here is not a "hard" bound but a "typical" one.
ERROR: Untestable condition input on Block: not1: in port 1; value is always 0 ERROR: Deactivated Block: not1: output is always True	These messages all indicate design errors in which signals are stuck at specific values, giving the model dead logic. You must fix the MATLAB model.
ERROR: Exception in InterpretTemplateToCode(): Exception: System.ComponentModel.Win32Exception Message: The system cannot find the file specified	This message occurs when you try to generate test harness vectors and

Message/Indicator	Problem/Explanation
Source: System at System.Diagnostics.Process.StartWithCreateProcess(ProcessStartInfo startInfo) at System.Diagnostics.Process.Start() at HiLiTE.JUtilities.OSUtilities.exec(String exeName, String args) at HiLiTE.SimulinkCodeGen.CodeGenEngine.GenerateModelCode(IDictionary sFChartSymbols, Boolean generateCode)	perl is not installed.
[ERROR] Exception: HiLiTE.Utilities.ApplicationTimeoutException Message: Model ELEV_PFLDT_CONTROLLER will take more than 7200 seconds to complete	When this happens, HiLiTE aborts to avoid a potential infinite loop. The consequence of this is that you may get partial or no output.  You can lengthen the timeout value by changing the value of the <i>MaxTime</i> Option in the <i>TestGenLimits</i> OptionSet in the .hilite file. Specify the max amount of run time allowed per model in seconds. If you remove or comment out this Option, there will be a default time limit of 3600 seconds
[ERROR] Exception processing current model at HiLiTE.ModelBase.InputPort.GetLink(LinkCharacteristics ltype)  Exception: HiLiTE.ModelBase.PortLinkAssociationException	This ERROR message occurs when there is a space at the beginning of the input port name. just removed the space and HiLiTE will work.
[ERROR] Unable to evaluate block formula...	This ERROR message occurs when unsupported Template Formula Language (TFL) is used for a block. See Section 10.6.

## When Starting a Command file, HiLiTE Installs

If the HiLiTE blocks and templates data (%HILITE\_HOME% \Data\HiLiTEBlocks.xml) is missing or has been renamed, HiLiTE will re-install it when you try to run a command file. When you double-click a .hilite file, an installation progress window appears. Allow the installation to proceed. When the progress bar is complete, the dialog will close and the command window will be displayed. HiLiTE will run the requested command and model.

Reinstallation will occur if you have deleted, renamed, or moved the HiLiTE database. If you carried out any of these actions inadvertently and had made changes to templates, you may want to recover the deleted, modified, or moved database.



## Running Multiple Instances of HiLiTE/Appender Error

If you have not completed a HiLiTE run and attempt to run another instance of HiLiTE, you will get an error message similar to:

```
log4net: ERROR [FileAppender] OpenFile(C:\Program Files\Hi Li TE\bin\Hi Li TE.log, False)
call failed.
System.IO.IOException: The process cannot access the file "C:\Program
Files\Hi Li TE\bin\Hi Li TE.log" because it is being used by another process.
You are most likely to get this error if you minimize the command window after starting HiLiTE and
forget to close it before re-executing.
```

- After running HiLiTE from a file browser (double-clicking a .hilite file), be sure to close the command window from which it ran.
- If you start HiLiTE by executing the HiLiTE command in a command window (e.g. hilite mymodel.xml), be sure an empty command prompt (e.g. c:\hilitemodels\>) is the last displayed line in the window before executing the command again in another window.

## HiLiTE.log is Missing

If, after you run a .hilite file you cannot find the file hilite.log in your %HILITE\_HOME%\bin directory, the file hilite.exe.config probably requires editing. To reconfigure HiLiTE so it creates a log file:

Open .\ %HILITE\_HOME%\bin\hilite.exe.config in a text editor.

Find the line (approximately line 54):

```
<!-- Setup the root category, add the appenders and set the default priority -->
The following section is a <root> element which should look something like this:
```

```
<root>
  <priority value="WARN" />
  <appender-ref ref="LogFileAppender" />
  <!--<appender-ref ref="ConsoleAppender" />-->
</root>
```

This sub-element/attribute combination <appender-ref ref="LogFileAppender" /> creates the hilite.log file.

- If it is missing from your .config file, type it in after the <priority> element.
- If it is commented out (the line begins <!--and ends -->), remove the comment symbols.
- If the line is present and not commented out, contact HiLiTE technical support.

Note: the element <appender-ref ref="ConsoleAppender" /> configures HiLiTE to display all logging information in the command window. It is usually commented out, but you can remove the comment marks to see the information as you run HiLiTE.

## Duplicate State Machine and Model Names

Don't name the state machine with the same name as the model file. When you run HiLiTE, it creates a model-level file that uses the model file name. Each state machine results in a file using the name of the state machine.

## HiLiTE cannot start MATLAB

If HiLiTE consistently shows that it cannot find a MATLAB license, especially when you are able to start MATLAB outside of HiLiTE, the MATLABEngine.dll may not have registered. Try registering %HILITE\_HOME%\Build\bin\MATLABEngine.dll

1. Right click on the file name and select **Open with...** from the context menu. Windows displays a message warning you that this is a file used by the operating system.
2. Click the **Open With** button. The next dialog says that Windows can't open the file.
3. Click the option button for **"Select the program from a list,"** then click **OK**.
4. In the next dialog, the first item should be Microsoft Register Server. Select this and click OK. If the registration works, Windows will display a confirmation message.  
If you get an error message, call HiLiTE technical support.

If the dll is registered, it's possible that more users are accessing MATLAB than your facility has licenses for. You can add a parameter to your command file to automatically retry MATLAB access. See the `maxRetries` attribute of the `<HiLiTEMgmt>` element in Appendix H.

## Can't Access MATLABEngine.dll

While the .msi installer will properly install the matlabengine.dll for most users, we have reports that this doesn't happen completely on some machines.. The matlabengine.dll is an old style COM library, so it needs to be registered with the system.

MATLABEngine.dll is located in the HiLiTE install directory. To manually register MATLABEngine.dll with the system, run regsvr32 on it. One way to do this is to attempt to open the DLL. This will fail, but when it does, you can select a program (\Windows\System32\regsvr32) to associate with the .DLL extension. This will then apply regsvr32 to the DLL, registering it. You can also run regsvr32 from the command line, giving the name of the DLL as a parameter.

If regsvr32 fails for some reason, consult the Microsoft MSDN website; start here:

[Explanation of Regsvr32 Usage and Error Messages](http://support.microsoft.com/XSLTH3128121123120121120120)  
(<http://support.microsoft.com/XSLTH3128121123120121120120> )

The following information is taken from that page:

### Regsvr32.exe and Dependencies

RegSvr32.exe depends on the Kernel32.dll, User32.dll, and Ole32.dll files (and the Msvcrt.dll and Advapi32.dll files in Windows NT). Regsvr32.exe loads the file you are trying to register or un-register, along with all of its dependencies. The process may be unsuccessful if a required file is missing or damaged.

You can use Depends.exe to determine dependencies for the file you are trying to register or un-register. Depends.exe is included with the *Microsoft Windows 98 Resource Kit* and the *Microsoft Windows NT 4.0 Resource Kit* support tools.

### If dependencies are missing, you may not have installed all the HiLiTE prerequisites.

See Section 1.2 Installation and Processing Requirements at the beginning of this guide for a complete list. In particular, you may be missing the Microsoft .NET redistributable, the Visual J# redistributable, or MATLAB. The Microsoft .NET redistributable should come installed as part of Windows XP, but it is not part of the standard Windows 2000 install.

## Initialization Errors

**ERROR - Initialization failed - no query processed**

Problems with MATLAB license may occur if you are using a shared license on a network that allows a limited number of users at one time. If you have an individual license on the machine where you work, this problem should not occur.

If HiLiTE cannot obtain a MATLAB license when it tries to read a model, it will quit without generating output. Note that you may get a MATLAB license and then fail to get a Simulink license (they are separate). This will result in initialization errors, and again HiLiTE will stop without generating anything useful.

While we have not received reports of failure to obtain a Stateflow license, this may also result in errors.

## Problems after installing a new version of MATLAB

If you have installed a new version of MATLAB and retained the previous version, you may get errors related to unregistered or wrongly registered dynamically link libraries (dll).

1. One possible solution is to register the newest version of MATLAB using a MATLAB command. For example, to register MATLAB 7.1, r14,
2. Start the correct version of MATLAB (probably the shortcut on your desk top).
3. When you see a command prompt (>>), type (or copy and paste) this command:

```
cd([matlabroot ' /bin/win32' ])
!matlab /regserver
```

MATLAB may generate a warning to the effect that “simulink.dll has the same name as a MATLAB builtin. We suggest you rename the function to avoid a potential name conflict.” You can ignore this.

Try re-running HiLiTE.

If you still get registration or dll errors, call HiLiTE support.

## Missing Libraries

This error occurs when HiLiTE cannot find the block libraries for a MATLAB model. If you try to generate test procedures from models you received from elsewhere or that you moved to a new machine, you may not have acquired libraries referenced by those models or may have placed them in the wrong directory. For information about storing libraries, refer to the MATLAB documentation or contact the person who supplied the models.

## Garbled output or MATLAB crashes

In running a model, it is possible for race conditions to develop between a clock and the data inputs associated with the clock. When running HiLiTE, such race conditions may show up either as garbled output or when MATLAB crashes. If you suspect that race conditions are causing current problems, you can enforce a delay using the HiLiTEgmt element with a raceDelayMS attribute. See Appendix H.

## FAQ

### I'm not getting the output I expect. What went wrong?

If you aren't getting any output, or are getting a lot less output than you expect, the most likely cause is that the models that aren't generating output are timing out before they finish running. If you look at the log file or the console window when these run you will see a message like the following:

```
[ERROR] Exception: HiLiTE.Utilities.ApplicationTimeoutException
Message: Model ELEV_PFLDT_CONTROLLER will take more than 7200 seconds to complete
```

When this happens, HiLiTE aborts to avoid a potential infinite loop. The consequence of this is that you may get partial or no output.

You can lengthen the timeout value by changing the value of the MaxTime Option in the TestGenLimits OptionSet in the .hilite file. Specify the max amount of run time allowed per model in seconds. If you

remove or comment out this Option, there will be no timeout value - HiLiTE will continue to run to completion.

**I'm now working with the latest HiLiTE version and was looking for the support for the models that make heavy use of busses. There is a script referred to (Buscreators.m) that does not appear to be a part of the load I installed. Where is it?**

All scripts are in the %HILITE\_HOME%\Scripts directory.

**Why didn't HiLiTE generate a TPI file when I expected one?**

If you specify whether test points are on or off (see information about option sets in Section 4, Table 2), it will have an effect on the CTP output. By default, the test points are on. If you leave this default, but the declarations file has no information about test points, then HiLiTE will not generate any tpi file. A special case occurs if test points have siblings that are outputs; these test points will be ignored. If *all* test points in the model are for outputs, then HiLiTE will ignore the declarations file and generate the tpi file.



## Appendix B. Creating Range and Data Type Files

HiLiTE imports range information from production models, so in many cases you do not need to create or specify separate range files. Range files, though are useful for experimentation, as you can easily change range values in the files without changing the model. This appendix includes additional information about modifying data that HiLiTE uses to generate tests and to perform model analysis. The items include:

- Instructions for creating range files (called by the *<SymbolFile>* element)
- Detailed explanation of universal data types and language independent representations used for defaults and in system dictionary translations
- Instructions for creating header files

### Data Types

HiLiTE is language neutral; universal data types (UDTs) and language independent variables (LIR) provide implementation-independent concepts that describe the most fundamental characteristics of data.

If you do not include a reference to symbol file in the command file, or if certain symbols are missing from the referenced files, you can instruct HiLiTE to use default values. Default ranges are established for UDTs and LIRs in the DefaultRange element described in the tables in Section 4.

**UniversalInteger**—Positive or negative whole number of unlimited size.

**UniversalReal**—Positive or negative whole or decimal number of unlimited size. This data type will support double datatype alone or both double and integer types.

**UniversalBoolean**—True or False

**Language independent representations** (LIRs) define the characteristics of a variable in a way that takes machine characteristics into account without focusing on how a language specifies the type. For HiLiTE, LIRs specify data kind and size. You can use them in symbol files to specify variable type. In a command file, use them in the Translation variable of a *<Column>* element to translate information to kind and size to effect range and accuracy (for example, translating a “flag” to Bool or a number type to one that includes size limitation, such as Int64).

The currently defined LIRs are:

Int64 Int32 Int16 Int8	All of these LIRs define integers. The number indicates maximum size of the integer in number of bits. For example, Int32 can be any integer between -2 billion and +2 billion.
UInt64 UInt32 UInt16 UInt8	All of these LIRs define positive (unsigned) integers. The number indicates maximum size of the integer in number of bits. For example, UInt32 can be any integer up to 4 billion.
Float64 Float32	Floating point numbers. Any real number up to the maximum size.
Bool	1 (true) or 0 (false)

**Note:** The “LangIndep\_” prefix may be included in data type names (see Figure 114), but the prefix is not necessary (e.g. LangIndep\_Int64 is equivalent to Int64 alone).

## Creating Range and Data Type Files

If a model does not include range information or if you want to test ranges different from those given in a model, specify the information in a range file. Range information is imported into HiLiTE from an ASCII range file based on descriptions given in *<ColumnSpec>* and *<SymbolFile>* elements in the command file. *ColumnSpec* elements describe the format of the file containing the range/data type information. *<SymbolFile>* elements indicate which files to read and which *<ColumnSpec>* to use when reading them.

If you are generating tests for multiple models, you can include all the range information for all models in one file or you can specify multiple range files. If a symbol is defined more than once (either in the same file or across different files), HiLiTE will use the last definition that it reads.

If HiLiTE cannot determine operating ranges, HiLiTE will generate a MissingSymbols.csv file automatically for symbols whose operating ranges cannot be determined. This file is in a standard Range file format and can simply be renamed, filled in, and referenced by the .hilite file.

Note: Not all symbols need defined ranges; for example, Boolean data is always 0 or 1, so Boolean symbols do not need to be included in range files. All model inputs and certain outputs (in feedback loops) need this information specified. If HiLiTE needs range/data type information for a symbol that has not been specified, the run log (HiLiTE.log) will show a warning or error. These problems should always be corrected, as lack of this information can lead to failure to generate all needed test cases.

A simple way to make symbol files is to create them first in a spreadsheet program such as Excel, then save the data to a .csv file.

Range and data type files:

- Must be in ASCII format
- May contain column titles and comment lines. All other lines must contain a symbol name in the column indicated by the ColumnSpec used to read that file.
- Include the symbol name on each line (other than column titles and comments). Rows may also contain the min and max values for the symbols, as well as the symbol’s fundamental data type and language independent data representation.
- Can include more columns than you require for generating test procedures. Use the ColumnSpec and Column elements to specify which columns HiLiTE will use from the command file. (For information about these elements, refer to Section 4,

Creating HiLiTE Command Files).

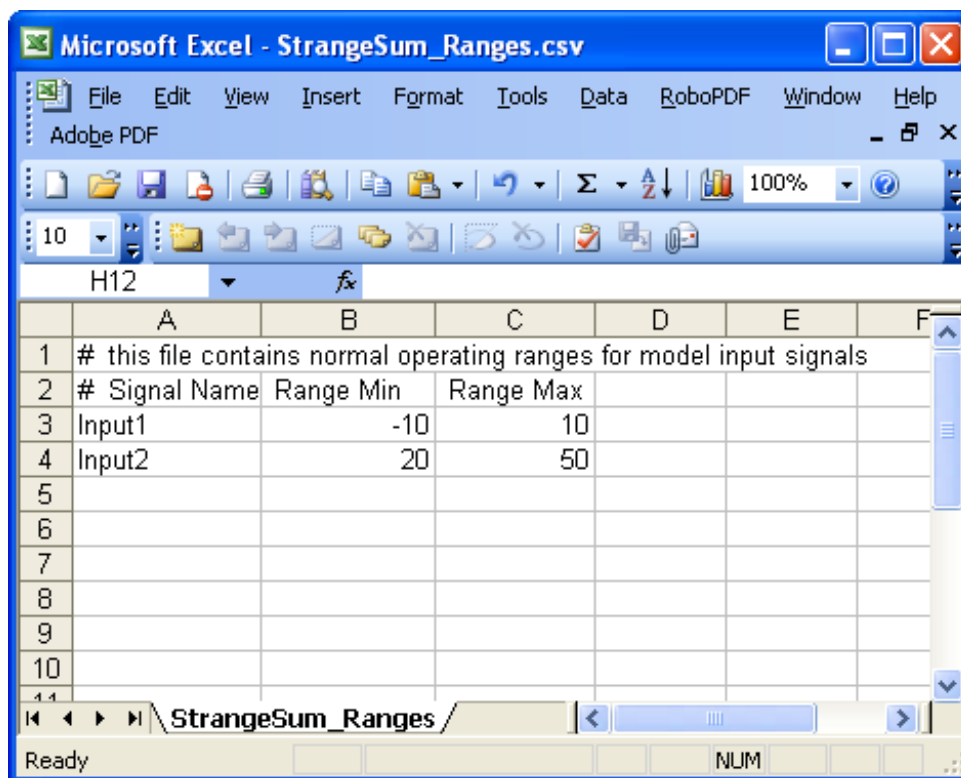
- Have columns delimited by commas, spaces, or tabs. For all delimiters except blanks, each delimiter denotes the boundary between two columns. This means that multiple consecutive delimiters indicate empty cells. Multiple consecutive blanks, however, are treated as a single delimiter.
  - If you want the ASCII file display with fixed column widths, use spaces to separate columns rather than tabs. HiLiTE interprets any number of spaces as a single column delimiter. If you separate columns with tabs, HiLiTE will interpret each tab as a separate column delimiter.
  - Note: the delimiter must be specified in the associated ColumnSpec.
- Can include comments that HiLiTE will not process. HiLiTE will ignore all information in a row that begins with a comment character. Select comment delimiters wisely; for example, # is often used for a comment delimiter but it can also be a symbol for a unit of measure.

The left most column in a symbol file is column 0.

Figure 113 illustrates a very basic range file which is described in the HiLiTE command file with the following column specification:

```
<!-- describe range file format -->
  <ColumnSpec specName="OpRangeOnly" delimiters="," commentChars="#"
    titleLines="1">
    <Column name="symbol" col="0"/>
    <Column name="opmin" col="1"/>
    <Column name="opmax" col="2"/>
  </ColumnSpec>
<!-- specify file that contains normal operating ranges for this
      model's inputs -->
<SymbolFile fName="StrangeSum_Ranges.csv" specName="OpRangeOnly" />
```





Microsoft Excel - StrangeSum\_Ranges.csv

File Edit View Insert Format Tools Data RoboPDF Window Help

Adobe PDF

100%

H12

	A	B	C	D	E	F
1	# this file contains normal operating ranges for model input signals					
2	# Signal Name	Range Min	Range Max			
3	Input1	-10	10			
4	Input2	20	50			
5						
6						
7						
8						
9						
10						

StrangeSum\_Ranges

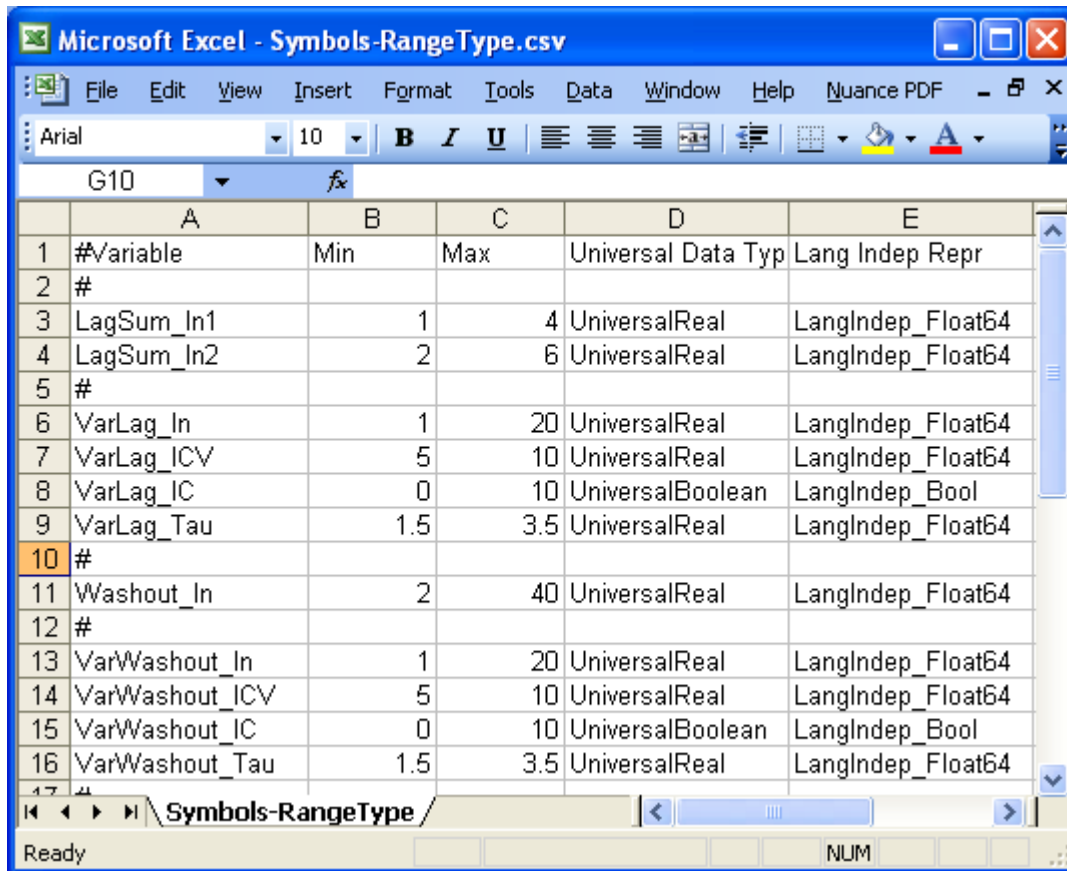
Ready NUM

**Figure 113 - Basic Range File Containing Only Normal Operating Ranges**

Figure 114 is an example of a file formatted for the following, more complex, column specification:

```
<ColumnSpec specName='CombinedRangeDataType' delimiters=',',
             commentChars='#' titleLines='1'>
  <Column name='symbol'    col='0' />
  <Column name='min'       col='1' />
  <Column name='max'       col='3' />
  <Column name='utype'     col='4' />
  <Column name='lir'       col='5' />
</ColumnSpec>

<SymbolFile fName="Symbols-RangeType.csv"
             specName='CombinedRangeDataType' />
```



Microsoft Excel - Symbols-RangeType.csv

	A	B	C	D	E
1	#Variable	Min	Max	Universal Data Typ	Lang Indep Repr
2	#				
3	LagSum_In1	1	4	UniversalReal	LangIndep_Float64
4	LagSum_In2	2	6	UniversalReal	LangIndep_Float64
5	#				
6	VarLag_In	1	20	UniversalReal	LangIndep_Float64
7	VarLag_ICV	5	10	UniversalReal	LangIndep_Float64
8	VarLag_IC	0	10	UniversalBoolean	LangIndep_Bool
9	VarLag_Tau	1.5	3.5	UniversalReal	LangIndep_Float64
10	#				
11	Washout_In	2	40	UniversalReal	LangIndep_Float64
12	#				
13	VarWashout_In	1	20	UniversalReal	LangIndep_Float64
14	VarWashout_ICV	5	10	UniversalReal	LangIndep_Float64
15	VarWashout_IC	0	10	UniversalBoolean	LangIndep_Bool
16	VarWashout_Tau	1.5	3.5	UniversalReal	LangIndep_Float64
17	#				

Ready NUM

Figure 114 - Example File Containing Range and Data Type Information

## Naming Variables in Symbol Files

Variable names can represent model inputs, outputs, and intermediate signals inside the models. Additionally, a variable can be a vector comprised of an array of elements. The symbol file variable naming specification allows you to name all of these variable types.

**Model inputs and outputs.** These variable names consist of a single identifier (see Figure 113 and Figure 114)

**Intermediate signal in a model.** Intermediate signals are typically used to specify data types or ranges for variables within feedback loops. The variable name uniquely identifies a signal within a model as an output of a block using this format:

`<model - name>/<block-hierarchical-name-within-model>.outputPortName`

**Vector variables.** A model input, output, or intermediate signal can be a vector; the range for each vector element can be specified independently in a range file. The syntax is a C-style array notation for symbol name using a zero-based index as shown in this example:

	A	B	C	D
1	# operating ranges of vectorized signals			
2	# symbol	op min	op max	
3	Sum_Vector_In1[0]	0	1	
4	Sum_Vector_In1[1]	10	11	
5	Sum_Vector_In1[2]	20	21	
6	Sum_Vector_In2[0]	100	101	
7	Sum_Vector_In2[1]	200	201	
8	Sum_Vector_In2[2]	300	301	
9	# a scalar range			
10	Counter_Scalar_In1	11	21	
11	# feedback loop vector range			
12	myModel/simpleDelayVec.Output[0]	100	200	
13	myModel/simpleDelayVec.Output[1]	200	300	
14	myModel/simpleDelayVec.Output[2]	300	400	
15				

Figure 115. Examples of Vector Ranges

### Notes on vector range format.

1. In the symbol file, the vector symbol names must appear in strict ascending numerical order of the vector index; otherwise error messages will be raised by HiLiTE. Note that sorting in alphabetical ascending order in Excel may not achieve this desired numerical order.
2. The number of vector indices specified in the symbol file must be exactly equal to the dimension of the variable in the model.

- The vector index is zero-based.
- A scalar symbol format can alternatively be specified for a vector variable (i.e., without the '[x]' notation). In this case, the same range is applied to all vector elements.

### Including Unique Identifiers in the Range File

HiLiTE extracts unique-identifier information (if present) from a symbol file and reports it in the Status Report file. The unique identifier is optional and can be in any line of the symbol (.csv) file, provided special formatting is observed. Two formats are supported:

**Format 1:** Contents of the line must be in the following format: [SymbolFile\_GUID\_Fmt1\_15]

#<any-text>*UniqueIdentifier*, <unique-identifier-string>, <any-text>  
(where *UniqueIdentifier* is a keyword and '#' ',' are delimiters)

	A	B	C	D	E	F
1	# NOTE: the purpose of this file is to test the whether HiLiTE reports the unique ID correctly					
2	# the symbol names used may not apply to any model					
3	# operating ranges of model inputs					
4	# symbol	op min	op max			
5	symbolname1	11	21			
6	# UniqueIdentifier	UniqueID_011	this line has unique id			
7	symbolname2	11	21			
8						

**Format 2:** Contents of the line must be in the following format: [SymbolFile\_GUID\_Fmt2\_16]

<any-text>{<guid-string>}<any-text>, **GUID**, <any-text>

	A	B	C	D	E	F
1	# NOTE: the purpose of this file is to test the whether HiLiTE reports the GUID correctly					
2	# the symbol names used may not apply to any model					
3	#Name	Data Type	MinRange	MaxRange	ProducerModel	
4	inputname1	float32	10	20	unknown	
5	inputname2	float32	0	10	unknown	
6	#the following line has GUID					
7	{F51DFC3B-21C5-4423-ABC8-C20D61FDDA3F}	GUID	0	0	na	

Notes on formats:

Keywords: **UniqueIdentifier**, **GUID**

Delimiters: '#', '{', '}', ','

delimiters cannot be used within <unique-identifier-string> and <guid-string>



## Appendix C. Reporting Problems and Enhancement Requests

If you encounter problems that are not covered by the current HiLiTE Release Notes or elsewhere in this User Guide, report them to the HiLiTE management using the JIRA tracking system. You can also use this system to suggest upgrades and enhancements for HiLiTE. Using this system gives you a forum for influencing HiLiTE development, ensures that HiLiTE developers can refer to permanent records of your concerns, and lets you track progress in addressing those concerns.

The JIRA system is administered at the Honeywell Aerospace level and can be accessed at the following URL:

<https://jira.honeywell.com/jira/secure/Dashboard.jspa>

After entering the JIRA system, please access the project: *Aero-SysSW-HILITE-Support*.

This page intentionally left blank.

## Appendix D. Test Generation Objectives

The goal of testing is, of course, to make sure the thing tested works as intended. The key, therefore, is determining what was intended. While modeling in general is all about focusing on intent, many features of modeling tools such as Simulink and Stateflow, aren't usable unless the model includes a lot of detail. At this point, intent and implementation get intermingled.

HiLiTE is a requirements-template based testing tool. It is designed to verify that object code complies with requirements of the model. Current testing tools generate tests at the low-level design details of the model, which can provide good structural code coverage but are inadequate for testing the data and control coupling incident upon each block. If subtle errors occur in the lowest-level details of the model, a low-level based test will not detect it, but a requirements template-based test will.

HiLiTE may not generate all needed tests in all situations. It will do its best, but it is always possible that the model will contain block combinations that are inherently un-testable or too convoluted for HiLiTE to handle. While HiLiTE will generate a high percentage of the tests requested, it may not generate some. HiLiTE generates a status report that lists the tests it didn't generate (and why—if it knows) so you can determine if those tests can be created manually.



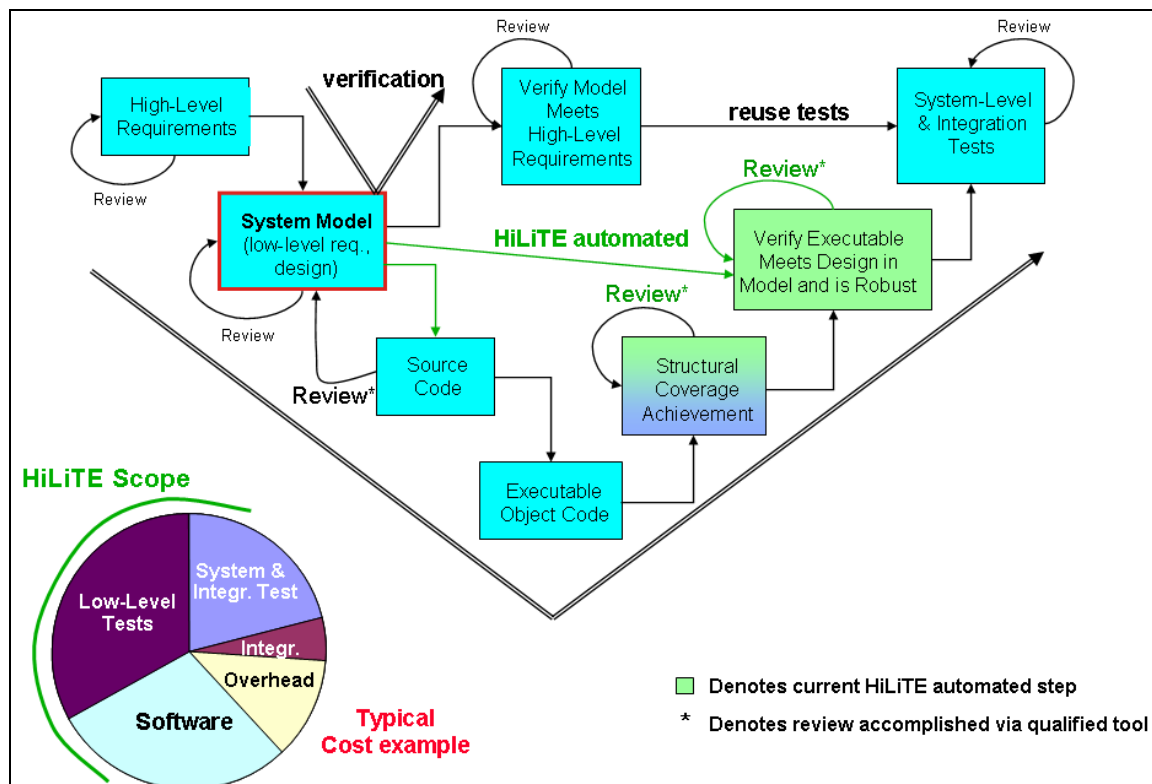


Figure 116 - Honeywell Aerospace MBD Process for DO-178B using HAM & HiLiTE Tools

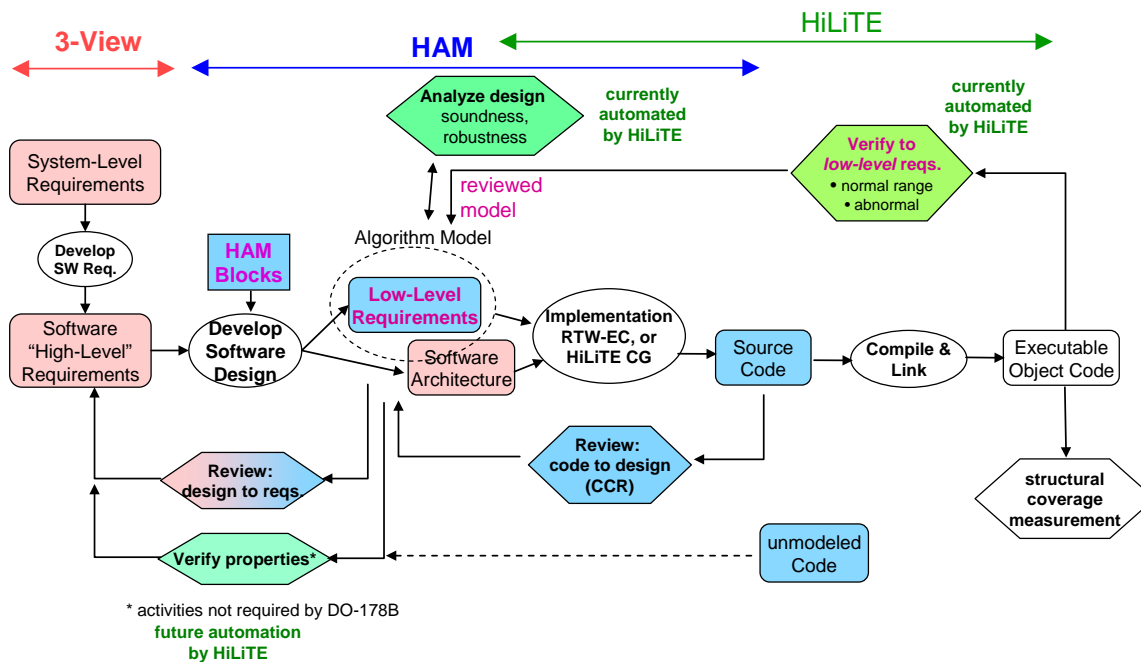


Figure 117 - DO-178B Based Process

## Automated Model-Based Test Generation

Model-based test generation is a top-down approach that relies on test case templates at the library block level. The templates ensure that tests adhere to the “requirements-contract” of the library block such that all the data control couplings of the block instance in the object code can be tested according to this contract. It avoids using low-level design details of library-blocks to be sure that detailed design/implementation errors can be detected.

This type of testing can automatically detect patterns of blocks in specific combinations and feedback loops and apply a template to the entire pattern. Pattern detection makes it possible to test the intent embedded in the design: counters, memory switches, timer feedbacks.

Test case templates can be defined at any subsystem level. The designer supplies a template (with multiple steps), identifying the subsystem by its hierarchy name or mask. The testing tool then figures out the time sequence of model inputs and expected outputs to simulate the embedded subsystem with its template.

## HiLiTE Test Generation

HiLiTE test generation applies requirements-based test case templates at the library block level and to block combination patterns such as feedback loops, strobes, counters, etc. The test generation has three goals.

- HiLiTE verifies that object-code complies with requirements in the model normal-range test cases and all functional operation. To do this, HiLiTE:
  - verifies that each library block instance behaves as it is intended. Testing the block-library in MATLAB and SCV help part of this objective. HiLiTE test generation helps further by testing each block instance with its specific parameters and in its embedding context (e.g. of feedback loops). It tests each time-dependent block in context through multiple steps: e.g. test a filter’s response to  $5 * \text{Tau}$ ; cycle through a timer.
  - verifies data/control coupling among the blocks and block combinations. It tests correct operation of arithmetic and Boolean expressions and control/data coupling through local and global variables.
- HiLiTE verifies that object-code is “robust” with requirements in the model abnormal-range (robustness) test cases, anomalies (e.g. stuck at values, dead blocks, divide by zero, logic/math conflicts). It verifies robustness through block-level test templates are based upon both operating and maximum possible ranges and tests at boundaries and for max possible ranges; force overflow; and absence of divide-by-zero.
- HiLiTE achieves structural coverage of the generated code (including mc/dc) by applying the library-block level requirements-based templates on each instance of an embedded block.

**DO-178B Objectives Satisfied using HiLiTE**

Using HiLiTE helps you meet DO-178 certification requirements:

- **A-6.3** Verify low-level requirements are fully satisfied by the executable object code
- **A-6.4** Verify executable object code is robust with low-level requirements.
- **Levels A & B:** independence of test generation is required from the implementation path of low-level requirements.
- **6.4.2** “Functionally test each time-dependent block in context thru multiple frames” e.g. test a filter’s response to  $5 * \text{Tau}$ ; cycle through a time.

## Appendix E. HiLiTE Support for Block Libraries and MATLAB Versions

**Note:** Please refer to Release Notes of a particular HiLiTE release to see open issues and limitations. Please refer to a specific Tool Qualification Accomplishment summary for information on which version is qualified and its open issues and limitations.

### HAM and MATLAB Versions Supported

HiLiTE currently supports following versions of HAM:

- HAM 5 Release 6, MATLAB Simulink/Stateflow Release R14SP3
- HAM 6 Release 1, MATLAB Simulink/Stateflow Release R2007a
- HAM 5 Release 7, MATLAB Simulink/Stateflow Release R14SP3
- HAM 5 Release 8, MATLAB Simulink/Stateflow Release R14SP3

### Block Libraries Supported

Please refer to Section 4.2 for details of the command options.

HiLiTE supports HAM, CCC, and Boeing block libraries that are mapped as CCCMapping.xml, HAMMapping.xml, and BoeingMapping.xml in the %HiLiTE\_HOME%\Block Libraries. Use the tables below to find the names of corresponding HiLiTE block types as given in the Template Manager.

**HAM:** The HAM block library of the HAM version being used is supported. Include the following specification in .hilite command files:

`<Extension>HAM</Extension>`

The following list of HAM blocks are supported by HiLiTE.

**Table 18. HAM Library Blocks Supported by HiLiTE**

<b>HAM Block Name</b>	<b>HAM Block Mask Type</b>	<b>HiLiTE Block Type Name</b>
1-D Look-Up Table	Look-Up Table (HW)	Table1D
Absolute Value	Absolute Value (HW)	Abs
AND	And (HW)	And
Angle Summation	Angle Sum (HW)	SumAngle
ArcCosine	ArcCosine (HW)	TrigonometricFunction
ArcSine	ArcSine (HW)	TrigonometricFunction
ArcTangent	ArcTangent (HW)	TrigonometricFunction
ArcTangent 2	ArcTangent2 (HW)	ArcTan2
ARINC429 Rx Decode	ARINC429 Rx Decode (HW)	ARINC429RxDecode
ARINC429 Tx Encode	ARINC429 Tx Encode (HW)	ARINC429TxEncode
Asymmetrical Limiter	Asymmetric Limit (HW)	RangeLimitFixed
Asymmetrical Limiter with Flag	Limiter Hold (HW)	RangeLimitFixedWithFlag
Average	Average (HW)	Average
Bit Shift	Bit Shift (HW)	BitShift
Bit Unpack	Bit Unpack (HW)	BitOr
Bitwise AND	Bitwise And (HW)	BitAnd1
Bitwise AND 2	Bitwise And 2 (HW)	BitAndMulti
Bus Creator	BusCreator (HW)	Mux
Bus Selector	BusSelector (HW)	BusSelector
Ceiling	Ceiling (HW)	Rounding
Confirm OFF	Confirm OFF (HW)	TimerFixed
Confirm ON	Confirm ON (HW)	TimerFixed
Constant	Constant (HW)	Constant
Convert Float-to-U32	Conv Float to U32 (HW)	U32ToFloat
Convert U32-to-Float	Conv U32 to Float (HW)	U32ToFloat
Cosine	Cosine (HW)	TrigonometricFunction
Data Comparison	Data Compare (HW)	RelationalDataCompare
Dead Band	Deadband (HW)	Deadband
Dead Zone	Deadzone (HW)	Deadzone
Debounce	Debounce (HW)	DebounceFixed
Decoder	Decoder (HW)	Decoder
Demultiplexer	Demux (HW)	Demux
Divide	Divide (HW)	Divide
Enable	Enable (HW)	Enable
Encoder	Encoder (HW)	Encoder
Equal To	Equal To (HW)	RelationalOperator
Exponential	Exponential (HW)	MathFunctionUnary

HAM Block Name	HAM Block Mask Type	HiLiTE Block Type Name
Fader	Fader (HW)	Fader
Falling Edge One-Shot	Falling Edge One-Shot (HW)	OneShotFallingHoldFixed
Fix	Fix (HW)	Rounding
Floor	Floor (HW)	Rounding
Frame Delay	Frame Delay (HW)	UnitDelayReset
From	From (HW)	From
Gain	Gain (HW)	Gain
Goto	Goto (HW)	Goto
Greater Than	Greater Than (HW)	RelationalOperator
Greater Than or Equal To	Greater Than or Equal (HW)	RelationalOperator
Ground	Ground (HW)	Constant
Initial Condition Trigger	Initial Condition Trigger (HW)	InitialConditionTrigger
Inport	Inport (HW)	Inport
Lag Filter	Lag Filter (HW)	LagFilterFixedTau
Lead-Lag Filter	Lead Lag Filter (HW)	LeadLagFilterFixedTau
Less Than	Less Than (HW)	RelationalOperator
Less Than or Equal To	Less Than or Equal (HW)	RelationalOperator
Linked Model Revision	Linked Model Revision (HW)	DocBlock
Look-Up Table (2-D)	Look-Up Table 2-D (HW)	Table2D
Look-Up Table (3-D)	Look-Up Table 3-D (HW)	Table3D
Look-Up Table N-D	Look-Up Table N-D (HW)	NdTable2 NdTable3
Look-Up Table Pre-Index	Look-Up Table PreIndex (HW)	PreIndexTable
Matrix Selector	Matrix Selector (HW)	MatrixSelector
Maximum Value	Maximum (HW)	Max
Minimum Value	Minimum (HW)	Min
Modulus	Modulus (HW)	Modulus
Multiplexer	Mux (HW)	Mux
Multiport Switch	Multiport Switch (HW)	MultiPortSwitch
NAND	Nand (HW)	NAND
Non-Multiplexed Selector	Non-Mux Selector (HW)	PortSelector
NOR	Nor (HW)	NOR
NOT	Not (HW)	NOT
Not Equal To	Not Equal To (HW)	RelationalOperator
OR	Or (HW)	OR
Output	Output (HW)	Output
Power	Power (HW)	Power
Product	Product (HW)	Product
Quadratic Filter	Quadratic Filter (HW)	QuadFilter
Register	Register (HW)	Register

HAM Block Name	HAM Block Mask Type	HiLiTE Block Type Name
Relay	Relay (HW)	HysterisisFixed
Reset Integrator	Reset Integrator (HW)	IntegratorReset
Reset Lead-Lag Filter	Reset Lead Lag Filter (HW)	LeadLagFilterFixedTauResetVal
Reset Variable Asymmetrically Limited Integrator	Limited Reset Integrator (HW)	IntegratorResetLimited
Reset-Hold Variable Asymmetrically Limited Integrator	Limited Reset Hold Integrator (HW)	IntegratorResetLimitedHold
Revision	Revision (HW)	DocBlock
Rising Edge One-Shot	Rising Edge One-Shot (HW)	OneShotRisingHoldFixed
Round	Round (HW)	Rounding
Sample and Hold	Sample And Hold (HW)	SampleAndHold
Sample Time	Sample Time (HW)	Constant
Scheduler	Scheduler (HW)	RangeLimitAdjScaled
Selector	Mux Selector (HW)	PortSelector
Set-Reset Flip-Flop	SR Flip Flop (HW)	SRFlipFlop
Set-Reset Latch	Set-Reset Latch (HW)	LatchResetDom
Sign	Sign (HW)	Signum
Sign Comparison	Sign Comparison (HW)	RelationalSignCompare
Signal Conversion	Signal Conversion (HW)	SignalConvert
Simple Delay	Simple Delay (HW)	UnitDelay
Simple Integrator	Simple Integrator (HW)	IntegratorSimple
Sine	Sine (HW)	TrigonometricFunction
Sliding Window Debounce	Sliding Debounce (HW)	DebounceSliding
Square Root	Square Root (HW)	SquareRoot
StimulateBoolean	Stimulate Boolean (HW)	StimulateBoolean
StimulateNumeric	Stimulate Numeric (HW)	StimulateNumeric
Subsystem	Subsystem (HW)	GenericSubsystem
Subsystem Revision	Subsystem Revision (HW)	DocBlock
Summation	Sum (HW)	Sum
Switch	Switch (HW)	Switch1S0
Switched Fader	Switched Fader (HW)	FaderSwitched
Symmetrical Rate Limiter	Symmetrical Rate Limiter (HW)	RateLimitFixed
Tangent	Tangent (HW)	TrigonometricFunction
Tangent (Hyperbolic)	Tangent Hyperbolic (HW)	TrigonometricFunction
Terminator	Terminator (HW)	Terminator
Test Point	Test Point (HW)	TestPoint
Transpose	Transpose (HW)	Transpose

HAM Block Name	HAM Block Mask Type	HiLiTE Block Type Name
Typecast	Typecast (HW)	DataTypeConversion
Variable Asymmetrical Limiter	Variable Asymmetrical Limit (HW)	RangeLimitAdjVar
Variable Asymmetrical Rate Limiter	Variable Rate Limiter (HW)	RateLimitAdjResetResetVal
Variable Confirm	Variable Confirm (HW)	ConfirmVal
Variable Lead-Lag Filter	Variable Lead Lag Filter (HW)	LeadLagFilterVarTau
Variable Limiter Flag	Variable Limiter Flag (HW)	RangeLimitVarWithFlag
Variable Reset Debounce	Variable Reset Debounce (HW)	DebounceVariableReset
Variable Reset Lag Filter	Variable Reset Lag Filter (HW)	LagFilterVarTauResetVal
Variable Reset Washout Filter	Variable Reset Washout Filter (HW)	WashoutFilterVarTauResetVal
vectorAssignment	Vector Assignment (HW)	VectorAssignment
Vector Selector	Vector Selector (HW)	VectorSelector
Washout Filter	Washout Filter (HW)	WashoutFilterFixedTau
XOR	Xor (HW)	XOR

**CCC:** The CCC Block Library is supported. Include the following specification in .hilite command files, which automatically implies the use of HAM block library:

<Extension>CCC/Extension>

HiLiTE supports the following CCC blocks without limitations.

**Table 19. CCC Library Blocks Supported by HiLiTE**

CCC Block Name	Mask Type	HiLiTE Block Type
Comparator with Hysteresis Block	CCC Comparator Hysteresis (HW)	HysteresisComparator
Comparator with Hysteresis SF Block	CCC SFcn Comparator Hysteresis (HW)	HysteresisComparator
Confirm ON SF Block	CCC SFcn Confirm ON (HW)	TimerFixed_CCC
Differentiator (Rectangular Implementation) Block	CCC Differentiator Rectangular (HW)	DifferentiatorResetLimited
Lag Filter Block	CCC Lag Rect (HW)	LagFilterVarTauP20
Lead-Lag Filter Block	CCC Lead Lag Rect (HW)	LeadLagFilterVarTauRectCCC
One Shot Block	CCC OneShot (HW)	OneShotHoldAdj
Range Limit Block	CCC Range Limit (HW)	RangeLimitAdjVar
Rate Limit Block	CCC Rate Limit (HW)	RateLimitAdjResetResetValEx
Selector Switch	CCC Selector Switch (HW)	MultiPortSwitchSelectorSwitch
Simple Differentiator (Rectangular Implementation) Block	CCC Simple Differentiator Rect (HW)	Differentiator
Simple Switch Block	CCC Simple Switch (HW)	Switch1S0



CCC Block Name	Mask Type	HiLiTE Block Type
Track Hold Block	CCC Track Hold (HW)	SampleAndHoldP20
Track Hold SF Block	CCC SFcn Track Hold (HW)	SampleAndHoldP20
Up-Down Rate Timer Block	CCC Up Down Rate Timer (HW)	BidirectionalVariableConfirm
Up-Down Timer Block	CCC Up Down Timer (HW)	UpDownTimer
Variable Confirm SF Block	CCC SFcn Variable Confirm (HW)	TimerAdj_CCC
Variable Debounce SF Block	CCC SFcn Variable Reset Debounce (HW)	DebounceVariableReset_CCC
Variable Timer Block	CCC Variable Timer (HW)	TimerAdjReset
Window Block	CCC Window (HW)	RangeLimitAdjFlagOnly

**Boeing:** The Boeing 787 Flight Controls library is supported. Include the following specification in .hilite command files, which automatically implies the use of HAM block library:

<Extensi on>Boei ng</Extensi on>

The following list of Boeing blocks are supported by HiLiTE.

**Table 20. Boeing Library Blocks Supported by HiLiTE**

Block Name	Mask Type	HiLiTE Block Name
amxovbpx	Boeing A-Minus-X-Over-B-Plus-X (HW)	PolynomialRatio
andPlusDelay	Boeing andPlusDelay (HW)	AndDelay
angle_remainder	Boeing Angle Remainder (HW)	NormalizeDeg_PlusMinus360
aovbpx	Boeing A-Over-B-Plus-X (HW)	PolynomialRatio
apbovx	Boeing A-Plus-B-Over-X (HW)	PolynomialRatio
apxovbpx	Boeing A-Plus-X-Over-B-Plus-X (HW)	PolynomialRatio
ArcSine_Degrees	Boeing Arcsine to Degrees (HW)	TrigonometricFunction
AvgWPV	Boeing Average with Past Value (HW)	AvgWPastValueReset
BackwardDifference	Boeing Backward Difference Integrator (HW)	IntegratorResetBoeing
BackwardDifference_Po sitionLimited	Boeing Backward Difference - Position Limited Integrator (HW)	IntegratorResetLimitedBoeing
Cosine_Degrees	Boeing Cosine of Degrees (HW)	TrigonometricFunction
DeadZone_Asymmetric	Boeing Asymmetric Dead Zone (HW)	DeadZoneAsymmetric
DeadZone_Symmetric	Boeing Symmetric Dead Zone (HW)	DeadZoneSymmetric
Differentiator	Boeing Differentiator (HW)	DifferentiatorReset
EdgeDetector	Boeing Edge Detector (HW)	OneShotBothEdgeReset
EdgeDetector_Leading	Boeing Leading Edge Detector (HW)	OneShotRisingReset
EdgeDetector_Trailing	Boeing Trailing Edge Detector (HW)	OneShotFallingReset
eqlz	Boeing eqlz (HW)	EqualizeIntegrator
ExpTimeConstant	Boeing Exponential Time Constant (HW)	ExpTimeConstant
Filter1stOrder_Tustin	Boeing 1st Order Tustin (HW)	LeadLagFilterTustin1stOrderRese t

Block Name	Mask Type	HiLiTE Block Name
Filter2ndOrder_Tustin	Boeing 2nd Order Tustin Filter (HW)	LeadLagFilterTustin2ndOrderReset
Filter2ndOrder_Tustin_Recalc	Boeing 2nd Order Tustin Filter Recalc (HW)	LeadLagFilterTustin2ndOrderResetRecalc
hgcon_five	AFC hgcon_five (HW)	HGCon
hgcon_four	AFC hgcon_four (HW)	HGCon
hgcon_two	AFC hgcon_two (HW)	HGCon
HoldOutput	Boeing Hold Output (HW)	SampleAndHoldReset
Hysteresis	Boeing Hysteresis (HW)	HysteresisBoeing
integ_cs_filter	AFC Integrator with Coast-Skip Reset with out Hold Output (HW)	IntegratorCoastSkipReset
integ_cs_filter_hld	AFC Integrator with Coast-Skip Reset with Hold Output (HW)	IntegratorCoastSkipResetHoldOut
Lag_ZTrans	Boeing Lag Z Transform (HW)	LagFilterZTransResetVal
Lag_ZTrans_reCalc	Boeing Lag Z Transform reCalc (HW)	LagFilterVarTauZTransResetVal
Lag_ZTransFreeze	Boeing Lag Z Transform reCalc with Freeze (HW)	LagFilterVarTauZTransHoldResetVal
Lag_ZTransFreezePosLimits	Boeing Lag Z Transform with Freeze and Position Limits (HW)	LagFilterZTransResetValLimFreeze
Lag_ZTransPosLimits	Boeing Lag Z Transform with Position Limits (HW)	LagFilterZTransResetValLim
Lag_ZTransRateLimits	Boeing Lag Z Transform with Rate Limits (HW)	LagFilterZTransResetValRateLim
Lag_ZTransRateLimits_reCalc	Boeing Lag Z Transform with Rate Limits reCalc (HW)	LagFilterVarTauZTransResetValRateLim
Lag_ZTransRateOut	Boeing Lag Z Transform with Rate Output (HW)	LagFilterZTransResetValRateOut
Lag1stOrder_Rectangular	Boeing Lag 1st Order Rectangular (HW)	LagFilter1stOrderRectResetFrInput
Lag1stOrder_Tustin	Boeing Lag 1st Order Tustin (HW)	LagFilterFixedTau1stOrderResetVal
Lag1stOrder_Tustin_Recalc	Boeing Lag 1st Order Tustin Recalc (HW)	LagFilterVarTau1stOrderResetVal
Latch_ROS	Boeing Reset Over Set Latch (HW)	LatchResetDomInit
Latch_SOR	Boeing Set Over Reset Latch (HW)	LatchSetDomInit
loc_logic_two	AFC loc_logic_two (HW)	loc_logic_two
master_sel_and_hld	AFC Master Select and Hold (HW)	PrioritySwitchResetHold
pfc_doff	PFC Time Delay Off (HW)	TimerAdjResetInitMode_DelayTrueFalse
pfc_don	PFC Time Delay On (HW)	TimerAdjResetInitMode_DelayTrueFalse
pfc_intg	PFC Integrator (HW)	SecondOrderIntegrator
pfc_lag_r_p	PFC Lag Filter with Rate and Position	LagFilterVarTau1stOrderResetVal

Block Name	Mask Type	HiLiTE Block Name
	Limiting (HW)	RLimPLim
pfc_latchsor	PFC Latch SOR (HW)	LatchSetDomInitVal
pfc_ldlg_r_p	PFC Lead-Lag Filter with Rate and Position Limits (HW)	LeadLagFilterVarTau1stOrderResetValRLimPLim
pfc_lhys	PFC Logical Hysteresis (HW)	HysterisisVarMidBwReset
pfc_mdyr	PFC Frame Delay (HW)	MultiFrameDelayResetVal
pfc_mfrm_delonoff	PFC Multiple Frame Delay On and Off (HW)	DebounceVarNumStepsResetVal
pfc_onof	PFC Delay On and Off (HW)	DebounceVariableResetVal
pfc_scmx	PFC Surface Command Mixer (HW)	SurfaceCommandMixerSym
pfc_scmxal	PFC Surface Command Mixer with Asymmetric Limits (HW)	SurfaceCommandMixerAsym
pfc_tfs	PFC Transient Free Switch with Constant Inputs (HW)	TFSConstantInputs
pfc_tfsch	PFC Transient Free Switch - Constant and Hold (HW)	TFSConstantHold
pfc_washout	PFC Tustin Washout Filter (HW)	WashoutFilterTustinResetVal
pfc_washout_r_p	PFC Washout Filter with Rate and Position Limiting (HW)	WashoutFilterRatePosLim
Rate_Asymmetric	Boeing Asymmetric Rate Limit (HW)	RateLimitAdjResetResetVal_Boeing
Rate_Symmetric	Boeing Symmetric Rate Limiter (HW)	RateLimitAdjSymmResetResetVal
ResolveAngle_PlusMinus180	Boeing ResolveAngle_PlusMinus180 (HW)	NormalizeDeg_PlusMinus180
Safe_Divide	Boeing Safe_Divide (HW)	IsSafeDivide
Safe_Reciprocal	Boeing Safe_Reciprocal (HW)	Reciprocal
SignOfReal	Boeing Sign of Real (HW)	Signum
Sine_Degrees	Boeing Sine of Degrees (HW)	TrigonometricFunction
sof	AFC sof (HW)	HGCon
Switch_3Position	Boeing 3-Position Switch (HW)	SwitchCascaded3Pos
switch_midval	AFC Switched Mid-Value Select (HW)	MidValueSelectAFCSwitched
Tan_Degrees	Boeing Tangent of Degrees (HW)	TrigonometricFunction
threshold	AFC Threshold (HW)	Threshold
TimeDelay_False	Boeing Time Delay to False (HW)	TimerAdjResetInitMode
TimeDelay_OnOff	Boeing TimeDelayOnOff (HW)	DebounceVariableResetSymm
TimeDelay_True	Boeing Time Delay to True (HW)	TimerAdjResetInitMode
trans_suppress	AFC Transient Suppressor with Tustin Filter (HW)	TransientSuppressWTustinFilter
TransferOfSign	Boeing SignFunc (HW)	TransferOfSign
TransportLagReal	Boeing Transport Lag Real (HW)	TransportLag
Trapezoidal	Boeing Trapezoidal Integrator (HW)	IntegratorResetBoeing

Block Name	Mask Type	HiLiTE Block Name
Trapezoidal_PositionLimited	Boeing Trapezoidal - Position Limited Integrator (HW)	IntegratorResetLimitedBoeing
TrapezoidPLTDSFDP.mdl	Trapezoidal - Position Limited Integrator DP (HW)	IntegratorResetLimitedBoeing
turn_dir_cmd	AFC Turn Direction Command (HW)	TurnDirCmd
Value_Asymmetric	Boeing Asymmetric Limiter (HW)	RangeLimitAdjVar
Value_Max	Boeing MaximumValueSelect (HW)	Max
Value_Mid	Boeing Middle Value Select (HW)	MidValueSelect
Value_Min	Boeing Minimum Value Select (HW)	Min
Value_Symmetric	Boeing Symmetric Limiter (HW)	RangeLimitAdjSymm
within_limits	AFC Within Limits (HW)	DualComparatorPFC
xch_csol	Boeing xch_csol (HW)	ValidatedLogicalVoter
xln_csol_eq	Boeing xln_csol_eq (HW)	RelationalOperatorXlnEx
xln_csol_ge	Boeing xln_csol_ge (HW)	RelationalOperatorXlnEx
xln_csol_gt	Boeing xln_csol_gt (HW)	RelationalOperatorXlnEx
xln_csol_le	Boeing xln_csol_le (HW)	RelationalOperatorXlnEx
xln_csol_lt	Boeing xln_csol_lt (HW)	RelationalOperatorXlnEx
xlnCsolLhys	Boeing xln_csol_lhys (HW)	HysterisisCSolVarMidBwReset
xlnCsolne	Boeing xln_csol_ne (HW)	RelationalOperatorXlnEx
xlnCsolTos	Boeing xln_csol_tos (HW)	RelationalOperatorXlnTxOfSign



## Appendix F. Generating Test Vector Output for Component Test Platform (CTP)

To use HiLiTE output with the Component Test Platform (CTP), you must generate a CTP-compatible input file with an extension ‘.tpi’. This file (henceforth called a TPI file) must conform to the CTP Test Vector File Format Specification in Ref. [2].

TPI files can be generated either per model or per block. Per model files contain all the test cases generated by HiLiTE for a single model. Per block files contain all the test cases generated for a specific block from a model. For per block testing, HiLiTE will generate one TPI file for each block in a model for which any tests are generated. (No tests are generated for certain kinds of blocks such as Input/Output blocks.)

A unique trace anchor is generated for each generated file. This trace anchor appears on the “AT3 Anchor:” line. Each file will also contain a list of all the unique trace tags for the blocks whose tests are contained in the file. Anchor trace tags can be specified in the model either in the form of annotation or in the revision block using define block requirements tracing for blocks present in the model. These trace tags appear on the “AT5 Source:” line. Trace tags for blocks present inside the model will appear on “AT5 Source:” line. Currently, the CTP Output Generator that comes with HiLiTE assumes these trace tags are available in each model as per HAM specifications or in annotation inside the model. This means that models that do not contain HAM Revision blocks or do not use HAM standards in recording trace information or do not contain annotation with trace tag starting with a specific requirement keyword-ToDocId pair will result in CTP files that do not contain this information.

### Correctness Criteria

The TPI file (containing Test Cases) will be deemed correct under the following conditions:

1. It conforms to the CTP Test Vector File Format and the specific requirements defined for the Component Test Platform (CTP).

2. Each test case conforms to the requirements for the test case as deemed correct for the particular type of test case generation (e.g. Simulink, Stateflow)
3. Each test case expresses a set of inputs and expected outputs for the specified model that is achievable, self consistent, and conforms to the intent of the model.

All of these criteria must be verified manually with the qualification artifacts. They should also be verified by use with CTP.

Each TPI file consists of two main parts: the header section, which identifies the model and the block that is the focus of the test, and lists all the input and output variables; and the test section, which consists of a sequence of tests.

The header section includes several text sections that are ignored by CTP. These include file identifying information followed by the anchor section mentioned previously. This is then followed by a list of all the input and then output variables, starting with data types, and then names. Output variables are further split into two groups—model level outputs, and test points. Test points are externally visible variables that are used solely for testing. These are mapped to variables in a data structure. If the test point handles non scalar data, the variable will be subscripted, with one line for each element of the structure.

Each test case in the test section corresponds to a single time step. Initial conditions are present at the beginning of the first test case in each file, as well as any time the Initialize keyword appears in the test section. There is no indication in the TPI file format if test cases are related to each other (HiLiTE supplies this information—it is simply unsupported by CTP).

Each test case has a header comment, followed by a list of input variable values, output variable values, and output tolerances. In the header comment, the “Testing” line indicates the block under test. This is the block whose HiLiTE Test Procedure was used to generate the test. The “Description” line will contain the description of the Procedure Template, if one has been specified. The “Inputs” line will contain the list of input variable names that are not don’t cares in the test vector for the block under test. The “Outputs” line will contain the list of testpoint and output variable names that are not don’t cares in the test vector for the block under test. The “TestReference” line in the test case header identifies the HiLiTE Test Procedure used to generate this test case. That Test Procedure name may be followed by a number in curly braces “{ }”—that indicates that the Test Procedure is vectorized. The values in the curly braces indicate which element in the vector this particular test applies to. If no curly braces are present, then this test case is for scalar data—the most common case. The name may also be followed by an integer in square brackets “[ ]”. This indicates that the test procedure has been applied to a multiport block. The subscript inside the “[ ]” indicates which port of the multiport this test procedure applies to. Both “{ }” and “[ ]” may follow the same name. The “Requirements” line will contain the list of requirement trace tags specified for the block under test.

Test Case values are specified in columns on a line. Each column corresponds to a line in the file header. For Test Case input values, there is one column for each input variable, and the columns are in the same order as the input variables in the header. For Test Case output value, there is one column for each output (normal + testpoint) and the columns are in the same order as the output variables.

Test case values may be specified exactly. If a particular variable value does not matter, the test case will indicate that by specifying “X” in the column for that variable. Tolerance values indicate the variation on either side of the specified output value that will still be considered as matching the specified output value. Tolerance values for don’t cares will be ignored.

## OptionSet Elements for Output Generation to CTP

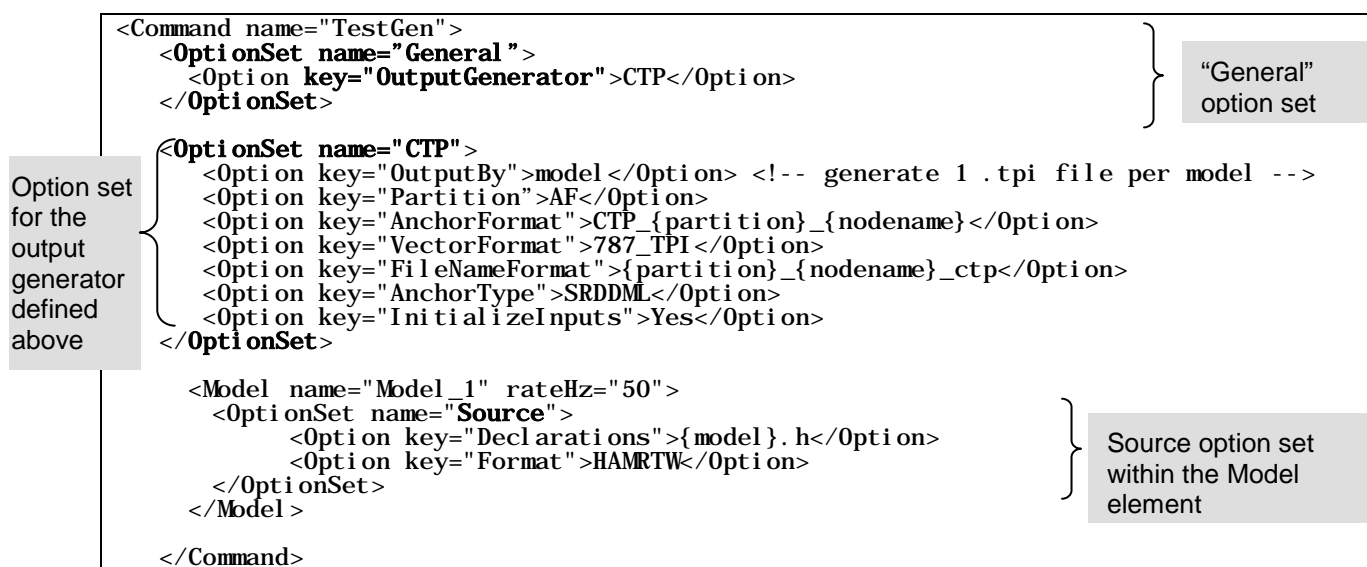
Option sets identify how HiLiTE will handle certain block characteristics, the test harness for which you want output generated, and other special circumstances. When you generate output for use with CTP, you must use this element to identify the output generator (CTP) and the vector format (e.g. EPIC Ascii).

OptionSet elements in a HiLiTE command file can be used as subelements of either the <Command> or <Model> element. Option sets included in the <Model> element take precedence over those in the <Command> element.

Option sets must be named. The “General” option set contains Option keys that require no further attributes and names other option sets that do require further detail. Use the “General” OptionSet to specify CTP as your output generator. CTP then becomes the name of an OptionSet for which you will specify the output format details.

- OptionSet **General**: Key attributes and values that can be used are described in Table 21.
- OptionSet **CTP** specifies the format for the test case vectors that HiLiTE creates, including anchor and partitioning information. Table 22 describes its keys and values.
- OptionSet **UseExplicitAT3Anchors** supports user-supplied AT3 anchors in generated TPI files, its option key is described in Table 24.
- OptionSet **Source** is used to find specific source code. See Table 2.

Figure 118 gives examples of option sets used in a HiLiTE command file that will create CTP output.



**Figure 118 - Example Command Element Containing CTP and Source OptionSets**

**Table 21. Key Values Available for the General OptionSet**

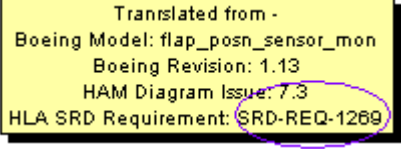
Keys	Definition and Available Values
<Option key="OutputGenerator">	<p>tells which test harness you want output formatted for.</p> <p><b>HiLiTE</b> specifies that HiLiTE should create a .csv file for the HiLiTE harness. This value is the default, so does not need to be specified.</p> <p><b>CTP</b> specifies that HiLiTE should create output to use in the CTP test harness. If you use this value, you must create a CTP option set in which you specify the character set to use in the output.</p> <p><b>SATGT</b> specifies the test harness to use for testing AS900 models.</p> <p>Other test harness options will become available in the future.</p>



**Table 22. Keys and Values Available for CTP OptionSet**

Key	Definition and Values
<Option key="VectorFormat">	<p>If the test harness can accept more than one format, use this key to identify the format you prefer. NOTE: This option belongs in the named OptionSet with the same name as the OutputGenerator option value. Valid values depend on the OutputGenerator chosen.</p> <p>For CTP output you can specify:</p> <p><b>EPIC-TPI</b></p> <p><b>787-TPI</b></p> <p><b>Binary</b></p>
<Option key="OutputBy">	<p>indicates how the CTP output tpi files are to be generated. Currently available values are:</p> <p><b>model</b> to generate one .tpi file for each model processed.</p> <p><b>block</b> to generate one .tpi file for each block under test.</p>
<Option key="Partition">	<p>Denotes the name of the Partition to use in the tpi file anchor and the tpi file name. <b>Example:</b></p> <pre>&lt;Option key="Partition"&gt;AF&lt;/Option&gt;</pre>
<Option key="AnchorFormat">	<p>is a string format for how the .TPI anchor should be created.</p> <p><b>Example:</b></p> <pre>&lt;Option key="AnchorFormat"&gt;CTP_{partition}_- {nodename}&lt;/Option&gt;</pre> <p>{partition} indicates where to insert the partition information.</p> <p>{nodename} indicates where to insert the fully qualified name of either the model or the block under test. If output is by model, the nodename will be the model name; if by block, the nodename is the block name.</p>
<Option key="FileNameFormat">	<p>is a string for generating a filename for CTP Optionsets.</p> <p><b>Example:</b></p> <pre>&lt;Option key="FileNameFormat"&gt;- CTP_{partition}_{nodename}&lt;/Option&gt;</pre> <p>{partition} represents where to partition the file name</p> <p>{nodename} indicates the block or model name should be.</p> <p>The entire string is used as a string format to generate the file name. If you want to use the partition as a subdirectory name, you can format the option as in the next example.</p> <p><b>Example:</b></p> <pre>&lt;Option key="FileNameFormat"&gt;- {partition}\{nodename}_ctp&lt;/Option&gt;</pre>

Key	Definition and Values
<Option key="InitializeInputs">	<p>Indicates whether HiLiTE should generate input variables that need to be initialized. "InitializeInputs" will list all the input variables along with initial values in the tpi file, before any actual test case or Initialize keyword. CTP tool can set input variables with these initial values to avoid things such as divide by zero issue.</p> <p><b>Example:</b></p> <pre>&lt;Option key="InitializeInputs"&gt;Yes&lt;/Option&gt;</pre> <p>User can specify 'Yes' or 'No'. By default, InitializeInputs is set to 'Yes'. Input variables are listed in the "variable initializations" section of tpi file.</p> <pre>; variable initializations ; variableName initialValue ; end initializations</pre>
<Option key="DefaultAnchor">	<p>Indicates whether HiLiTE should generate defaultAnchor (SRDDML) in the AnchorTrace section (AT5 Source: line) of TPI file. Use this option to enable/disable the generation of a default anchor.</p> <p>For any model, the default anchor is named "Filename_MDL"; it is categorized as an SRDDML anchor. If an SRDDML anchor is not specified for a state chart at the model level, HiLiTE will log an error message to the HiLiTE log file.</p> <p>Note that by default, Default Anchor is set to "Yes" (HiLiTE will generate a default anchor if this option is not specified in the HiLiTE command file).</p> <p><b>Example:</b></p> <pre>&lt;Option key="DefaultAnchor"&gt;No&lt;/Option&gt;</pre>
<Option key="AnchorType">	<p><b>When Anchor Trace is specified in Revision Block</b></p> <p>Indicates which anchor to select from the revision block of the model. AnchorType's "SRDDML", "SRDD", "SRDDL" and "SRDDML/SRDD/SRDDLL" are currently supported by HiLiTE. "SRDDML/SRDD/SRDDLL" means that HiLiTE will look out for all three anchors in the revision block and list them in the tpi file.</p> <p>Note that by default, HiLiTE will generate <i>FileName_MDL</i> as SRDDML anchor tag for all models.</p> <p><b>Example:</b></p> <pre>&lt;Option key="AnchorType"&gt;SRDDML &lt;/Option&gt; &lt;Option key="AnchorType"&gt;SRDD&lt;/Option&gt; &lt;Option key="AnchorType"&gt;SRDDL&lt;/Option&gt; &lt;Option key="AnchorType"&gt;SRDDML/SRDD/SRDDLL&lt;/Option&gt;</pre> <p>By default, AnchorType is set to "SRDDML".</p> <p><b>Decoding requirement tag</b></p> <p>For requirement tag {'other' ' ' 'true' 'srd_5_2' 'SRDD_AFS' 'SRDD_COR'},</p> <p>'other' – (Index: 0)  ' ' – (Index: 1)  " – (Index: 2)  'true' – (Index: 3)  'srd_5_2' – (Index: 4. this corresponds to SRDDML anchor)  'SRDD_AFS' – (Index: 5. this corresponds to SRDD anchor)</p>

Key	Definition and Values
	<p>'SRDD_COR' – (Index: 6. this corresponds to SRDDL anchor)</p> <p>Requirement tags should conform to this format for HiLiTE to properly identify the AnchorTypes.</p> <p><b>When Anchor Trace is specified as annotation in the model</b></p> <p>For certain models (ex: Deihl HL models), anchor trace is specified as annotation in the model instead of specifying the trace in the revision block. HiLiTE will extract the anchor trace from annotation in the model based on the keyword specified by the user in the .hilite file. Also user can specify the To-Doc Id, along with keyword.</p> <p><b>Keyword Example:</b> For all HL models, anchor trace starts with a certain keyword. In the below figure, keyword is “SRD-REQ-”. It is the responsibility of the user to properly specify the keyword and To-Doc Id in the .hilite file. The anchor trace (in annotation) looks like the following.</p>  <p><b>Example1 :</b></p> <pre>&lt;Option key="AnchorType"&gt;Annotation{SRD-REQ-}{SRDD} &lt;/Option&gt;</pre> <p>Annotation tells HiLiTE to look for anchor trace in the annotation of the model. The keyword should be specified in between the flower brackets as shown in the example above. Here SRDD is the To-Doc Id.</p> <p><b>Example2 (Multiple requirement keyword-ToDocId pairs):</b></p> <pre>&lt;Option key="AnchorType"&gt;Annotation{SRD-REQ-}{SRDD},{SDD-REQ-}{SRDDML} &lt;/Option&gt;</pre> <p>If there are more than one requirement keyword to look for in the annotation of the model, user has to specify that information in the HiLiTE file properly. HiLiTE supports multiple requirement keyword-ToDoc Id paris. Note that multiple requirement keyword-ToDoc Id paris are separated by a comma.</p> <p>In the above example, the first requirement keyword is “SRD-REQ-“ and the corresponding ToDoc Id is “SRDD”. The second requirement keyword is “SDD-REQ-“ and the corresponding ToDoc Id is “SRDDML”.</p>
<pre>&lt;Option key="ResolveUsableIOUsingHAMScript"&gt;</pre>	<p>Indicates whether HiLiTE should resolve UsableIO using HAM script. When test vectors are generated in isolation for StateChart, sometimes HiLiTE could not resolve the actual UsableIO. In this case, HiLiTE will automatically suggest in the log file to enable this option. Enabling this option might help HiLiTE to resolve the input signal names which are driving the state chart under test. If you enable this option, make sure that the following path is added in MATLAB's path variable : &lt;HAM&gt;\ham5release10\ham\cCodeReview by following these steps:</p> <ol style="list-style-type: none"> <li>1) In the MATLAB window select menu option <b>File-&gt;Set Path-&gt;Add with Subfolders</b></li> <li>2) <b>Browse</b> for the required path and click <b>OK</b></li> <li>3) Click <b>Save</b>.</li> </ol>

Key	Definition and Values
	<p>Example:</p> <pre>&lt;Option key= " ResolveUsableIOUsingHAMScript"&gt; True &lt;/Option&gt;</pre> <p>User can specify 'True' or 'False'. By default, 'ResolveUsableIOUsingHAMScript' is set to 'False'.</p> <p><b>Example Model:</b> This scenario can be found in PEDALRIGTEST43.mdl</p>
<pre>&lt;Option key="EnableLoopCount"&gt;</pre>	<p>Indicates whether HiLiTE should enable loop count in the TPI file. Loop statements look like comments and specify the number of iterations to be executed in the loop. (The Loop statement must appear in one of these two locations: either 1-before the Test Case # line or 2-immediately before the INPUT VARIABLES line for a given test case.).</p> <p>When the testcase header is followed by Loop count. B787 CTP TPI files shall accept loop statements in the following format:</p> <p style="text-align: center;">;Loop x</p> <p>where:</p> <ul style="list-style-type: none"> <li>▪ The first character of the line is a semi-colon</li> <li>▪ The word Loop (case-insensitive ) follows immediately after the semi-colon (no space or tab)</li> <li>▪ One or more spaces follow the word Loop</li> <li>▪ No other characters appear after the word Loop and before the iteration count x</li> <li>▪ x appears on the same physical line</li> <li>▪ x specifies an integer with a value greater than zero</li> </ul> <p>Note that by default, EnableLoopCount is set to "False"</p> <p><b>Example:</b></p> <pre>&lt;Option key="EnableLoopCount"&gt;true&lt;/Option&gt;</pre>
<pre>&lt;Option key="PartitionLocationInTraceTag"&gt;</pre> <p>(this option key was used previously but is not recommended for most usage now)</p>	<p>Indicates where the Partition information is within the trace tag. This information is used to generate the .tpi file anchor. The value is a regular expression that conforms to Microsoft VS.NET regular expression syntax.</p> <p>Example:</p> <pre>&lt;Option key="PartitionLocationInTraceTag"&gt;[a-zA-Z]_(?&amp;lt;partition&amp;gt;[a-zA-Z]*)_d+&lt;/Option&gt;</pre> <p>Note that &amp;lt;partition&amp;gt; in this example is the escaped equivalent of "&lt;partition&gt;". It is escaped because it is embedded in XML and XML treats "&lt;" and "&gt;" as special characters. The example expression says that the partition is embedded in a string that has an alpha prefix and a numeric suffix of the form SRDD_PFC_024842 (PFC is the partition).</p>

**Table 23. Keys and Values Available for the Source OptionSet**

Key	Definition and Values
<Option key="Declarations">	<p>The value for this key is the name of a file containing variable declarations for use by an OutputGenerator. The value can be a "concrete" name or a name template. An example of a "concrete" name is <b>my_model.h</b></p> <p>A name template has the format:  <b>{model}.h</b> HiLiTE will substitute the name of the model under analysis for {model}</p> <p>The name may be relative or fully qualified. The name (either as specified or generated from the name template) is then used to locate the source code file that includes declarations.</p>
<Option key="Format">	<p>The value for this key indicates the source code format of the file named in the Declarations option. This keyword is used to look up the class used to parse the file containing the declaration information.</p> <p><b>HAMRTW</b> is currently the only valid value.</p>
<Option key="OutOfMemoryProtection"> Default: off	<p>Due to some performance problems with tpi file generation, an option called "OutOfMemoryProtection" has been added. This option, if specified and set to "on", tells the CTPOutputGenerator to use a temp file on disk while creating the tpi files. If it is not specified OR if set to "off", the CTPOutputGenerator will simply use an in memory string. For large tpi files, you will want to set this option to "on" (or "true" - they both work). Since few models run into OutOfMemory errors, you can simply wait to see which models have this particular problem and then turn this option on for those models only.</p> <p><b>Example:</b>  &lt;Opti on key="OutOfMemoryProtecti on"&gt;on&lt;/Opti on&gt;</p>

**Note** If you specify whether test points are on or off (see information about option sets in Section 4, Table 2), the CTP output will be affected. By default, the test points are on. If you leave this default, but the declarations file has no information about test points, then HiLiTE will not generate any tpi file. A special case occurs if test points have siblings that are outputs; these test points will be ignored. If *all* test points in the model are for outputs, then HiLiTE will ignore the declarations file and generate the tpi file.

**Table 24. Keys and Values Available for UseExplicitAT3Anchors OptionSet**

Key	Definition and Values
<Option key="CTP_Stateflow_Isolation/StateChart1">	<p>The value for this key is the name of a StateFlow chart.</p> <p><b>Example:</b>  &lt;OptionSet name="UseExplicitAT3Anchors"&gt;    &lt;Option key=      "CTP_Stateflow_Isolation/StateChart1"&gt;      Name1&lt;/Option&gt;  &lt;/OptionSet&gt;</p>

## Anchor Tracing in TPI files

When an anchor type is specified as SRDDML/SRDD/SRDDLL, HiLiTE extracts all three anchor types from the model irrespective of whether test vectors are generated for the blocks that reference those anchors as specified in revision block. All anchors specified in the Revision block of the model appear in the AT5 header of the TPI file, as shown in Figure 119 and Figure 120. In the "Requirements" line for the test cases for the blocks in the TPI file, the anchors to which the block traces appear as shown in Figure 121.

For Stateflow charts, HiLiTE extracts the SRDDML anchors specified in the States, Transitions, TruthTables as specially formatted comments. The union of all these anchors appears in the "AT5 Source" line of Anchor-Trace section in the TPI file header. In this header, all anchors are extracted from the Stateflow chart, regardless of whether a particular state, transition, or condition was tested or not.

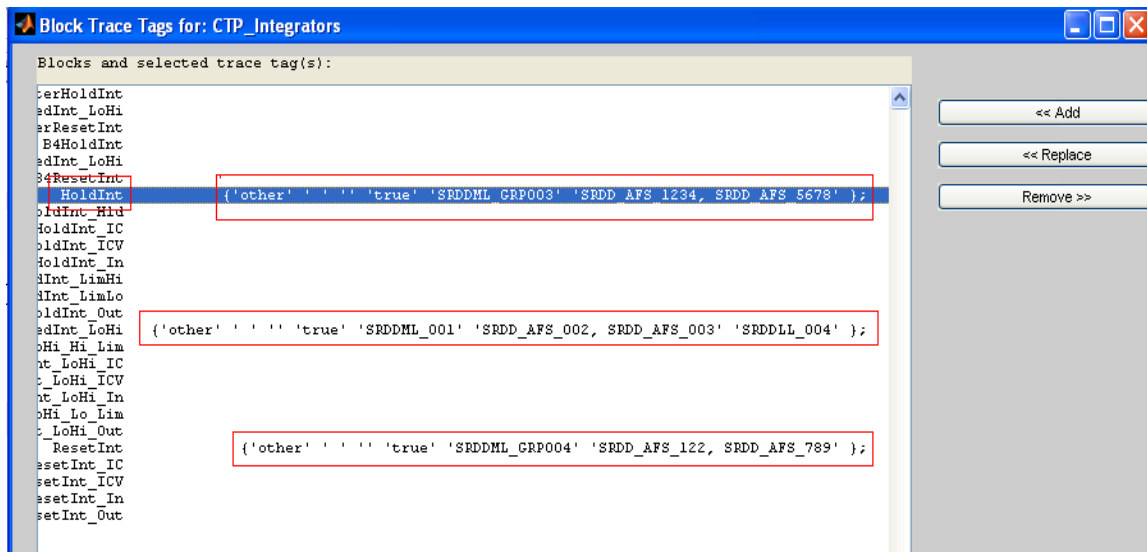


Figure 119 - Anchors specified in the Revision block of the model

```

;*****
; AT1 Begin Anchor / Trace data
; AT2 Do not modify this line or the line below
; AT3 Anchor: CTP_CTP_Integrators
; AT4 Do not modify this line or the line above
; AT5 Source: SRDDML: CTP_Integrators_mdl,SRDDML_GRP003,SRDDML_001,SRDDML_GRP004; SRDD: SRDD_AFS_1234,
SRDD_AFS_5678,SRDD_AFS_002, SRDD_AFS_003,SRDD_AFS_122, SRDD_AFS_789; SRDDL: SRDDL_004
; AT6 End Anchor / Trace data
;*****
; #Inputs #Outputs #TestPoints #Testcases

```

Figure 120 - All Anchors specified in revision block appear in AT5 Source in TPI file

```

;*****
; Test Case # 19
; Testing : CTP_Integrators/HoldInt
; Description : HOLD Operation
; Inputs : 10)HoldInt_ICV,11)HoldInt_IC,12)HoldInt_Hld,13)HoldInt_LimLo,14)HoldInt_LimHi
; Outputs : 3)HoldInt_Out
; TestReference : This is to test if output is held when HOLD is TRUE.
; Type :
; Requirements : SRDDML_GRP003,SRDD_AFS_1234,SRDD_AFS_5678
;*****

```

**Figure 121 - Anchor appearing in "Requirements" line for the blocks in the TPI file for which supported anchors are specified in Revision block**

## **Appendix G. Block Attributes Lists**

This appendix lists block attributes as supported by the HiLiTE. Tables below describe:

- Attributes for HAM blocks
- Attributes for Boeing blocks
- Attributes for CCC blocks
- Common attributes (for all blocks)



Table 25. HAM block attributes

Mask Type	HiLiTEBlockType	Attribute	Description
Asymmetric Limit (HW)	RangeLimitFixed	LoLimit HiLimit	Return LL value of the block Returns LU attribute value of block
Bit Shift (HW)	BitShift	shiftRight	Returns value after right shift
Bit Unpack (HW)	BitOr	BitMaskHex	Returns bitMask value in hex notation
Bitwise And (HW)	BitAnd1	bitMask	Returns bitMask value
Confirm ON (HW) / Confirm OFF (HW)	TimerFixed	ConfirmVal TMAX InitOPDrivenByIP	Returns TRUE for Confirm ON (HW) block Returns FALSE for Confirm OFF (HW) block Returns the ON/OFF time Returns TRUE if Initial Output is set to Input
Deadband (HW)	DeadBand	DeadBandWidth	Returns BW of the block
Deadzone (HW)	Deadzone	LoLimit HiLimit	Return LL value of the block Returns UL value of block
Decoder (HW)	Decoder	Num StartingValue	Returns total number of output ports Returns the Starting value
Fader (HW)	Fader	fadetime	Return Fadetime of the block
Limited Reset Hold Integrator (HW)	IntegratorResetLimitedHold	Lim1IsLoLim Lim2IsLoLim	Returns True if Lim1 is the Low limit port. Returns True if Lim2 is the Low limit port.
Limited Reset Integrator (HW)	IntegratorResetLimited	Lim1IsLoLim Lim2IsLoLim	Returns True if Lim1 is the Low limit port. Returns True if Lim2 is the Low limit port.
Limiter Hold (HW)	RangeLimitFixedWithFlag	LoLimit HiLimit	Return LL value of the block Returns LU attribute value of block
Mux Selector (HW)	MuxedSelector	getoutput	getoutput is not supported for Mux Selector (HW). Test vector generation is disabled for this block.
Non-Mux Selector (HW)	PortSelector	getoutput	getoutput is not supported for Non-Mux Selector (HW). Test vector generation is disabled for this block.

Mask Type	HiLiTEBlockType	Attribute	Description
Register (HW)	Register	IsEvenPort IsOddPort	Returns True for even numbered port Returns True for odd numbered port
Relay (HW)	HysterisisFixed	RelayOnPoint RelayOffPoint RelayOnValue RelayOffValue	Returns onpt (RelayOnPoint) value of the block Returns offpt (RelayOffPoint) of block Returns onpt (RelayOnPoint) value of the block Returns offpt (RelayOffPoint) of block
Reset Lead Lag Filter (HW)	LeadLagFilterFixedTauResetVal	Td LoLimit HiLimit	Returns time constant, Tau Return LL value of the block Return LU value of the block
Simple Delay (HW)	UnitDelay	InitialCondition	Returns IC value of the block
Simple Integrator (HW)	IntegratorSimple	OutputIC	Returns IC value of the block
Switched Fader (HW)	FaderSwitched	fadetime	Returns fadetime of block
Symmetrical Rate Limiter (HW)	RateLimitFixed	HiLimit	Return UL value of block
Symmetrical Rate Limiter (HW)	RateLimitFixed	LoLimit	Return LL value of the block
Variable Asymmetrical Limit (HW)	RangeLimitAdjVar	Lim1IsLoLim	Returns True if Lim1 is the Low limit port
Variable Asymmetrical Limit (HW)	RangeLimitAdjVar	Lim2IsLoLim	Returns True if Lim2 is the Low limit port
Washout Filter (HW)	WashoutFilterFixedTau	Td	Returns time constant, Tau
Washout Filter (HW)	WashoutFilterFixedTau	Tn	Returns time constant, Tau

Table 26. Boeing block attributes

Mask Type	HiLiTEBlockType	Attribute	Description
AFC Transient Suppressor with Tustin Filter (HW)	TransientSuppressWTustinFilter	Td	Returns time constant, Tau.
Boeing Lag 1st Order Rectangular (HW)	LagFilter1stOrderRectResetFrInput	Td	Returns time constant, Tau.

Mask Type	HiLiTEBlockType	Attribute	Description
Boeing Lag 1st Order Tustin (HW)	LagFilterFixedTau1stOrderResetVal	Td	Returns time constant, Tau.
Boeing Lag 1st Order Tustin Recalc (HW)	LagFilterVarTau1stOrderResetVal	Td	Returns time constant, Tau.
Boeing Lag Z Transform reCalc with Freeze (HW)	LagFilterVarTauZTransHoldResetVal	Td	Returns time constant, Tau.
Boeing Lag Z Transform with Freeze and Position Limits (HW)	LagFilterZTransResetValLim	Td	Returns time constant, Tau.
Boeing Lag Z Transform with Freeze and Position Limits (HW)	LagFilterZTransResetValLimFreeze	Td	Returns time constant, Tau.
Boeing Lag Z Transform with Position Limits (HW)	LagFilterZTransHoldResetValLim	Td	Returns time constant, Tau.
Boeing Lag Z Transform with Rate Limits (HW)	LagFilterZTransResetVal	Td	Returns time constant, Tau.
Boeing Lag Z Transform with Rate Limits (HW)	LagFilterZTransResetValRateLim	Td	Returns time constant, Tau.
Boeing Lag Z Transform with Rate Limits reCalc (HW)	LagFilterVarTauZTransResetVal	Td	Returns time constant, Tau.
Boeing Lag Z Transform with Rate Output (HW)	LagFilterZTransResetValRateOut	Td	Returns time constant, Tau.
Boeing Time Delay to True (HW) / Boeing Time Delay to False (HW)	TimerAdjResetInitMode	ConfirmVal	Returns True for TimeDelayTRUE. Returns False for TimeDelayFalse.
Boeing Transport Lag Real (HW)	TransportLag	DelayVal	Returns delay time in seconds.
Boeing Washout Z-Transform Filter (HW)	WashoutFilterZTransReset	Tn Td	Returns time constant, Tau.
confirmOff_defaultIn (HW)	TimerAdjResetInitMode_Subsys_OffInOnIn	ConfirmVal	Returns True/False.
confirmOff_defaultOn (HW)	TimerAdjResetInitMode_Subsys_OffOnOnOff	ConfirmVal	Returns True/False.
confirmOff_defaultOn (HW)	TimerAdjResetInitMode_Subsys_OffOnOnOff	ICBool	Returns True/False.
confirmOn_defaultOn (HW)	TimerAdjResetInitMode_Subsys_OffOffOnOn	ConfirmVal	Returns True/False.
PastValueBooleanNI	UnitDelayResetOnly	InitialCondition	Returns IC value of the block.

Mask Type	HiLiTEBlockType	Attribute	Description
PFC Time Delay On (HW) / PFC Time Delay off (HW)	TimerAdjResetInitMode_DelayTrueFalse	ConfirmVal	Returns True for TimeDelayOn. Returns False for TimeDelayOff.
PFC Tustin Washout Filter (HW)	WashoutFilterTustinResetVal	Tn Td	Returns time constant, Tau.
PFC Washout Filter with Rate and Position Limiting (HW)	WashoutFilterRatePosLim	Tn Td	Returns time constant, Tau.

**Table 27. CCC block attributes**

Mask Type	HiLiTEBlockType	Attributes	Description
CCC Differentiator Rectangular (HW)	DifferentiatorResetLimited	OutputIC	Returns IC value of the block.
CCC Range Limit (HW)	RangeLimitAdjVar	Lim1IsLoLim Lim2IsLoLim	Returns True if Lim1 is the Low limit port. Returns True if Lim2 is the Low limit port.
CCC SFcn Confirm ON (HW)	TimerFixed_CCC	ConfirmVal TMAX InitOPDrivenByIP	Returns true Returns confirmTime of the block Returns True if Initial Output is set to Input
CCC SFcn Variable Reset Debounce (HW)	DebounceVariableReset_CCC	OutputIC	Returns IC value of the block.

**Table 28. Common block attributes**

<b>CommonBlock attributes</b>	<b>Description</b>
AllInputsAllowContainZero	Returns Boolean value '1', if the maximum allowable range of all inputs contain zero.
AllInputsBool	Returns Boolean value '1', if all inputs of the block has Boolean datatype.
AllInputsConstant	Returns Boolean value '1', if all inputs of the block are constants (hard wired
AllInputsContainZero	Returns Boolean value '1', if all inputs of the block can have a zero value. A test case for all 0 inputs should be enabled only when this is true.
AllInputsInteger	Returns Boolean value '1', if all inputs of the block has integer (signed and unsigned integer) datatype.
AllInputsOpContainZero	Returns Boolean value '1', if the normal operating range of all inputs contain zero.
AllInputsReal	Returns Boolean value '1', if all inputs of the block has Real datatype.
AllInputsTied	Returns Boolean value '1', if all inputs of the block are tied together.
ICV_Input_Tied	When the input port and ICV port of a block are tied together, this attribute will be set to TRUE. Otherwise, it will be set to FALSE
InputsRelated	When the inputs of a block get their values from the same source (when there is a fork join), this attribute will be set to TRUE. Otherwise, it will be set to FALSE
SomeInputsRelatedNotTied	This indicates that some inputs of the blocks are related to each other but not tied together. If this is true then a test case for specific or same values at the inputs or all max (or all min) values at the inputs is not possible.
SomeInputsTied	Returns Boolean value '1', if some inputs of the block are tied together. If this is true then a test case for different values at the inputs (hard coded, one max and one min) values at the inputs is not possible.
TotalInPortCount	Returns total count of data port and not the control port
TotalOutPortCount	Returns the count of output ports

## **Appendix H. Command File Elements for Diagnostics and Development**

This appendix describes HiLiTE command file elements that are useful for developers and other HiLiTE support team members who need to diagnose issues.

**Table 29. Command file elements used for development and diagnostics**

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<b>&lt;HiLiTEgmt /&gt;</b>	<p>If this optional element is specified, it must be the first element in a HiLiTE command file. Use it to manage the way HiLiTE runs. The following attributes are useful for diagnostics:</p> <p><i>maxRetries</i> tells how many times HiLiTE will try to start MATLAB before failing.</p> <p><i>priority</i> sets the urgency of the HiLiTE run. (Default = Low) Possible values are the Windows task priority levels: "High," "AboveNormal," "Normal," "BelowNormal," "Low". At Low, HiLiTE runs in the background and you can continue working on other applications. You may want to change the priority if you are running a very complex model or batch of models that would otherwise take a very long time to run.</p> <p><i>raceDelayMS</i> in rare cases of improper execution with MATLAB, this attribute can be used to specify the length of a delay to overcome race conditions between a model clock and the data inputs associated with the clock. Use this attribute to tune MATLAB if model output is garbled or MATLAB crashes.</p> <p><i>secsBetweenTries</i> tells how long HiLiTE will wait before the next try to start the MATLAB.</p>	<pre> &lt;HiLiTEgmt maxRetries="12" /&gt;  &lt;HiLiTEgmt priority="High" /&gt;  &lt;HiLiTEgmt raceDelayMS="6" /&gt;  &lt;HiLiTEgmt secsBetweenTries="15" /&gt; </pre>
<b>&lt;Model&gt;</b> attributes for model information output in XML	<p>include these optional Model attributes to output the HiLiTE internal representation of the model. They are useful for debugging.</p> <p><i>rawXML</i>: (Default= 'no'). If set to 'yes,' XML will be generated from the first (raw) stage of the import process. This raw XML can be used to look through model details.</p> <p><i>instanceXML</i>: (Default= 'no'). If set to 'yes', XML will be generated from the second (instance) stage of the import process. This XML can be used to look through model details.</p>	<pre> &lt;Command name='ModelCheck' &gt;   &lt;Model name='AR065h' rawXML='yes'     instanceXML='yes' /&gt; &lt;/Command&gt; </pre>
<b>&lt;Command&gt; name</b>  Additional commands	<p>use either of these commands to verify the validity of formulas and models:</p> <p><i>FormulaCheck</i>—Use this command whenever the database is updated—in particular, when any of the templates have been changed. <i>FormulaCheck</i> reads all the formulae in the database and applies the regular expressions described in the file specified by <i>FormulaTemplates</i> to those formulae. The required attribute:</p>	<pre> &lt;Command name='FormulaCheck'   FormulaTemplates='TFLKeywords.properties' /&gt; </pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p><i>FormulaTemplates</i> names the file holding regular expressions.</p> <p><i>ModelCheck</i>—Use this command for conducting an initial sanity check on a new model. This command invokes model analysis to look for unreachable sections, divide by zero, and other inherent problems in a model. Use this command to check a new or edited model.</p>	<b>&lt;Command name='ModelCheck' /&gt;</b>
<b>&lt;Command name='TestGen' outputSorter =</b>	<p>use these TestGen command attributes:</p> <p><i>outputSorter</i> (default: WeakTopoSort) to specify how the output columns are to be sorted. By default, the columns are sorted close to topological order (inputs to outputs). Other sorting possibilities are available; they are defined in <i>keywordmapping.xml</i></p> <p><i>MaximizeOverrideUsage</i>: (Default= "no") If "yes," the test generator will take every possible override it can, that is, not just where overrides are specified in the model or the .hilite file, but wherever block implementations allow it. Use this sparingly.</p>	<pre>&lt;Command name='TestGen'   outputSorter='TopologicalSort'&gt;   &lt;Model name='testExample1' /&gt; &lt;/Command&gt;  &lt;Command name='TestGen'   MaximizeOverrideUsage="yes" &gt;   &lt;Model name='tm390e_new' /&gt; &lt;/Command&gt;</pre>
<TestOutput> additional subelements (Future)	<p>These elements are useful for retesting a model with different inputs or other changes.</p> <p>Future Options:</p> <p>The &lt;Date&gt; and &lt;Time&gt; elements have a fmt attribute that specifies how HiLiTE should show the date/time in the folder name. Use standard date/time designators (d=day, m=month, y=year, etc.)</p>	
<b>&lt;LibraryMapping /&gt;</b> for use only by HiLiTE developers	<p>names files that map user block type names to HiLiTE-supported block types. For basic HiLiTE usage, the Extension element (see Table 1) replaces LibraryMapping. Use LibraryMapping only if your project includes blocks that are mapped for groups not supported by the &lt;Extension&gt; element. See Appendix E for more information. If you do not use a &lt;LibraryMapping&gt; or &lt;Extension&gt; element, HiLiTE will default to raw Simulink block support.</p> <p>LibraryMapping has one attribute:</p> <p><i>fName</i> the name of the library .xml file. Name each library in a separate &lt;LibraryMapping&gt; element and list them in descending priority order. HiLiTE searches for LibraryMapping files in this order:</p> <ol style="list-style-type: none"> <li>1. Assumes the fName is fully qualified.</li> <li>2. Assumes the fName is relative to the working directory.</li> </ol>	<b>&lt;LibraryMapping fName='MyLibraryMapping.xml'/&gt;</b>



Element Tag, Attribute Name, and/or OptionSet	Description	Example
	<p>3. Assumes the fName is relative to the ProjectPath directory</p> <p>4. Assumes the fName is relative to the DATA directory.</p> <p>If libraries have overlapping block type names, HiLiTE uses the first one in which it finds the name.</p> <p>You can name more than one library for mapping models.</p>	
<p><b>&lt;ToolKeywords&gt;</b></p> <p><b>&lt;ConstantsMapping&gt;</b></p>	<p>refer HiLiTE to a list of strings to interpret as keywords, not identifiers—useful for suppressing spurious warnings. Includes one attribute:</p> <p><i>fName</i> names the file (.xml format) that contains the keywords you want HiLiTE to recognize.</p>	<p><b>&lt;ToolKeywords fName=</b> "SimulinkKeywords.xml" /&gt;</p> <p><b>&lt;ConstantsMapping fName=</b> "SimulinkConstants.xml" /&gt;</p>
<p>&lt;OptionSet name= <b>"SignalBoundaries"</b> Required subelement &lt;Option&gt; &lt;/Option&gt;</p>	<p>Developers or users who need to diagnose problems may want to try some of the following key attributes for Option subelements of this OptionSet :</p> <p><i>EnableTestPoint</i> or <i>DisableTestPoint</i> tell HiLiTE to enable or disable a specifically named Test Point. These can be used in conjunction with the other Options to fine tune which test points are used. The element value for each Option should be the fully qualified name of the port where the test point is attached.</p> <p><i>Overrides</i> tells HiLiTE to perform an override on specific signals (outputs of blocks) in a model. The value is the name of the output block. This override create new inputs in the model, but the same range constraints will be applied to them as coming from the range propagation from upstream to the point at which override is applied. Multiple override specifications can be separated by a space. Make sure you have test point enabled code before you use this. There must be a test point or a model output at the point where you specify the override. The overrides are specified at the output port of a block (this is different from suggested overrides which are mostly at input ports, so you may have to trace to the previous block and specify the override there. Each override must be manually determined by the user and put in the .hilite file.</p> <p>Note: the Element &lt;OverrideExamplesForCommandOption&gt; in Modelname_Summary.xml file contains the overrides data. You can copy the text within this element and put it in the Overrides option in the .hilite file.</p>	<p>&lt;OptionSet name="<b>SignalBoundaries</b>" &lt;Option <b>key="Overrides"</b>&gt; BQT_New_Overrides/sum.Output BQT_New_Overrides/sum1.Output BQT_New_Overrides/subsystem/ SubSys_Sum.Output &lt;/Option&gt; &lt;/OptionSet</p>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<p>&lt;OptionSet name="Scripts"&gt;  Required subelement  &lt;Option&gt; &lt;/Option&gt;</p>	<p>lets you specify one or more MATLAB scripts (.m file) to be executed on a model either before or after HiLiTE loads the model. Option keys specify the path of the MATLAB script files, the names of a script file, and when the script is executed.</p> <p><i>Path</i> specifies the path of the MATLAB script file. There is a provision to specify the path in absolute terms as well as in relative terms. In most cases, standard scripts will be packaged with HiLiTE and the path specification will use the keyword {HiLiTE_HOME} and the reference to the Scripts directory.</p> <p><i>PostLoad</i> PostLoad option key lets the user to specify a MATLAB script file name (without .m extension) to be executed by HiLiTE after loading the model. User can specify multiple preload script files – HiLiTE will run them one by one after loading the model.</p> <p><i>PreLoad</i> lets you specify a MATLAB script file name (without .m extension) to be executed by HiLiTE before loading the model for processing, analysis, and test generation. You can specify multiple preload script files, HiLiTE will run them one by one before loading the model.</p> <ul style="list-style-type: none"> <li>When you use the PreLoad attribute, you can give the Option a value of <i>BusCreators</i>, which is a standard script containing input Bus Objects. It must be used with models that use input Bus Objects. This script allows replacing the inports using the Bus Objects with the actual Bus Creators using the Bus Object.</li> <li>Boeing models like MCP or Diehl often use lot of Bus Objects (BO) which are huge structures. In these models, bus objects are used in the input ports so as to reuse the same bus object across different models without the usage of Bus Creators in all these models. The script (BusCreators.m) finds all the inports in the model that uses the BO and replaces this inport with a Bus Creator Block with appropriate input ports attached using the same Bus Object.</li> <li>When BusCreators.m script file is executed by the HiLiTE, it modifies the model. Since we don't want the model to get modified, HiLiTE makes a copy of the model with a random name just before opening the model. After querying and closing the model, HiLiTE will delete the modified model and renames the randomly named model with the actual model name.</li> </ul>	<pre> &lt;OptionSet name="Scripts"&gt;   &lt;Option key="Path"&gt;C:\MyDirectory &lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="Scripts"&gt;   &lt;Option key=     "Path"&gt;{HiLiTE_HOME}\Scripts&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="Scripts"&gt;   &lt;Option key="Path"&gt;C:\MyDirectory&lt;/Option&gt;   &lt;Option key="PreLoad"&gt;MyScript&lt;/Option&gt; &lt;/OptionSet&gt; &lt;OptionSet name="Scripts"&gt;   &lt;Option key="Path"&gt;C:\MyDirectory&lt;/Option&gt;   &lt;Option key="PostLoad"&gt;MyScript&lt;/Option&gt; &lt;/OptionSet&gt;  &lt;OptionSet name="Scripts"&gt;   &lt;Option key="Path"&gt;     HiLiTE_HOME}\Scripts&lt;/Option&gt;   &lt;Option key="PreLoad"&gt;BusCreators&lt;/Option&gt; &lt;/OptionSet&gt; </pre>

Element Tag, Attribute Name, and/or OptionSet	Description	Example
<OptionSet name=" <b>TestGenLimits</b> "> Additional option key values	<i>MaxPreviousSteps</i> : (default=3) <i>MaxPrevValueOccurrences</i> : (default= 1) <i>MaxInfLoopRptCnt</i> : (default= 3)	

## Appendix I. Blocks with HiLiTE Built-In (Dynamic) Templates

For a number of HAM blocks, test case generation depends on values of non-static datapoints, block parameters, and other values that cannot be expressed using the Template Formula Language. For these blocks, the HiLiTE Template Manager shows blank templates and does not capture the test cases. The templates for these blocks are built-in – HiLiTE generates the test cases dynamically during test generation. Each test case that appears in the generated/built-in template denotes a test requirement of the same name. For example, if the template lists ten test cases, but HiLiTE is able to generate only five of these, then the requirements coverage for the block instance is 50%.

The following table lists the affected HAM blocks with their mask and block types. The remainder of this appendix details the blocks and the test cases that HiLiTE generates.

Note that test cases generated for these blocks cannot be modified/customized except where an option is listed in this appendix. You can manually add test cases to those generated by HiLiTE as necessary.

## Built-In (Dynamic) Templates

**Table 30. HAM blocks with HiLiTE Built-In (dynamic) templates**

HAM Block name	MaskType	HiLiTE BlockType	Limitations
1-D Look-Up Table	Look-Up Table (HW)	Table1D	None
Look-Up Table (2-D)	Look-Up Table 2-D (HW)	Table2D	None
Look-Up Table (3-D)	Look-Up Table 3-D (HW)	Table3D	None
Look-Up Table Pre-Index	Look-Up Table PreIndex (HW)	PreIndexTable	<ul style="list-style-type: none"> <li>• HiLiTE does not generate extrapolation test cases for this block.</li> <li>• Supports only the “Linear” Search Method and “Slow” Data Change Rate.</li> </ul>
Look-Up Table N-D	Look-Up Table N-D (HW)	NdTable1 (When number of dimensions: 1) NdTable2 (When number of dimensions: 2) NdTable3 (When number of dimensions: 3)	<ul style="list-style-type: none"> <li>• HiLiTE does not generate clipping, interpolation, and extrapolation test cases for nD look-up tables with one or two dimensions.</li> <li>• Does not generate any test cases for nD look-up tables with three dimensions.</li> </ul>
Vector Selector	Vector Selector (HW)	VectorSelector	None
Matrix Selector	Matrix Selector (HW)	MatrixSelector	None
Subsystem	Subsystem (HW)	GenericSubsystem	None
Inport	Inport (HW)	Inport (for generating robustness test cases)	None
Bus Selector	BusSelector (HW)	BusSelector	None
Test Point	Test Point (HW)	TestPoint	None
StimulateNumeric	Stimulate Numeric (HW)	StimulateNumeric	None
StimulateBoolean	Stimulate Boolean (HW)	StimulateBoolean	None
Future: vectorAssignment	Vector Assignment (HW)	VectorAssignment	None

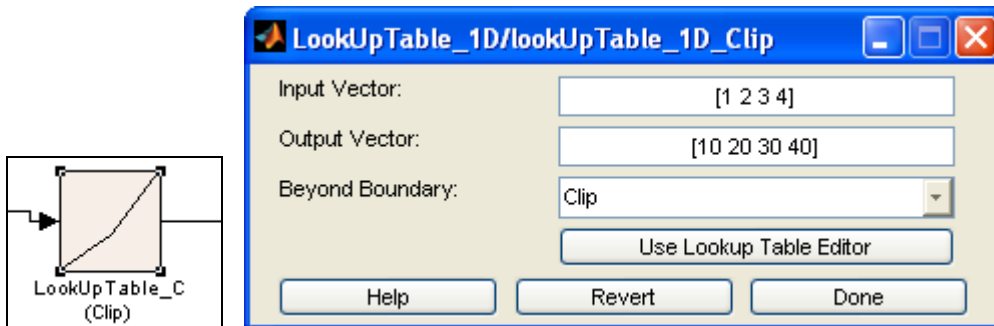
## Look-Up Tables

For lookup tables, HiLiTE generates normal data point, interpolation, and clipping or extrapolation test cases. Whether HiLiTE generates extrapolation or clipping type test cases depends on the value of the block's **Beyond Boundary** attribute. When the **Beyond Boundary** is set to **Clip**, HiLiTE will generate normal data point test cases, interpolation test cases, and clipping test cases. If the **Beyond Boundary** is set to **Extrapolate**, HiLiTE generates extrapolation test cases rather than clip test cases.

**Note about interpolation tests.** HiLiTE generates either all possible or some interpolation tests based upon command file options. To satisfy DO-178B objectives of requirements-based testing of the interpolation functionality, only one interpolation test case is needed for the entire table. Some projects, however, may institute stricter testing guidelines to create a test case for each possible interpolation point in the table. HiLiTE does provide an option to generate test cases for all possible interpolation points; but an error is raised for those interpolation points where the slope is flat – i.e., the end points of the table cell have the same value. Users have a choice to separately create manual test cases for those interpolation points when such error messages are given by HiLiTE.

### 1-D Look-Up Table

A 1-D Look-Up Table uses three datapoints, essentially a 1-row table.



#### Data Point test cases:

For lookup tables, HiLite will generate normal data point test cases regardless of constants/constraints at inputs. If test vectors are not generated, there will be a dip in coverage. HiLiTE will generate a test case for each data point:

- Data Point [20]
- Data Point [30]
- Data Point [40]

These test cases are shown in the dynamically generated templates shown in Table 31 and Table 32.

#### Interpolation test cases:

The interpolation point is the mid-point of the corresponding data points. In this case, for LookUpTable\_C block. The first interpolation point is 1.5 (mid point of 1 and 2) and the second interpolation point is 2.5 (mid point of 2 and 3).

The number of interpolation test cases generated depends on your HiLiTE command file elements. By default, HiLiTE will generate one interpolation test case for the interpolation point (input) between the first two data points. For full interpolation, include the **TestTemplates** optionset with an option key **LookUpTableAllInterpolationPoints** set to **True**.

```
<OptionSet name="TestTemplates">  
  <Option key="LookUpTableAllInterpolationPoints">True </Option>  
</OptionSet>
```

## Built-In (Dynamic) Templates

If LookUpTableAllInterpolationPoints set to True, HiLiTE will generate interpolation test cases between all data points.

Interpolation test cases for this look up table example are shown in Table 31 and Table 32.

**Beyond Boundary Test Cases****Clipping test cases:**

HiLiTE generates two clipping test cases: Clipping Below Limit and Clipping Above Limit. For LookUpTable\_C block, clipping test cases “Clipping Below Limit [0]” and “Clipping Above Limit [2]” will be generated. The value inside the square brackets denotes the index of the data point in the input vector.

For the Clipping Below Limit test case, an MDVD below the first data point in the input vector is selected. For LookUpTable\_C block, the MDVD for the first data point (-1) is -1.001.

For the Clipping Above Limit test case, an MDVD above the last data point in the input vector is selected. For LookUpTable\_C block, where the last data point is 1, this MDVD is 1.001. These test cases are shown in the dynamically generated template in Table 31.

**Table 31. Dynamically-generated test cases for Look-Up Table 1D-Clip, fully interpolated**

<b>Tests for Block: lookUpTable_1D_Clip</b>						
<b>Block Type: Table1D; Mask Type: Look-Up Table (HW)</b>						
<b>Test Case Templates (Standard): Data Point [0], Interpolation [0], Clipping Below Limit [0], Data Point [1], Interpolation [1], Data Point [2], Interpolation [2], Data Point [3], Clipping Above Limit [3]</b>						
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports		
				Normal Range	x1	y
				min → max →	0 100	10 40
Data Point [0]	Test table input for Data point [0]	Req. = Test Case	1		1	10
Interpolation [0]	Test table input for Interpolation	Req. = Test Case	1		1.5	15
Clipping Below Limit [0]	Test table input for clipping	Req. = Test Case	1		0.9999999	10
Data Point [1]	Test table input for Data point [1]	Req. = Test Case	1		2	20
Interpolation [1]	Test table input for Interpolation	Req. = Test Case	1		2.5	25
Data Point [2]	Test table input for Data point [2]	Req. = Test Case	1		3	30
Interpolation [2]	Test table input for Interpolation	Req. = Test Case	1		3.5	35
Data Point [3]	Test table input for Data point [3]	Req. = Test Case	1		4	40
Clipping Above Limit [3]	Test table input for clipping	Req. = Test Case	1		4.0000001	40

### Extrapolation test cases:

HiLiTE generates two extrapolation test cases: Extrapolation Below Min and Extrapolation Above Max. For LookUpTable\_E block, HiLiTE generates the test cases: “Extrapolation Below Min [0]” and “Extrapolation Above Max [2]”. The value inside the square brackets denotes the index of the data point in the input vector.

For Extrapolation Below Min, a value (inputVector[0] -1) below the first data point in the input vector is selected. For LookUpTable\_E block, the first data point is -1, so this value is -2.

For Extrapolation Above Max, a value (inputVector[inputVector.length -1] +1) above the last data point in the input vector is selected. For LookUpTable\_E block, the last data point is 1, so this value is 2..

Table 32 shows the extrapolation test cases generated for the example above.

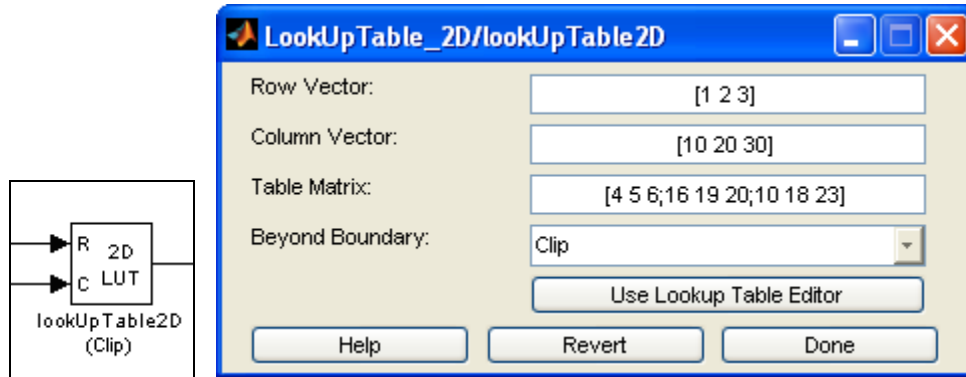
**Table 32. Dynamically-generated test cases for Look-Up Table 1D-Extrapolate, fully interpolated**

<b>Tests for Block: lookUpTable_1D_Extrap</b>						
<b>Block Type: Table1D; Mask Type: Look-Up Table (HW)</b>						
<b>Test Case Templates (Standard): Data Point [0], Interpolation [0], Exptrapolation Below Min[0], Data Point [1], Interpolation [1], Data Point [2], Interpolation [2], Data Point [3], Exptrapolation Above Max[3]</b>						
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports		
				Normal Range	x1	y
				min → max →	0 100	10 40
Data Point [0]	Test table input for Data point [0]	Req. = Test Case	1		1	10
Interpolation [0]	Test table input for Interpolation	Req. = Test Case	1		1.5	15
Exptrapolation Below Min[0]	Test table input for Exptrapolation	Req. = Test Case	1		0	0
Data Point [1]	Test table input for Data point [1]	Req. = Test Case	1		2	20
Interpolation [1]	Test table input for Interpolation	Req. = Test Case	1		2.5	25
Data Point [2]	Test table input for Data point [2]	Req. = Test Case	1		3	30
Interpolation [2]	Test table input for Interpolation	Req. = Test Case	1		3.5	35
Data Point [3]	Test table input for Data point [3]	Req. = Test Case	1		4	40
Exptrapolation Above Max[3]	Test table input for Exptrapolation	Req. = Test Case	1		5	50



## Look-Up Table (2-D)

Two-dimensional look up tables are three row, three column matrixes. For this HAM block, HiLiTE generates data point, interpolation, clipping, and extrapolation test cases. (Whether it generates clipping or extrapolation test cases depends on the **Beyond Boundary** value.)



### Data Point test cases:

In LookUpTable2D (Clip), **Row Vector** is specified as [1 2 3], **Column Vector** is specified as [10 20 30], and **Table Matrix** is specified with three rows and three columns. HiLiTE will generate nine data point test cases. These test cases are shown in the dynamically generated template in Table 33 and Table 34.

### Interpolation test cases:

The number of interpolation test cases generated depends on the **GenerateForAllInterpolationPoints** attribute as described on page 242.

When the **GenerateForAllInterpolationPoints** attribute is set to *false* for LookUpTable2D\_C block, HiLiTE will generate only the “Interpolation [0 0]” test case. The interpolation points for row and column is 1.5 (mid point of 1 and 2). These test cases are shown in the dynamically generated template in Table 33 and Table 34.

If **GenerateForAllInterpolationPoints** attribute is set to *true*, HiLiTE will generate interpolation test cases between all data points in rows and columns.

### Beyond Boundary Test Cases

#### Clipping test cases:

HiLiTE generates two clipping test cases: Clipping Below Limit and Clipping Above Limit. For LookUpTable2D\_C block, clipping test cases “Clipping Below Limit [0 0]” and “Clipping Above Limit [2 2]” will be generated. The values inside the square brackets denote the index of the row and column vector respectively.

For the Clipping Below Limit test case, an MDVD below the first data point in the row and column vectors is selected. For LookUpTable2D\_C block, the MDVD for the first data point in row and column (1) is 0.9999999.

For the Clipping Above Limit test case, the MDVD above the last data point in the row and column vectors is selected. For LookUpTable2D\_C block, where the last data point is 3, this MDVD this value3.00000001.

Standard test case templates shown in Table 33 are for a two-dimensional look up table with a **Beyond Boundary** of **Clip**.

**Table 33. Dynamically generated test cases for Look-Up Table 2-D (HW)  
with clipping, fully interpolated**

## Tests for Block: lookUpTable2D

Block Type: Table2D; Mask Type: Look-Up Table 2-D (HW)

Test Case Templates (Standard): Data Point [0 0], Interpolation [0 0], Clipping Below Limit [0 0], Data Point [0 1], Interpolation [0 1], Data Point [0 2], Data Point [1 0], Interpolation [1 0], Data Point [1 1], Interpolation [1 1], Data Point [1 2], Data Point [2 0], Data Point [2 1], Data Point [2 2], Clipping Above Limit [2 2]

Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Normal Range	Block Under Test Ports		
					x1	x2	y
					min →	0	4
				max →	5	50	23
Data Point [0 0]	Test table input for Data point [0 0]	Req. = Test Case	1		1	10	4
Interpolation [0 0]	Test table input for Interpolation	Req. = Test Case	1		1.5	15	11
Clipping Below Limit [0 0]	Test table input for clipping	Req. = Test Case	1		0.9999999	9.9999999	4
Data Point [0 1]	Test table input for Data point [0 1]	Req. = Test Case	1		1	20	5
Interpolation [0 1]	Test table input for Interpolation	Req. = Test Case	1		1.5	25	12.5
Data Point [0 2]	Test table input for Data point [0 2]	Req. = Test Case	1		1	30	6
Data Point [1 0]	Test table input for Data point [1 0]	Req. = Test Case	1		2	10	16
Interpolation [1 0]	Test table input for Interpolation	Req. = Test Case	1		2.5	15	15.75
Data Point [1 1]	Test table input for Data point [1 1]	Req. = Test Case	1		2	20	19
Interpolation [1 1]	Test table input for Interpolation	Req. = Test Case	1		2.5	25	20
Data Point [1 2]	Test table input for Data point [1 2]	Req. = Test Case	1		2	30	20
Data Point [2 0]	Test table input for Data point [2 0]	Req. = Test Case	1		3	10	10
Data Point [2 1]	Test table input for Data point [2 1]	Req. = Test Case	1		3	20	18
Data Point [2 2]	Test table input for Data point [2 2]	Req. = Test Case	1		3	30	23
Clipping Above Limit [2 2]	Test table input for clipping	Req. = Test Case	1		3.00000001	30.0000001	23

## Built-In (Dynamic) Templates

**Extrapolation test cases:**

HiLiTE generates two extrapolation test cases: Extrapolation Below Min and Extrapolation Above Max. For the LookUpTable2D\_E block, it generates extrapolation test cases “Extrapolation Below Min [0 0]” and “Extrapolation Above Max [2 2]”. The values inside the square brackets denote the index of the row and column vector respectively.

For Extrapolation Below Min, a row value ( $\text{rowVector}[0] - 1$ ) and column value ( $\text{colVector}[0] - 1$ ) will be selected. For LookUpTable2D\_E block, 0 is selected for row and column. The value (0) is below the first data point in row and column (1).

For Extrapolation Above Max, a row value ( $\text{rowVector}[\text{row.length}] - 1$ ) and column value ( $\text{colVector}[\text{col.length}] - 1$ ) will be selected. For LookUpTable2D1 block, the last data point is 3, so the selected value is 4.

**Test Case Templates**

Standard test case templates shown in Table 34 are those HiLiTE generates for a Look-Up Table 2-D (HW) with Beyond Boundary of Extrapolate and with full interpolation.

**Table 34. Dynamically generated test cases for Look-Up Table 2-D (HW)  
with extrapolation, fully interpolated**

## Tests for Block: lookUpTable2D1

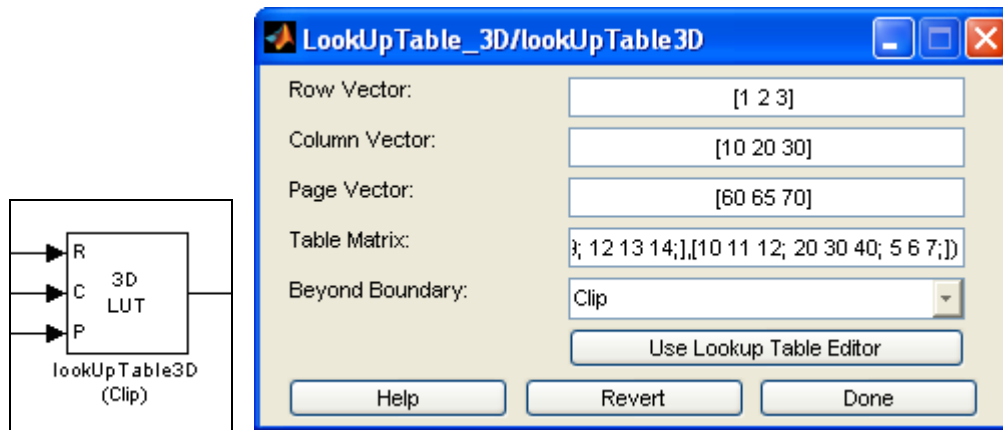
**Block Type: Table2D; Mask Type: Look-Up Table 2-D (HW)**

**Test Case Templates (Standard):** Data Point [0 0], Interpolation [0 0], Extrapolation Below Min[0 0], Data Point [0 1], Interpolation [0 1], Data Point [0 2], Data Point [1 0], Interpolation [1 0], Data Point [1 1], Interpolation [1 1], Data Point [1 2], Data Point [2 0], Data Point [2 1], Data Point [2 2], Extrapolation Above Max[2 2]

Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports			
				Normal Range	x1	x2	y
				min → max →	0 5	5 50	4 23
Data Point [0 0]	Test table input for Data point [0 0]	Req. = Test Case	1		1	10	4
Interpolation [0 0]	Test table input for Interpolation	Req. = Test Case	1		1.5	15	11
Extrapolation Below Min[0 0]	Test table input for Extrapolation	Req. = Test Case	1		0	9	83
Data Point [0 1]	Test table input for Data point [0 1]	Req. = Test Case	1		1	20	5
Interpolation [0 1]	Test table input for Interpolation	Req. = Test Case	1		1.5	25	12.5
Data Point [0 2]	Test table input for Data point [0 2]	Req. = Test Case	1		1	30	6
Data Point [1 0]	Test table input for Data point [1 0]	Req. = Test Case	1		2	10	16
Interpolation [1 0]	Test table input for Interpolation	Req. = Test Case	1		2.5	15	15.75
Data Point [1 1]	Test table input for Data point [1 1]	Req. = Test Case	1		2	20	19
Interpolation [1 1]	Test table input for Interpolation	Req. = Test Case	1		2.5	25	20
Data Point [1 2]	Test table input for Data point [1 2]	Req. = Test Case	1		2	30	20
Data Point [2 0]	Test table input for Data point [2 0]	Req. = Test Case	1		3	10	10
Data Point [2 1]	Test table input for Data point [2 1]	Req. = Test Case	1		3	20	18
Data Point [2 2]	Test table input for Data point [2 2]	Req. = Test Case	1		3	30	23
Extrapolation Above Max[2 2]	Test table input for Extrapolation	Req. = Test Case	1		4	31	-197.2

## Look-Up Table (3-D)

Look-Up Table (3-D) HAM blocks have three-dimensional matrices consisting of row, column, and page values. For this HAM block, HiLiTE generates data point, interpolation, clipping, and extrapolation test cases. (Whether it generates clipping or extrapolation test cases depends on the **Beyond Boundary** value.).



### Data Point test cases:

In LookUpTable3D\_C, Row Vector is [1 2 3], Column Vector is [10 20 30], and Page Vector is [60 65 70]. The Table Matrix has 3 rows, 3 columns, and 3 pages. HiLiTE will generate 27 data point test cases, some of which are shown in the dynamically-generated template in Table 35 and Table 36.

### Interpolation test cases:

The number of interpolation test cases generated depends on the **GenerateForAllInterpolationPoints** attribute as described on page 242.

When the **GenerateForAllInterpolationPoints** attribute is set to *false*, for the LookUpTable3D\_C block, HiLiTE will generate only the “Interpolation [0 0 0]” test case. The interpolation point for row, column, and page is 1.5 (mid point of 1 and 2). Some of these dynamically generated test cases are shown in Table 35 and Table 36.

If **GenerateForAllInterpolationPoints** attribute is set to *true*, HiLiTE will generate interpolation test cases between all data points in rows, columns, and pages.

### Beyond Boundary Test Cases

#### Clipping test cases:

HiLiTE generates two clipping test cases: Clipping Below Limit and Clipping Above Limit. For LookUpTable3D\_C, clipping test cases “Clipping Below Limit [0 0 0]” and “Clipping Above Limit [2 2 2]” will be generated. The values inside the square brackets denote the index of the row, column, and page vector respectively.

For the Clipping Below Limit test case, an MDVD below the first data point in the row, column, and page vectors are selected. For LookUpTable3D, the first data point in row, column, and page is 1, so the selected MDVD is 0.9999999.

For the Clipping Above Limit test case, an MDVD above the last data point in the row, column, and page vectors is selected. So, for LookUpTable3D block, the MDVD for this test case is 3.00000001 because the last data point in row, column, and page is 3.

Standard test case templates shown in Table 35 are for a Look-Up Table (3-D) block in which the Beyond Boundary value is Cl i p.

**Table 35. Dynamically generated test cases for Look-Up Table 3D (HW),  
with clipping, fully interpolated**

<b>Tests for Block: lookUpTable3D</b>								
<b>Block Type: Table3D; Mask Type: Look-Up Table 3-D (HW)</b>								
<b>Test Case Templates (Standard):</b> Data Point [0 0 0], Interpolation [0 0 0], Clipping Below Limit [0 0 0], Data Point [0 0 1], Interpolation [0 0 1], Data Point [0 0 2], Data Point [0 1 0], Interpolation [0 1 0], Data Point [0 1 1], Interpolation [0 1 1], Data Point [0 1 2], Data Point [0 2 0], Data Point [0 2 1], Data Point [0 2 2], Data Point [1 0 0], Interpolation [1 0 0], Data Point [1 0 1], Interpolation [1 0 1], Data Point [1 0 2], Data Point [1 1 0], Interpolation [1 1 0], Data Point [1 1 1], Interpolation [1 1 1], Data Point [1 1 2], Data Point [1 2 0], Data Point [1 2 1], Data Point [1 2 2], Data Point [2 0 0], Interpolation [2 0 0], Data Point [2 0 1], Interpolation [2 0 1], Data Point [2 0 2], Data Point [2 1 0], Interpolation [2 1 0], Data Point [2 1 1], Interpolation [2 1 1], Data Point [2 1 2], Data Point [2 2 0], Data Point [2 2 1], Data Point [2 2 2], Clipping Above Limit [2 2 2]								
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports				
				Normal Range	x2	x3	x1	y
				min → max →	0 5	5 25	-1000 1000	1 40
Data Point [0 0 0]	Test table input for Data point [0 0 0]	Req. = Test Case	1		1	10	60	4
Interpolation [0 0 0]	Test table input for Interpolation	Req. = Test Case	1		1.5	15	60	11
Clipping Below Limit [0 0 0]	Test table input for clipping	Req. = Test Case	1		0.9999999	9.9999999	59.999999	4
Data Point [0 0 1]	Test table input for Data point [0 0 1]	Req. = Test Case	1		1	20	60	5
Interpolation [0 0 1]	Test table input for Interpolation	Req. = Test Case	1		1.5	25	60	12.5
Data Point [0 0 2]	Test table input for Data point [0 0 2]	Req. = Test Case	1		1	30	60	6
Data Point [0 1 0]	Test table input for Data point [0 1 0]	Req. = Test Case	1		2	10	60	16
Interpolation [0 1 0]	Test table input for Interpolation	Req. = Test Case	1		2.5	15	60	15.75
Data Point [0 1 1]	Test table input for Data point [0 1 1]	Req. = Test Case	1		2	20	60	19
Interpolation [0 1 1]	Test table input for Interpolation	Req. = Test Case	1		2.5	25	60	20

## Built-In (Dynamic) Templates

**Extrapolation test cases:**

HiLiTE generates two extrapolation test cases: Extrapolation Below Min and Extrapolation Above Max. For the LookUpTable3D\_E block, it generates extrapolation test cases “Extrapolation Below Min [0 0 0]” and “Extrapolation Above Max [2 2 2]”. The values inside the square brackets denote the index of the row, column, and page vectors respectively.

For Extrapolation Below Min, a row value ( $\text{rowVector}[0] - 1$ ), column value ( $\text{colVector}[0] - 1$ ), and page value ( $\text{pageVector}[0] - 1$ ) will be selected. For LookUpTable3D\_E block, selected row, column, and page is 0, which is below the first data point in row, column, and page (1).

For Extrapolation Above Max, the selected row ( $\text{rowVector}[\text{row.length}] - 1$ ), column ( $\text{colVector}[\text{col.length}] - 1$ ) and page ( $\text{pageVector}[\text{page.length}] - 1$ ) values are selected. For LookUpTable3D\_E block, this value is 4, which is greater than the last data point in row, column, and page (3).

The example in Table 36 illustrates test case generation for Look-Up Table (3-D) block with a **Beyond Boundary of Extrapolate**.

**Table 36. Dynamically generated test cases for Look-Up Table 3D-Extrapolate**

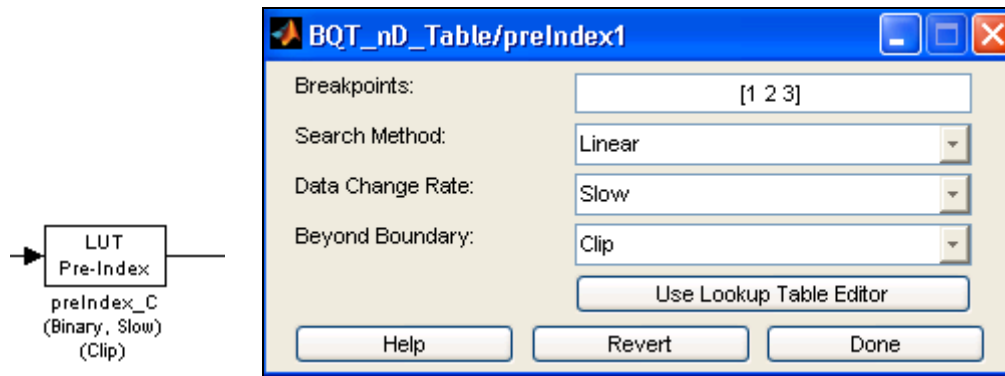
<b>Tests for Block: lookUpTable3D1</b>								
<b>Block Type: Table3D; Mask Type: Look-Up Table 3-D (HW)</b>								
<b>Test Case Templates (Standard):</b> Data Point [0 0 0], Interpolation [0 0 0], Extrapolation Below Min[0 0 0], Data Point [0 0 1], Interpolation [0 0 1], Data Point [0 0 2], Data Point [0 1 0], Interpolation [0 1 0], Data Point [0 1 1], Interpolation [0 1 1], Data Point [0 1 2], Data Point [0 2 0], Data Point [0 2 1], Data Point [0 2 2], Data Point [1 0 0], Interpolation [1 0 0], Data Point [1 0 1], Interpolation [1 0 1], Data Point [1 0 2], Data Point [1 1 0], Interpolation [1 1 0], Data Point [1 1 1], Interpolation [1 1 1], Data Point [1 1 2], Data Point [1 2 0], Data Point [1 2 1], Data Point [1 2 2], Data Point [2 0 0], Interpolation [2 0 0], Data Point [2 0 1], Interpolation [2 0 1], Data Point [2 1 0], Interpolation [2 1 0], Data Point [2 1 1], Interpolation [2 1 1], Data Point [2 1 2], Data Point [2 2 0], Data Point [2 2 1], Data Point [2 2 2], Extrapolation Above Max [2 2 2]								
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports				
				Normal Range	x2	x3	x1	y
				min → max →	0 100	5 25	-1000 1000	1 40
Data Point [0 0 0]	Test table input for Data point [0 0 0]	Req. = Test Case	1		1	10	60	4
Interpolation [0 0 0]	Test table input for Interpolation	Req. = Test Case	1		1.5	15	60	11
Extrapolation Below Min[0 0 0]	Test table input for Extrapolation	Req. = Test Case	1		0	59	9	2.52
Data Point [0 0 1]	Test table input for Data point [0 0 1]	Req. = Test Case	1		1	20	60	5
Interpolation [0 0 1]	Test table input for Interpolation	Req. = Test Case	1		1.5	25	60	12.5
Data Point [0 0 2]	Test table input for Data point [0 0 2]	Req. = Test Case	1		1	30	60	6
Data Point [0 1 0]	Test table input for Data point [0 1 0]	Req. = Test Case	1		2	10	60	16
Interpolation [0 1 0]	Test table input for Interpolation	Req. = Test Case	1		2.5	15	60	15.75
Data Point [0 1 1]	Test table input for Data point [0 1 1]	Req. = Test Case	1		2	20	60	19
Interpolation [0 1 1]	Test table input for Interpolation	Req. = Test Case	1		2.5	25	60	20
Data Point [0 1 2]	Test table input for Data point [0 1 2]	Req. = Test Case	1		2	30	60	20
Data Point [0 2 0]	Test table input for Data point [0 2 0]	Req. = Test Case	1		3	10	60	10
Data Point [0 2 1]	Test table input for Data point [0 2 1]	Req. = Test Case	1		3	20	60	10
Data Point [0 2 2]	Test table input for Data point [0 2 2]	Req. = Test Case	1		3	30	60	10



## Look-Up Table Pre-Index

For lookup table PreIndex, HiLiTE generates normal data point test cases, clipping test cases, and interpolation test cases. The type of test cases generated depends on the block's **Beyond Boundary**. By double clicking on the lookup table PreIndex block, you can set **Beyond Boundary** to either Clip or Extrapolate. When Clip is selected, HiLiTE will generate normal data point test cases, interpolation test cases, and clipping test cases.

Note that HiLiTE does *not* generate extrapolation test cases for this block. Also, note that it supports only the “Linear” Search Method and “Slow” Data Change Rate.



### Data Point test cases:

In Preindex1, Breakpoints has three data points, so HiLiTE will generate three data point test cases, shown in the dynamically generated template in Table 37.

### Interpolation test cases:

Unlike the previously described look-up table blocks, the number of interpolation test cases generated for preIndex1 block does *not* depend on the TestTemplate optionset with an option key LookUpTableAllInterpolationPoints set to True, as described on page 242.

By default, HiLiTE will generate interpolation test cases between all data points in the Breakpoints vector. For preIndex1 block, “Interpolation [1]” and “Interpolation [2]” test cases will be generated. The first interpolation point (input) is selected as 1.5 (mid point of 1 and 2) and the second interpolation point (input) is selected as 2.5 (mid point of 2 and 3).

### Beyond Boundary Test Cases

#### Clipping test cases:

HiLiTE generates two clipping test cases: Clipping Below Limit and Clipping Above Limit. For preIndex1, HiLiTE will generate clipping test cases “Clipping Below Limit [0]” and “Clipping Above Limit [2]”. The value inside the square brackets denotes the index of the Breakpoints vector.

For Clipping Below Limit test cases, an MDVD below the first data point in the input vector is selected. For preIndex1 block, the MDVD for the first data point (1) is 0.999.

For Clipping Above Limit test cases, an MDVD above the last data point in the input vector is selected. For preIndex1 block, where the last data point is 3, this MDVD this value3.001. Table 37 shows these test cases in the dynamically generated template.

### Test Case Templates

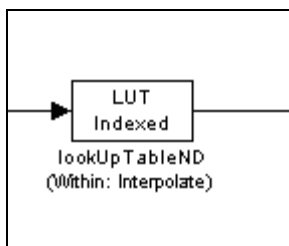
Standard test case templates shown in Table 37. DataPoint[0], DataPoint[1], DataPoint[2], Clipping Test case[0], Clipping Test case[2], Interpolation test case[1], Interpolation test case[2]

**Table 37. Dynamically generated test cases for Look-Up Table preIndex1**

<b>Tests for Block: preIndex_C</b>						
<b>Block Type: PreIndexTable; Mask Type: Look-Up Table PreIndex (HW)</b>						
<b>Test Case Templates (Standard): DataPoint[0], DataPoint[1], DataPoint[2], Clipping TestCase[0], Clipping TestCase[2], Interpolation testcase[1], Interpolation testcase[2]</b>						
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports		
				Normal Range	Input	Output
				min →	0	0
				max →	100	1
DataPoint[0]	Test pre-index table input data point[0]	Req. = Test Case	1		1	0
DataPoint[1]	Test pre-index table input data point[1]	Req. = Test Case	1		2	1
DataPoint[2]	Test pre-index table input data point[2]	Req. = Test Case	1		3	1
Clipping TestCase[0]	Test pre-index table for Clipping [0]	Req. = Test Case	1		0.9999999	0
Clipping TestCase[2]	Test pre-index table for Clipping [2]	Req. = Test Case	1		3.00000001	1
Interpolation testcase[1]	Test for pre-index table Interpolation[1]	Req. = Test Case	1		1.5	0.5
Interpolation testcase[2]	Test for pre-index table Interpolation[2]	Req. = Test Case	1		2.5	1

## Look-Up Table N-D (HW)–1D

For lookup table Look-Up Table N-D (1 dimension), HiLiTE generates only normal data point test cases. Note that HiLiTE does not generate clipping, interpolation, and extrapolation test cases for nD look-up tables with 1 dimension. These cases need to be added manually to the HiLiTE-generated test cases.



**BQT\_nD\_Table/nDTable1\_LUT**

Number of Dimensions: 1

Data: [7 20 30]

Within Boundary: Interpolate

Use Lookup Table Editor

Help Revert Done

## Built-In (Dynamic) Templates

**Data Point test cases:**

In nDTable1\_LUT, the Data vector has three data points, so HiLiTE will generate three data point test cases, shown in the dynamically generated template in Table 38. Note that Look-Up Table N-D (HW) always gets its input from preIndex table (index and fraction).

**Test Case Templates:**

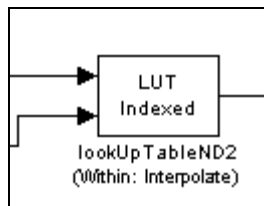
Table 38 shows standard test case templates for Look-Up Table N-D (HW)-1D.

**Table 38. Dynamically generated test cases for Look-Up Table N-D (HW)-1D**

<b>Tests for Block: lookUpTableND</b>						
<b>Block Type: NdTable1; Mask Type: Look-Up Table N-D (HW)</b>						
<b>Test Case Templates (Standard): DataPoint[0], DataPoint[1], DataPoint [2]</b>						
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports		
				Normal Range	Input1	Output
				min →	0	7
				max →	100	7
DataPoint[0]	Test nd-table input data point[0]	Req. = Test Case	1		0	7
DataPoint[1]	Test nd-table input data point[1]	Req. = Test Case	1		1	20
DataPoint[2]	Test nd-table input data point[2]	Req. = Test Case	1		1	30

**Look-Up Table N-D (HW)-2D**

For lookup table Look-Up Table N-D (two dimensions), HiLiTE generates only normal data point test cases. Note that HiLiTE does not generate clipping, interpolation, and extrapolation test cases for nD look-up tables with two dimensions. These cases need to be added manually to the HiLiTE-generated test cases.



### Data Point test cases:

In nDTable2\_LUT, the Data vector has data points in three rows and three columns, so HiLiTE will generate nine data point test cases. These test cases are shown in the dynamically generated template in Table 39.

Note that Look-Up Table N-D (HW) always gets the input from a preIndex table (index and fraction).

**Table 39. Dynamically generated test cases for Look-Up Table N-D (HW)-2D**

<b>Tests for Block: lookUpTable2D</b>							
<b>Block Type: Table2D; Mask Type: Look-Up Table 2-D (HW)</b>							
<b>Test Case Templates (Standard): Data Point [0 0], Interpolation [0 0], Clipping Below Limit [0 0], Data Point [0 1], Data Point [0 2], Data Point [1 0], Data Point [1 1], Data Point [1 2], Data Point [2 0], Data Point [2 1], Data Point [2 2], Clipping Above Limit [2 2]</b>							
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports			
				Normal Range	x1	x2	y
				min → max →	0 5	0 50	4 23
Data Point [0 0]	Test table input for Data point [0 0]	Req. = Test Case	1		1	10	4
Interpolation [0 0]	Test table input for Interpolation	Req. = Test Case	1		1.5	15	11
Clipping Below Limit [0 0]	Test table input for clipping	Req. = Test Case	1		0.9999999	9.9999999	4
Data Point [0 1]	Test table input for Data point [0 1]	Req. = Test Case	1		1	20	5
Data Point [0 2]	Test table input for Data point [0 2]	Req. = Test Case	1		1	30	6
Data Point [1 0]	Test table input for Data point [1 0]	Req. = Test Case	1		2	10	16
Data Point [1 1]	Test table input for Data point [1 1]	Req. = Test Case	1		2	20	19
Data Point [1 2]	Test table input for Data point [1 2]	Req. = Test Case	1		2	30	20
Data Point [2 0]	Test table input for Data point [2 0]	Req. = Test Case	1		3	10	10
Data Point [2 1]	Test table input for Data point [2 1]	Req. = Test Case	1		3	20	18
Data Point [2 2]	Test table input for Data point [2 2]	Req. = Test Case	1		3	30	23
Clipping Above Limit [2 2]	Test table input for clipping	Req. = Test Case	1		3.0000001	30.000001	23

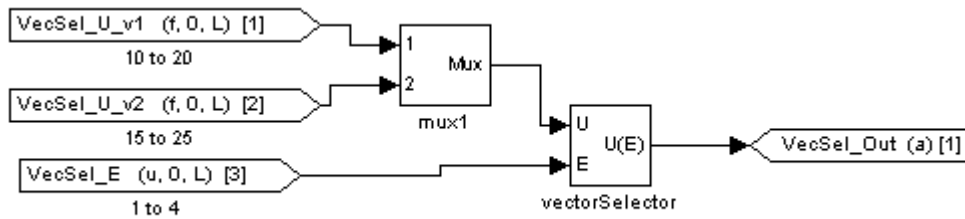
### Look-Up Table N-D (HW)-3D

HiLiTE does not currently support generation of test cases for Look-Up Table N-D (HW) with three dimensions.

## Built-In (Dynamic) Templates

**Vector Selector (HW)**

For the Vector Selector (HW) block, HiLiTE generates one test case for each signal at U(E) (Output) port.



For the block VS\_E\_Scalar, above, the dimension at output port U(E) is 2, so HiLiTE will generate two test cases as shown in Table 40. Note that HiLiTE uses 0-based indexes when generating the test case name.

**Test Case Templates**

Table 40 shows the standard test case templates.

**Table 40. Dynamically generated test cases for Vector Selector (HW)**

## Tests for Block: vectorSelector

**Block Type: VectorSelector; Mask Type: Vector Selector (HW)**

**Test Case Templates (Standard):** Port: vectorSelector.O1: Output Signal Number [0] to Port: vectorSelector.I1: Signal Number [0], Port: vectorSelector.O1: Output Signal Number [0] to Port: vectorSelector.I1: Signal Number [1]

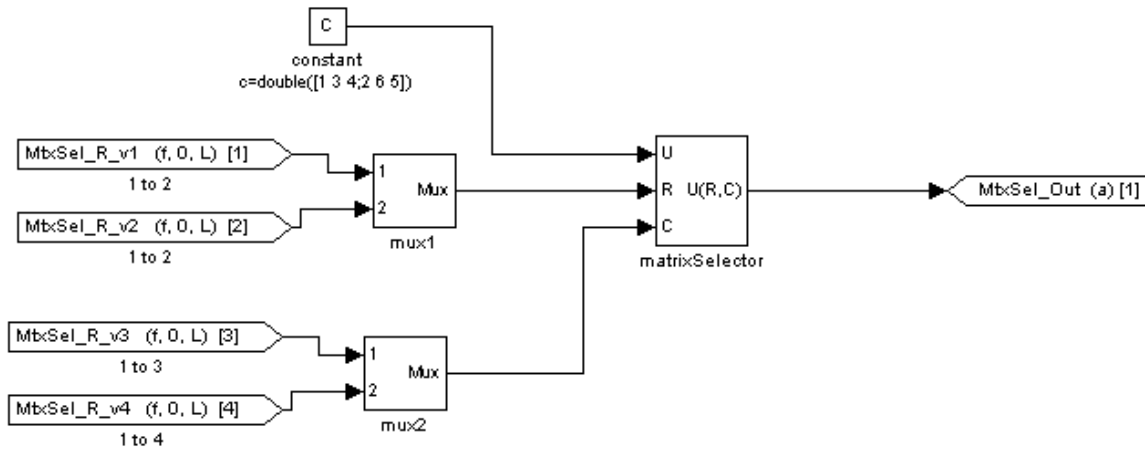
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports			
				Normal Range	Data	IndexPort	Output
				min →	10 15	1	10
				max →	20 25	4	25
Port: vectorSelector.O1: Output Signal Number [0] to Port: vectorSelector.I1: Signal Number [0]	missing in the template	Req. = Test Case	1		25	1	25
Port: vectorSelector.O1: Output Signal Number [0] to Port: vectorSelector.I1: Signal Number [1]	missing in the template	Req. = Test Case	1		25	2	25

Note: “missing in the template” under Test Case Description will be replaced with descriptive text in a later release.

## Matrix Selector (HW)

For Matrix Selector (HW) block, HiLiTE generates one test case for each signal at output port.

## Built-In (Dynamic) Templates



For the block `matrixSelector`, above, the dimension at output port is 4, so HiLiTE generates four test cases as shown in Table 41. Note that HiLiTE uses 0-based indexes when generating the test case name.

**Table 41. Dynamically generated test cases for Matrix Selector (HW)**

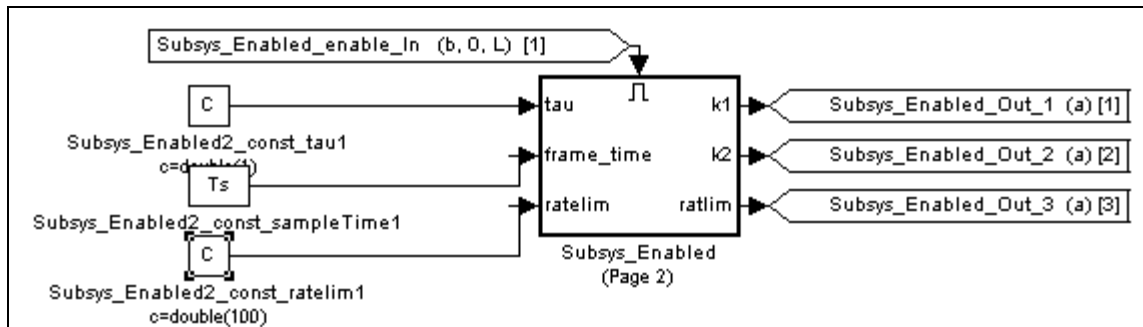
<b>Tests for Block: matrixSelector</b>								
<b>Block Type: MatrixSelector; Mask Type: Matrix Selector (HW)</b>								
<b>Test Case Templates (Standard):</b> Port: matrixSelector.O1: Output Signal Number [0] to Port: matrixSelector.I1: Signal Number [1][1], Port: matrixSelector.O1: Output Signal Number [1] to Port: matrixSelector.I1: Signal Number [1][1], Port: matrixSelector.O1: Output Signal Number [2] to Port: matrixSelector.I1: Signal Number [1][1], Port: matrixSelector.O1: Output Signal Number [3] to Port: matrixSelector.I1: Signal Number [1][1]								
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports				
				Normal Range	Data	IndexPort[	IndexPort[	Output
				min →	[1 3 4] [2 6 5]	1 1	1 1	1 1 1 1
				max →	[1 3 4] [2 6 5]	2 2	3 4	6 6 6 6
Port: matrixSelector.O1: Output Signal Number [0] to Port: matrixSelector.I1: Signal Number [1][1]	missing in the template	Req. = Test Case	1		6	2	2	6
Port: matrixSelector.O1: Output Signal Number [1] to Port: matrixSelector.I1: Signal Number [1][1]	missing in the template	Req. = Test Case	1		6	2	2	6
Port: matrixSelector.O1: Output Signal Number [2] to Port: matrixSelector.I1: Signal Number [1][1]	missing in the template	Req. = Test Case	1		6	2	2	6
Port: matrixSelector.O1: Output Signal Number [3] to Port: matrixSelector.I1: Signal Number [1][1]	missing in the template	Req. = Test Case	1		6	2	2	6

Note: “missing in the template” under Test Case Description will be replaced with descriptive text in a later release.



## Enable Subsystem (Nested Model)

For the Subsystem (HW) block, HiLiTE generates two kinds of test cases (Initial Condition and Reset/Hold) for each Output Block inside the enable subsystem



### Initial Condition

For each Output Block inside the Enable subsystem, an initial output value is specified for the block attribute `Initial Output` as shown below.

The Initial Condition test case will set the enable port value to false and expect the initial condition values at output.

### Reset/Held

The value selected for the block attribute `Output when Disabled` determines the test cases that HiLiTE generates.

The **Held** test case is generated in two steps:

1. The *Held* value will be set at the output block inside the Enable subsystem.
2. The Enable port will be set to *false* and the held value set in Step 1 will be expected at the output block.

The **Reset** test cases is also generated in two steps:

1. The *Held* value will be set at the output block inside the Enable subsystem.
2. The Enable port will be set to *false* and initial output value is expected at output block.

### Tests for Block: Subsys\_Enabled

Block Type: GenericSubsystem; Mask Type: Subsystem (HW)

### Test Case Templates

Standard test case templates shown in Table 42.

- Initial Condition Test: Output Block Subsys\_Enabled/k1 Signal[0],
- Held test for Output Block Subsys\_Enabled/k1 Signal[0],
- Initial Condition Test: Output Block Subsys\_Enabled/k2 Signal[0],
- Reset test for Output Block Subsys\_Enabled/k2 Signal[0],
- Initial Condition Test: Output Block Subsys\_Enabled/ratlim Signal[0],
- Held test for Output Block Subsys\_Enabled/ratlim Signal[0]

**Table 42. Dynamically generated test cases for Enable Subsystem (Nested Model)**

signal name			Subsys_Enabled~t_tau1	Subsys_Enabled~ble_In	Subsys_Enabled~eTime1	Subsys_Enabled~telim1	Subsys_Enabled~Out_1	Subsys_Enabled~Out_2	Subsys_Enabled~Out_3
Test Case Name	Time Steps	range	Input[1]	Enable[1]	Input[2]	Input[3]	Output[1]	Output[2]	Output[3]
		min→	1	0	0.05	100	1	0.05	100
		max→	1	1	0.05	100	1	0.05	100
Initial Condition Test: Output Block Subsys_Enabled/k1 Signal[0]	1		X	0	X	X	1	X	X
held test for Output Block Subsys_Enabled/k1 Signal[0]	1		1	1	X	X	1	X	X
	1		1	0	X	X	1	X	X
Initial Condition Test: Output Block Subsys_Enabled/k2 Signal[0]	1		X	0	X	X	X	0.05000000	X
reset test for Output Block Subsys_Enabled/k2 Signal[0]	1		X	1	0.05000000	X	X	0.05000000	X
	1		X	0	0.05000000	X	X	0.05000000	X
Initial Condition Test: Output Block Subsys_Enabled/ratlim Signal[0]	1		X	0	X	X	X	X	100
held test for Output Block Subsys_Enabled/ratlim Signal[0]	1		X	1	X	100	X	X	100
	1		X	0	X	100	X	X	100

## Built-In (Dynamic) Templates

**Inport (HW)**

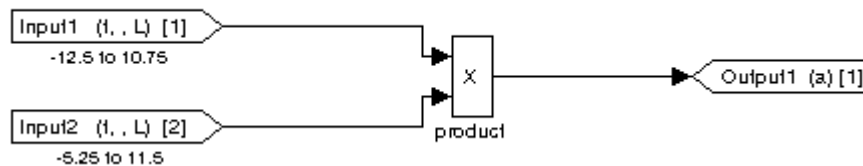
When *ModelInputsRobustnessCoverage* option is turned “On” in the command file HiLiTE performs robustness coverage analysis of model inputs. Each model input must be used for a value within, below, and above its normal operating range in at least one test vector generated. If a specific robustness value is missing for an input, HiLiTE attempts to generate additional robustness tests for that input. HiLiTE generates three robustness test cases for each input block:

Above Normal Op Range Max

Within Op Range Max and Min

Below Normal Op Range Min

An operating range for each Inport block must be specified in either the block properties (Lower Range and Upper Range ) or in a range file. If the range is not specified in either place, HiLiTE will use a default range.



For the block Input1, HiLiTE generates two test cases, as shown in Table 43.

**Table 43. Dynamically generated test cases for Inport (HW)**

### Tests for Block: Input1

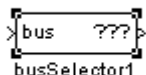
Block Type: Inport; Mask Type: Inport (HW)

Test Case Templates (Standard): At or Above Op Range Max, At or Below Op Range Min

Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports	
				Normal Range	Output
				min →	-12.5
				max →	10.75
At or Above Op Range Max	Tests Robustness of Model Input Value At or Above Max of operating range	Req. = Test Case	1		10.750001
At or Below Op Range Min	Tests Robustness of model input value At or Below Min of operating range	Req. = Test Case	1		-12.500001

## Bus Selector (HW)

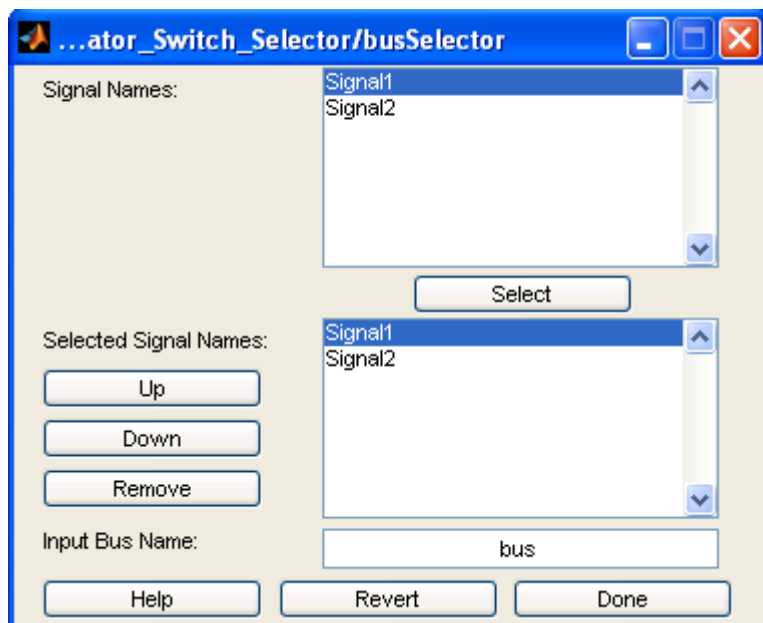
For the Bus Selector (HW) block, HiLiTE generates one test case for each selected signal from the input port.



Input to the bus selector block must be connected to the output of a Bus Creator block

### Selecting signals from input

In the window that opens when you double click on Bus Selector block, select signals from the Signal Names list. Click on the Select button to add them to the Selected Signal Names list. You can reorder your selections using the Up/Down/Remove buttons next to the Selected Signal Names list.



### Generated Test cases:

1. Output Port 1 signal Signal1 from Input Port signal Signal1 – Test cases for testing the signal selected Signal1.
2. Output Port 2 signal Signal2 from Input Port signal Signal2 - Test cases for testing the signal selected Signal2.

Table 44 shows dynamic test case templates for the Bus Selector Block.

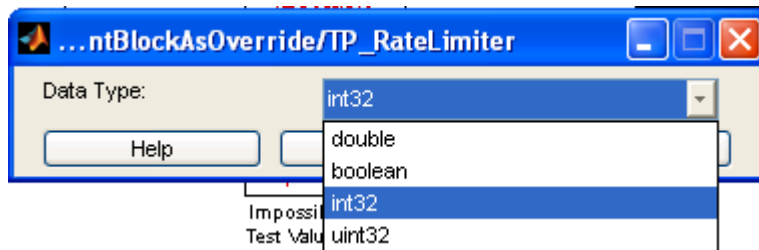
**Table 44. Dynamically generated test cases for a Bus Selector block**

<b>Tests for Block: BusSelectorCascade1</b>							
<b>Block Type: BusSelector; Mask Type: Bus Selector (HW)</b>							
<b>Test Case Templates (Standard): Output Port 1 signal Signal1 from Input Port signal Signal1, Output Port 2 signal Signal2 from Input Port signal Signal2</b>							
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Block Under Test Ports			
				Normal Range	Input	Output[1]	Output[2]
				min →	1 -10	1	-10
				max →	23 13	23	13
Output Port 1 signal Signal1 from Input Port signal Signal1	Test for normal operation	Req. = Test Case	1		23	23	X
Output Port 2 signal Signal2 from Input Port signal Signal2	Test for normal operation	Req. = Test Case	1		13	X	13

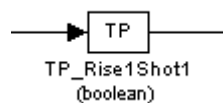
## Test Point (HW)

Test Point blocks work as secondary stimuli if the input testEnable value is True, else they are pass throughs. The stimulus is applied by overwriting externally (privately) visible global variables in the code that represent constants inside the Test point block., HiLiTE will locate the testgain and testbias external variables in the model source and set that variable's value to create a secondary stimulus.

The input accepted by a Test Point block is dependent on the data type selected in the MATLAB model. From the window opened by double-clicking on the Test Point block, select the appropriate data type, and click on the Done button.



## Generated Test cases for Boolean Test Points



HiLiTE generates two test vectors for Boolean Test Points:

1. testEnable set to false – when testEnable is set to false the block works as pass through, this vectors test pass through functionality, here testEnable is set to false and input is set to value and same value is expected at output.

- testEnable set to true - when testEnable is set to true the block works as secondary stimulus, this vectors test secondary stimulus functionality, here testEnable is set to true and input is set to don't care and testgain, testbias are set to value and same value is expected at output.

Table 45 shows Dynamic test case templates of a Test Point Boolean block.

**Table 45. Dynamically generated test cases for a Test Point Boolean block**

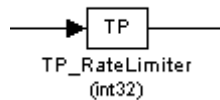
Tests for Block: TP\_Rise1Shot1

Block Type: TestPoint; Mask Type: Test Point (HW)

Test Case Templates (Standard): Op Max Signal Number [0], Op Min Signal Number [0]

					Block Under Test Ports			
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Normal Range	Input	OverrideGa	OverrideBi	Output
				min →	0	0	0	0
				max →	1	0	1	1
Op Max Signal Number [0]	testEnable set to false	Req. = Test Case	1		1	X	X	1
Op Min Signal Number [0]	testEnable set to true	Req. = Test Case	1		X	0	0	0

### Generated Test cases for numeric Test Points



HiLiTE generates two test vectors when the selected data type is double/int32/uint32:

- testEnable set to false – when testEnable is set to false the block works as pass through, this vectors test pass through functionality, here testEnable is set to false and input is set to value and same value is expected at output.
- testEnable set to true - when testEnable is set to true the block works as secondary stimulus, this vectors test secondary stimulus functionality, here testEnable is set to true and input is set to don't care and testgain, testbias are set to value and same value is expected at output.

Table 46 shows Dynamic test case templates for a Test Point Numeric block.

**Table 46. Dynamically generated test cases for a Test Point Numeric block**

<b>Tests for Block: TP_RateLimiter</b>								
<b>Block Type: TestPoint; Mask Type: Test Point (HW)</b>								
<b>Test Case Templates (Standard): Op Max Signal Number [0], Op Min Signal Number [0]</b>								
Test Case Name	Test Case Description	Test Requirements Covered	No. of Time Steps	Normal Range min → max →	Block Under Test Ports			
					Input	OverrideGa	OverrideBi	Output
					-20	0	-20	-20
Op Max Signal Number [0]	testEnable set to false	Req. = Test Case	1	32	0	32	32	32
Op Min Signal Number [0]	testEnable set to true	Req. = Test Case	1	X	0	-20	-20	-20

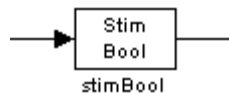
## Stimulation Blocks

HiLiTE generates dynamic tests for two types of stimulation blocks:

1. Stimulate Boolean (HW)
2. Stimulate Numeric (HW)

### Stimulate Boolean (HW)

A Stimulate Boolean block can be used as a secondary stimulus or as a pass through.



If DisconnectPort is False the block works as a secondary stimulus; else, it works as a pass through.

HiLiTE generates test cases to test secondary stimulus and pass through using operating range min and operating range max values, also test cases are generated to test secondary stimulus when input is set operating rang min and operating range max.

#### Generated test cases

1. Override Input at Op Max – Test case test for testing Override functionality by setting OverrideInputPort to Op max and input port to Don't care and DisconnectPort to 0;.
2. Override Input at Op Min – Test case test for testing Override functionality by setting OverrideInputPort to Op Min and input port to Don't care and DisconnectPort to 0;.
3. Upstream Input at Op Max & pass thru to output - Test case test for testing Pass Through functionality by setting InputPort to Op Max and OverrideInputPort to 0.
4. Upstream Input at Op Min & pass thru to output - Test case test for testing Pass Through functionality by setting InputPort to Op Min and OverrideInputPort to 0.
5. Override Partially Active with Upstream Input = 1 - Test case test for testing Override functionality by setting OverrideInputPort to Op Min and input port to 1 and DisconnectPort to 0.

6. Upstream Input at Op Max and Override Input at Op Max - Test case test for testing Override functionality by setting OverrideInputPort to Op Max and input port to 0 and DisconnectPort to 1.

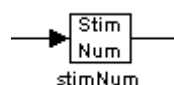
Table 47 shows the dynamic test case templates for a Stimulate Boolean block.

**Table 47. Dynamically generated test cases for a Stimulate Boolean block**

<b>Tests for Block: stimBool</b>							
<b>Block Type: StimulateBoolean; Mask Type: Stimulate Boolean (HW)</b>							
<b>Test Case Templates (Standard): Override Input at Op Max, Override Input at Op Min, Upstream Input at Op Max &amp; pass thru to output, Upstream Input at Op Min &amp; pass thru to output, Override Partially Active with Upstream Input = 1, Upstream Input at Op Max and Override Input at Op Max</b>							
Test Case Name	Test Case Description	No. of Time Steps	Block Under Test Ports				
			Normal Range	Input	Disconnect	OverrideIn	Output
			min →	0	0	0	0
			max →	1	1	1	1
Override Input at Op Max	test override capability - set override input at max value of input operating range and expect it at output	1		X	0	1	1
Override Input at Op Min	test override capability - set override input at min value of input operating range and expect it at output	1		X	0	0	0
Upstream Input at Op Max & pass thru to output	test signal pass thru capability - disable override, set max range at input and expect it at output	1		1	1	0	1
Upstream Input at Op Min & pass thru to output	test signal pass thru capability - disable override, set min range at input and expect it at output	1		0	1	0	0
Override Partially Active with Upstream Input = 1	Set up upstream input and partial override - test case for internal design coverage, if needed	1		1	0	0	0
Upstream Input at Op Max and Override Input at Op Max	test override capability - set input and override input at max value of operating range and expect it at output	1		1	0	1	1

## Stimulate Numeric (HW)

A Stimulate Numeric block can be used as a secondary stimulus or as a pass through.



If DisconnectPort is False, then the block works as a secondary stimulus; else, it works as a pass through.

HiLiTE generates test cases to test secondary stimulus and pass through using operating range min and operating range max values.



## Built-In (Dynamic) Templates

**Generated test cases**

1. Override Input at Op Max - Test case test for testing Override functionality by setting OverrideInputPort to Op max and input port to Don't care and DisconnectPort to 0.
2. Override Input at Op Min - Test case test for testing Override functionality by setting OverrideInputPort to Op Min and input port to Don't care and DisconnectPort to 0.
3. Upstream Input at Op Max & pass thru to output - - Test case test for testing Pass Through functionality by setting InputPort to Op Max and OverrideInputPort to 0.
4. Upstream Input at Op Min & pass thru to output - Test case test for testing Pass Through functionality by setting InputPort to Op Min and OverrideInputPort to 0.

Table 48 shows the dynamic test case templates for a Stimulate Numeric block.

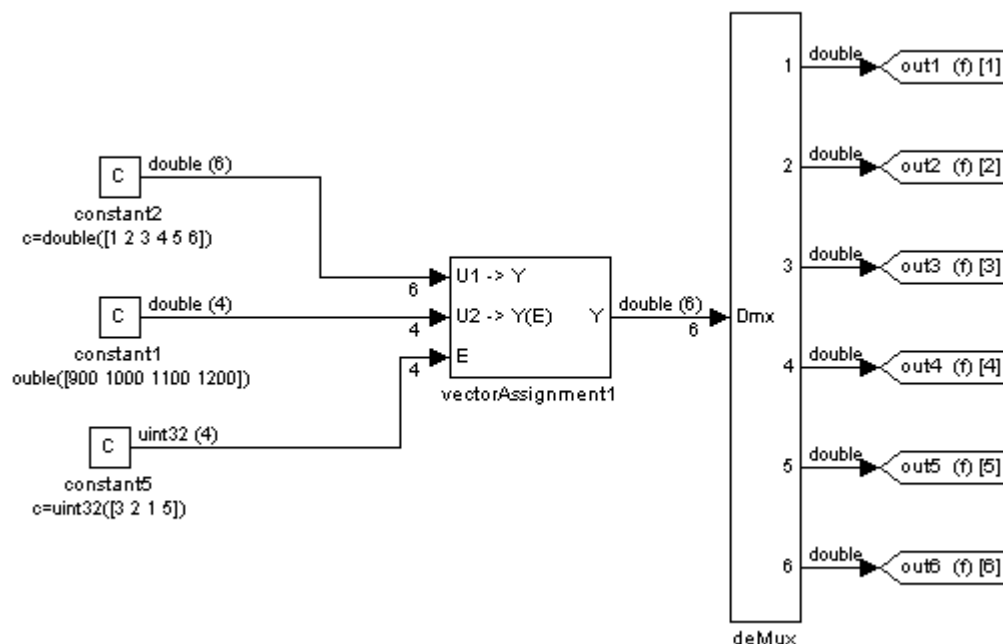
**Table 48. Dynamically generated test cases for a Stimulate Numeric Block**

<b>Tests for Block: stimNum</b>							
<b>Block Type: StimulateNumeric; Mask Type: Stimulate Numeric (HW)</b>							
<b>Test Case Templates (Standard): Override Input at Op Max, Override Input at Op Min, Upstream Input at Op Max &amp; pass thru to output, Upstream Input at Op Min &amp; pass thru to output</b>							
Test Case Name	Test Case Description	No. of Time Steps	Block Under Test Ports				
			Normal Range	Input	Disconnect	OverrideIn	Output
			min □	16	0	16	16
			max □	144	1	144	144
Override Input at Op Max	test override capability - set override input at max value of input operating range and expect it at output	1		X	0	144	144
Override Input at Op Min	test override capability - set override input at min value of input operating range and expect it at output	1		X	0	16	16
Upstream Input at Op Max & pass thru to output	test signal pass thru capability - disable override, set max range at input and expect it at output	1		144	1	0	144
Upstream Input at Op Min & pass thru to output	test signal pass thru capability - disable override, set min range at input and expect it at output	1		16	1	0	16

**Vector Assignment (HW)**

For the Vector Assignment (HW) block, HiLiTE generates one test case for each output signal at the Y (output) port.

# Appendix I. Blocks with HiLiTE Built-In (Dynamic) Templates



HiLiTE gives priority to signals coming into the **U2** port to drive the output by setting appropriate values at the **E** port. If the **U2** port cannot drive the output, then HiLiTE will select the corresponding signal at the **U1** port to drive the output.

HiLiTE uses a zero-based index while generating the test case name to select a particular signal of the vector. For example, the test “**U2 port signal: [2] Driving The Output**” means that HiLiTE has attempted to generate the test case where the value at the 3<sup>rd</sup> signal of the **U2** port will be at the output port. Also note proper values are placed at all (not only 3<sup>rd</sup> signal ) signals at the **E** port.

## Built-In (Dynamic) Templates

## Generated Test Cases

Table 49. Dynamically generated test cases for Vector Assignment (HW) block

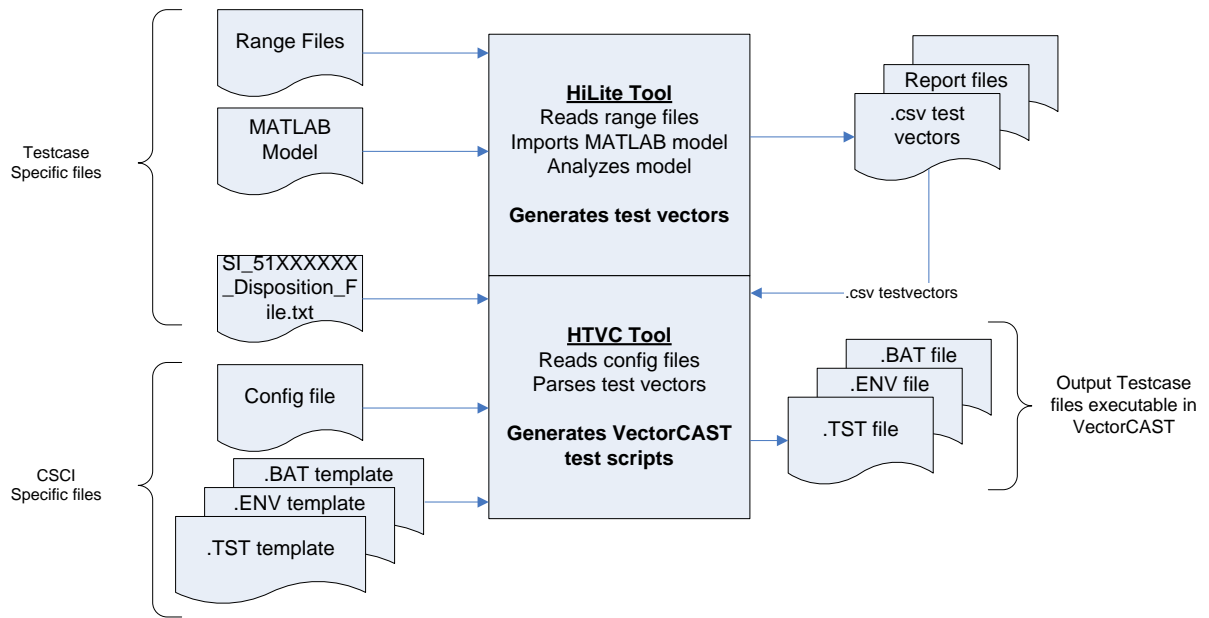
signal name →			constant2	constant1	constant5	vectorAssignment1.VectorAssign_1_TPS
Test Case Name	Time Steps	range	U1	U2	E	Y
		min →	1			1100
			2	900	3	1000
			3	1000	2	900
			4	1100	1	4
			5	1200	5	1200
			6			6
		max →	1			1100
			2	900	3	1000
			3	1000	2	900
			4	1100	1	4
			5	1200	5	1200
			6			6
U2 port signal: [2] Driving The Output	1		X	X	3	1100 X X X X X
U2 port signal: [1] Driving The Output	1	X	X 1000 X X	3 2 1 5	X 1000 X X X X	
U2 port signal: [0] Driving The Output	1	X	900 X X X	3 2 1 5	X X 900 X X X	
U1 port signal: [3] Driving The Output	1	X X X 4 X X	X	3 2 1 5	X X X 4 X X	
U2 port signal: [3] Driving The Output	1	X	X X X 1200	3 2 1 5	X X X X 1200 X	
U1 port signal: [5] Driving The Output	1	X X X X X 6	X	3 2 1 5	X X X X X 6	

## **Appendix J. Generating Test Vector Output for VectorCast Platform**

VectorCAST is a third party tool that supports structural coverage analysis and requirements based verification at the target level. The HTVC Tool generates the VectorCAST executable script file using the HiLiTE test vectors (in the \*.CSV file).

The HTVC (HiLiTE Test Vectors to VectorCAST conversion) tool is integrated with HiLiTE to support the auto-generation of VectorCAST script files. These VectorCAST scripts will be executed on the model implementation object code at the target level to ensure that the executable object code satisfies the low level requirements defined for that model. The HiLiTE tool in conjunction with the HTVC tool thus provides a complete end-to-end software verification solution whereby users can verify their Simulink model initially at the model level and then at the target level by executing the auto-generated VectorCAST script files on the model implementation object code. Figure 122 illustrates the conversion process.

## Built-In (Dynamic) Templates



**Figure 122 - MATLAB Model to VectorCAST executable Test Case Conversion Process**

## Testcase Specific files

- Range Files - Range file is the input to HiLiTE tool, this files contains the information of input ranges.
- MATLAB Model – is input to HiLiTE, model for which the test case need to be generated.
- SI\_51XXXXXX\_Disposition.txt – Is input to HTVC tool, should the present in folder path same as HiLiTE Vectors.csv Path and file name should follow format of SI\_51XXXXXX\_Disposition.txt , here X in file name can be any number[0-9].

## CSCI Specific Files

- Config file – This is the initialization file(ini), which is input to the HTVC tool, this files specifies absolute path of template file BAT, ENV and TST.
- .BAT, .ENV and .TST are inputs to HTVC tool, they are used during conversion to VectorCAST.

## OptionSet Elements for Output Generation to HTVC

Option sets identify how HiLiTE will handle certain block characteristics, the test harness for which you want output generated, and other special circumstances. When you generate output for use with HTVC, you must use this element to identify the output generator (HTVC).

OptionSet elements in a HiLiTE command file can be used as sub elements of either the <Command> or <Model> element. Option sets included in the <Model> element take precedence over those in the <Command> element. The following example shows how to use option sets to create HTVC output. Table 50 defines the key values that can be used with the HTVC OptionSet.

```
<Command name="TestGen">
  <OptionSet name="TestGenDirectives">
    <Option key="MakeStimAndObsPointSuggestions">0n</Option>
  </OptionSet>
  <OptionSet name="Source">
    <Option key="Format">HAMRTW</Option>
    <Option key="R2007a">true</Option>
    <Option key="Declarations">{model}.h</Option>
  </OptionSet>
  <OptionSet name="HTH">
    <Option key="GenerateHarnessCode">True</Option>
    <Option key="UY_IO_Style">True</Option>
  </OptionSet>
  <OptionSet name = "HTVC">
    <Option key="GenerateVectorCastOutput">0n</Option>
  </OptionSet>
  <Model name="TCS_CTRL_MDL24" rateHz=' 5' >
  </Model>
</Command>
```

Table 50. Key Values Available for the HTVC OptionSet

Keys	Definition and Available Values
< Option key="GenerateVectorCastOutput "> On	<b>HTVC</b> specifies that HiLiTE should create output to use in the HTVC test harness. If you use this value, you must create a HTVC option set in which you specify the character set to use in the output. <b>GenerateVectorCastOutput</b> On specifies HiLiTE generate t Vector Cast format output fils.
<Option key="GenerateHarnessCode">True</O ption>	<b>GenerateHarnessCode</b> should the set to ture, which instructs the HiLiTE to Generate the interface files, HTVC tool uses this interface files.



# INDEX

## %

%HILITE\_HOME%, 7

## A

acronyms, xviii  
action coverage, 74  
alternate template families, 144  
assigning test cases to groups, 148  
attributes, element, 23, 31

## B

Beyond Boundary, 242  
block attributes  
    supported by HiLiTE, 228  
block libraries, HiLiTE support for, 208  
block requirements, 141  
    capturing, 144  
    categories, 110  
    coverage, 240  
    coverage reporting, 111  
    creating, 141  
    robustness, 111  
Block Type, definition, 129  
blocks  
    complex, 70  
    function, 70  
    requirements, 110  
    with dynamically generated test cases, 240  
Boeing block attributes  
    supported by HiLiTE, 230  
Boolean data type  
    allowed values, 149  
Bugzilla tracking system, 202

## C

cascaded function blocks, 70  
CCC block attributes  
    supported by HiLiTE, 232  
command file elements  
    advanced, 48  
    Args, 42  
    basic, 25  
    Block, 49  
    BlockDBFile, 153  
    BlockDBFile, 33

BlockType, 49  
Category, 26, 42  
Column, 27  
ColumnSpec, 26, 53  
Command, 28, 32, 53, 235, 236  
ConstantsMapping, 237  
Extension, 26  
HiLiTE Mgmt, 48, 235  
HiLiTEParameters, 25  
LibraryMapping, 236  
Model, 29, 32, 49, 53, 235  
ModelAnalysis, 45  
OptionSet, 33, 54  
PossibleStimPointsDetails, 94  
ProjectPath, 25  
Root, 42  
SuggestedObservationPoints, 94  
SymbolFile, 28, 53, 194  
TemplateDBFile, 153  
TemplateDBFile, 33  
TemplateProc, 49  
TestOutput, 25, 42, 102  
ToolKeywords, 237  
Translation, 49  
User, 42  
Version, 42  
command file elements:, 220, 273  
command file processing, 52  
command files  
    about, 5, 6, 10, 52  
    advanced elements, 48  
    basic elements, 25  
    column specification, 195, 196  
    creating, 11, 23  
    diagnostic elements, 234  
    elements, 31, 186  
    format, 23  
    language independent representations, 194  
    model analysis, 116  
    observation points, 95  
    reading, 24  
    Stateflow, 81  
    test generation, 53, 55  
command options, 30  
common block attributes, 233  
compiling and linking model and harness code, 164  
compiling HTH code, 164  
compiling model code, 165  
complex blocks, 70  
complex models, 91, 119



- analyzing, 119
- Component Test Platform, 218
- condition coverage
  - test cases, 74
- configuration data, 7, 152
- configuration file errors, 186
- configuring changes to blocks and templates, 152
- creating
  - block requirements, 141
  - command files, 11, 23
  - data type files, 194, 195
  - models, 10
  - range files, 11, 194, 195
  - templates, 142
  - test cases, 146
  - test cases, tips, 149
- CTP test vectors, 218

## D

- data type files
  - creating, 194, 195
- data types, 134
- debug vectors, 79
- definitions, xv, xviii, 92
- design defects, 104
- diagnostics, 234
- DO-178B objectives, 2, 91, 102, 110, 207
- DOS command prompt, 6
- DOS command window, 15
  - output, 97
- dynamic templates, 240

## E

- elements, command file, 23, 25, 30, 31, 48, 186, 220, 234, 273, *See also* command file elements for individual elements
- errors, 98, 186
  - data type and range propagation, 107
  - model, 121
  - spurious in status report, 108
  - Stateflow, 112
- example models, 10, 56
  - memory switch pattern, 67
  - RiggedUnrigged, 84
  - SimpleExampleModel, 163
  - Statechart\_In\_Isolation.mdl, 75
  - Stateflow, 84
  - Validated Localizer, 63
- examples, 24, 28, 69, 95
  - BasicMath, 55, 58
  - Stateflow, 83
  - test generation, 63
- expected output values, 69
- extensive model analysis, 119

## F

- FAQ, troubleshooting, 191
- file location**, 24
- file paths, 24

- formatting path designations**, 24
- forward propagation, 61
- function blocks, 70
- functional requirements, 3, 110, 111

## G

- generated tests, 57
- generating test cases, 51
- generating test vectors
  - CTP, 218
- generating tests, 12, 15
- groups, test cases, 148

## H

- HAM block attributes
  - supported by HiLiTE, 229
- HAM block library, 208
- HAM blocks, 240
- HAM versions, HiLiTE support for, 208
- HiLiTE, 10
  - command files, 6
  - configuration data, 7
  - executing, 5, 12, 15
  - how it works, 1, 10, 52
  - input, optional, 7
  - input, required, 6
  - installation procedures, 2
  - logs, 7, 99
  - output, 7, 15, 57
  - processing, 10
  - processing requirements, 2
  - support, 2, 186
  - tutorial, 9
- HiLiTE Shell, 5, 12
- HiLiTE support for block libraries, 208
- HiLiTE support for HAM versions, 208
- HiLiTE support for MATLAB versions, 208
- HiLiTE Test Harness, 162
- HiLiTE Test Vectors to VectorCAST conversion, 272
- HTH executable, 166
- HTVC OptionSet, 273
- HTVC Tool, 272

## I

- improving test generation coverage, 91
- INI file, 182
- interface requirements, 35, 110, 111
- interpolation, 242
- isolated Stateflow charts, 75

## K

- keyword files, 8

## L

- Lag filter, test report, 65
- log files, 7, 99

## Index

lookup tables, 242

## M

- makefiles, 166
- mapping files, 8
- mapping test cases to block requirements, 141
- MATLAB models, 7
- MATLAB versions, HiLiTE support for, 208
- MDVD, 161
- measuring expected output, 61
- messages
  - warning and error, 99
- messages, 186
- minimum distinct value difference, 158, 161
- model analysis, 7, 104, 115
- model defects, 121
- model summary reports, 57, 102
- models
  - complex, 91
  - creating, 10
  - design defects, 104
  - examples, 10, 61
  - MATLAB, 7
- modified block data, 152
- modified template data, 152
- modifying templates, 145
- multi-chart StateFlow models, 81

## O

- objectives
  - DO-178B verification, 2, 91, 102, 110, 207
- observation points, 3, 4, 61, 62, 91, 93
  - placement, 95
- option key values, 30
- Option keys, 23, 33, 62, 94, 96, 220, 238
- option sets, 273
- option sets for CTP, 219
- OptionSet, 49
  - CTP, 45, 220
  - DataTypeAndRange, 40
  - DisableTestPointsExplicitly, 41
  - General, 34, 220
  - HTH, 164
  - HTVC, 273
  - ModelAnalysisDirectives, 46, 126
  - SateFlowOptions, 81
  - Scripts, 238
  - SignalBoundaries, 38, 62, 75, 95, 96, 237
  - Source, 43
  - Source, 220
  - Source, 225
  - StateflowDebugVectorsOption, 38
  - StateflowOptions, 36
  - TestGenDirectives, 35, 93
  - TestGenLimits, 34, 239
  - UnitDelayOverrideVariables, 47
  - UseExplicitAT3Anchors, 225
  - UseExplicitPortVariableNames, 42, 62, 63
  - UseExplicitSUTName, 45
  - ZipFilesOption, 44

- OptionSets, 23, 30, 31, 32, 33, 220
  - TestTemplates, 242
- output
  - test generation, 15, 57
  - test vector format, 54
  - test vectors, 8, 94
- output values, tolerance, 168
- overriding default tolerance, 168

## P

- path designations**, 24
- placing observation points, 95
- port variable names, 62
- propagation, 61

## R

- range files, 7, 11
  - creating, 194, 195
  - generating, 126
- rawValidSwitch1\_Memory pattern, 66
- robustness requirements, 35, 110, 111, 132
- running the test harness executable, 166

## S

- secondary stimuli, 91
- SimulinkTestGeneration, 103
- state reachability, 74
- Stateflow
  - action coverage, 74
  - isolated charts, 75
  - test cases, 73
- Stateflow example, 83
- StateflowTestCreation, 103
- status reports, 7, 16, 57, 105
- StimPointsSuggestions, 95
- Stimulate blocks, 93
- stimulation points, 91
  - placement, 95
  - suggestions, 94
- suggestion capability, 93
- summary reports, 7, 16

## T

- template families, 144
  - alternate, 144
- Template Formula Language, 154
- Template Manager, 128, 240
  - block requirements, 134
  - block type details, 130
  - mapping test cases to block requirements, 141
  - menus, 149
  - template details, 135
  - test case details, 136
  - test case steps, 138
- templates
  - adding test cases, 146
  - creating, 142

- creating test case groups and conditions, 145
  - deleting test cases, 146
  - modifying, 145
  - modifying test cases, 146
- terminology, xv, xviii, 92
- test case
  - pass/fail, 69
- test case report, 57
- test cases
  - creating, tips, 149
- test cases, 51, 57, 73
  - action coverage, 74
  - adding steps, 147
  - categories, 73
  - condition coverage, 74
  - conditions in templates, 145
  - creating in templates, 146
  - deleting steps, 148
  - groups, 148
  - groups in templates, 145
  - reordering steps, 148
  - state reachability, 74
  - Stateflow, 73
  - transition reachability, 74
- test generation*, 3, 7, 12, 15, 31, 43, 51, 53, 57, 63, 78, 93, 153
  - improving coverage, 91
  - output, 15
- test generation objectives, 204
- test generation status report, 106
- test harnesses, 1, 8, 51, 54
- test point variables, 62
- test points, 62
- test reports, 18, 57, 60
- test vector files, 16
- test vector output location, 31
- test vector reports, 7
- test vectors, 3, 7, 18, 31, 54, 168
  - generating CTP, 218
- testability limitations, 109
- testing isolated Stateflow charts, 75
- testing reports, 16

- tolerance, 69, 168, 182, 183, 219
  - overriding default, 168
  - specification, 71
- tolerance used by HiLiTE Test Harness, 168
- tool configuration data, 152
- TPI Converter, 170, 172
- TPI files, 170, 218
  - generation, 192
- transition reachability, 74
- troubleshooting, 186
- tutorial, 9

## U

- universal data types, 194
- UniversalDataType, 134
- user supplied variables, 79
- user-modified block data, 154
- user-modified template data, 154
- user-supplied test vectors, 78
- USV, 78
  - specifying a chart, 81

## V

- Validated Localizer
  - test generation, 63
- validSwitch memory procedure, 66
- variables, 62
- Vector Assignment, 269
- VectorCAST, 272

## W

- warning and error messages, 99
- Writer Report, 57, 79

## X

- XML files, 7, 21, 152
- XML tags, 23