

Simulink®

Modeling Guidelines for High-Integrity Systems



MATLAB® & SIMULINK®

R2015a



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Modeling Guidelines for High-Integrity Systems

© COPYRIGHT 2009–2015 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

September 2009	Online only	New for Version 1.0 (Release 2009b)
April 2010	Online only	Revised for Version 1.1 (Release 2010a)
September 2010	Online only	Revised for Version 1.2 (Release 2010b)
April 2011	Online only	Revised for Version 1.3 (Release 2011a)
September 2011	Online only	Revised for Version 1.4 (Release 2011b)
March 2012	Online only	Revised for Version 1.5 (Release 2012a)
September 2012	Online only	Revised for Version 1.6 (Release 2012b)
March 2013	Online only	Revised for Version 1.7 (Release 2013a)
September 2013	Online only	Revised for Version 1.8 (Release 2013b)
March 2014	Online only	Revised for Version 1.9 (Release 2014a)
October 2014	Online only	Revised for Version 1.10 (Release 2014b)
March 2015	Online only	Revised for Version 1.11 (Release 2015a)

	Introduction	
1		
	Motivation	1-2
	Guideline Template	1-3
	Model Advisor Checks for High-Integrity Modeling	
	Guidelines	1-4
	Simulink Block Considerations	
2		
	Math Operations	2-2
	hisl_0001: Usage of Abs block	2-3
	hisl_0002: Usage of Math Function blocks (rem and reciprocal)	2-5
	hisl_0003: Usage of Square Root blocks	2-7
	hisl_0028: Usage of Reciprocal Square Root blocks	2-8
	hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)	2-10
	hisl_0005: Usage of Product blocks	2-13
	hisl_0029: Usage of Assignment blocks	2-15
	Ports & Subsystems	2-20
	hisl_0006: Usage of While Iterator blocks	2-21
	hisl_0007: Usage of While Iterator subsystems	2-23
	hisl_0008: Usage of For Iterator Blocks	2-26
	hisl_0009: Usage of For Iterator Subsystem blocks	2-28
	hisl_0010: Usage of If blocks and If Action Subsystem blocks	2-29
	hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks	2-31

hisl_0012: Usage of conditionally executed subsystems	2-33
hisl_0024: Inport interface definition	2-35
hisl_0025: Design min/max specification of input interfaces .	2-36
hisl_0026: Design min/max specification of output interfaces	2-38
Signal Routing	2-40
hisl_0013: Usage of data store blocks	2-41
hisl_0015: Usage of Merge blocks	2-45
hisl_0021: Consistent vector indexing method	2-47
hisl_0022: Data type selection for index signals	2-49
hisl_0023: Verification of model and subsystem variants . . .	2-50
Logic and Bit Operations	2-51
hisl_0016: Usage of blocks that compute relational operators	2-52
hisl_0017: Usage of blocks that compute relational operators	
(2)	2-54
hisl_0018: Usage of Logical Operator block	2-55
hisl_0019: Usage of Bitwise Operator block	2-57

Stateflow Chart Considerations

3

Chart Properties	3-2
hisf_0001: Mealy and Moore semantics	3-3
hisf_0002: User-specified state/transition execution order . . .	3-5
hisf_0009: Strong data typing (Simulink and Stateflow	
boundary)	3-7
hisf_0011: Stateflow debugging settings	3-9
Chart Architecture	3-11
hisf_0003: Usage of bitwise operations	3-12
hisf_0004: Usage of recursive behavior	3-13
hisf_0007: Usage of junction conditions (maintaining mutual	
exclusion)	3-15
hisf_0010: Usage of transition paths (looping out of parent of	
source and destination objects)	3-16
hisf_0012: Chart comments	3-18
hisf_0013: Usage of transition paths (crossing parallel state	
boundaries)	3-19
hisf_0014: Usage of transition paths (passing through states)	3-22

hisl_0015: Strong data typing (casting variables and parameters in expressions)	3-23
---	------

MATLAB Function and MATLAB Code Considerations

4

MATLAB Functions	4-2
hisl_0001: Usage of standardized MATLAB function headers	4-3
hisl_0002: Strong data typing at MATLAB function boundaries	4-4
hisl_0003: Limitation of MATLAB function complexity	4-6
hisl_0005: Usage of global variables in MATLAB functions	4-8
MATLAB Code	4-11
hisl_0004: MATLAB Code Analyzer recommendations for code generation	4-11
hisl_0006: MATLAB code if / elseif / else patterns	4-15
hisl_0007: MATLAB code switch / case / otherwise patterns	4-17
hisl_0008: MATLAB code relational operator data types	4-19
hisl_0009: MATLAB code with equal / not equal relational operators	4-20
hisl_0010: MATLAB code with logical operators and functions	4-22

Configuration Parameter Considerations

5

Solver	5-2
hisl_0040: Configuration Parameters > Solver > Simulation time	5-3
hisl_0041: Configuration Parameters > Solver > Solver options	5-4
hisl_0042: Configuration Parameters > Solver > Tasking and sample time options	5-5

Diagnostics	5-7
hisl_0043: Configuration Parameters > Diagnostics > Solver ..	5-8
hisl_0044: Configuration Parameters > Diagnostics > Sample Time	5-10
hisl_0301: Configuration Parameters > Diagnostics > Compatibility	5-13
hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters	5-14
hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block	5-15
hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization	5-16
hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging	5-17
hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals	5-18
hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses	5-19
hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls	5-20
hisl_0309: Configuration Parameters > Diagnostics > Type Conversion	5-21
hisl_0310: Configuration Parameters > Diagnostics > Model Referencing	5-22
hisl_0311: Configuration Parameters > Diagnostics > Stateflow	5-23
 Optimizations	 5-24
hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)	5-25
hisl_0046: Configuration Parameters > Optimization > Block reduction	5-26
hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)	5-27
hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold	5-28
hisl_0052: Configuration Parameters > Optimization > Data initialization	5-29
hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values	5-30
hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions .	5-31

hisl_0055: Prioritization of code generation objectives for high-integrity systems	5-32
--	------

MISRA-C:2004 Compliance Considerations

6

Modeling Style	6-2
hisl_0061: Unique identifiers for clarity	6-3
hisl_0062: Global variables in graphical functions	6-9
hisl_0063: Length of user-defined function names to improve MISRA-C:2004 compliance	6-11
hisl_0064: Length of user-defined type object names to improve MISRA-C:2004 compliance	6-12
hisl_0065: Length of signal and parameter names to improve MISRA-C:2004 compliance	6-13
hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance	6-14
hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance	6-15
 Block Usage	 6-17
hisl_0020: Blocks not recommended for MISRA-C:2004 compliance	6-17
hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance	6-18
hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance	6-21
 Configuration Settings	 6-22
hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance	6-22
hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance	6-24
hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance	6-25
 Stateflow Chart Considerations	 6-26
hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance	6-27

hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance	6-29
hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance	6-31
hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance	6-32
hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance	6-33
System Level	6-36
hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance	6-36
hisl_0402: Use of custom #pragma to improve MISRA-C:2004 compliance	6-37
hisl_0403: Use of char data type improve MISRA-C:2004 compliance	6-38

Introduction

- “Motivation” on page 1-2
- “Guideline Template” on page 1-3
- “Model Advisor Checks for High-Integrity Modeling Guidelines” on page 1-4

Motivation

MathWorks® intends this document for engineers developing models and generating code for high-integrity systems using Model-Based Design with MathWorks products. This document describes creating Simulink® models that are complete, unambiguous, statically deterministic, robust, and verifiable. The document focus is on model settings, block usage, and block parameters that impact simulation behavior or code generated by the Embedded Coder® product.

These guidelines do not assume that you use a particular safety or certification standard. The guidelines reference some safety standards where applicable, including:

- DO-178C / DO-331
- IEC 61508
- ISO 26262
- EN 50128
- MISRA C

Guidelines in this document might also be applicable to related standards, including IEC 62304, and DO-254.

You can use the Model Advisor to support adhering to these guidelines. Each guideline lists the checks that are applicable to that guideline, or to parts of that guideline.

This document does not address model style or development processes. For more information about creating models in a way that improves consistency, clarity, and readability, see the “MAAB Control Algorithm Modeling” guidelines. Development process guidance and additional information for specific standards is available with the IEC Certification Kit (for IEC 61508 and ISO 26262) and DO Qualification Kit (for DO-178 and DO-254) products.

Disclaimer While adhering to the recommendations in this document will reduce the risk that an error is introduced during development and not be detected, it is not a guarantee that the system being developed will be safe. Conversely, if some of the recommendations in this document are not followed, it does not mean that the system being developed will be unsafe.

Guideline Template

Guideline descriptions are documented, using the following template. Companies that want to create additional guidelines are encouraged to use the same template.

ID: Title	<i>XX_nnnn</i> : Title of the guideline (unique, short)
Description	Description of the guideline
Prerequisites	Links to guidelines that are prerequisites to this guideline (ID: Title)
Notes	Notes for using the guideline
Rationale	Rational for providing the guideline
Model Advisor Check	Title of and link to the corresponding Model Advisor check, if a check exists
References	References to standards that apply to guideline
See Also	Links to additional information
Last Changed	Version number of last change
Examples	Guideline examples

Model Advisor Checks for High-Integrity Modeling Guidelines

Simulink Verification and Validation™ includes Model Advisor checks for compliance with safety standards referenced in the high-integrity guidelines, including:

- DO-178C / DO-331
- IEC 61508
- ISO 26262
- EN 50128

The high-integrity guidelines and corresponding Model Advisor checks are summarized in the following table. Not all guidelines have Model Advisor checks. For some of the guidelines without Model Advisor checks, it is not possible to automate checking of the guideline. Guidelines without a corresponding check are noted as not applicable. For information on using the Model Advisor, see “Run Model Checks” in the Simulink documentation.

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders:
“hisl_0001: Usage of Abs block”	<ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262 DO-178C/DO-331: “Check usage of Math Operations blocks” IEC 61508, EN 50128 and ISO 26262: “Check usage of Math Operations blocks”
“hisl_0002: Usage of Math Function blocks (rem and reciprocal)”	DO-178C/DO-331: “Check usage of Math Operations blocks”
“hisl_0003: Usage of Square Root blocks”	Not applicable
“hisl_0028: Usage of Reciprocal Square Root blocks”	Not applicable
“hisl_0004: Usage of Math Function blocks (natural	DO-178C/DO-331: “Check usage of Math Operations blocks”

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
logarithm and base 10 logarithm)”	
“hisl_0005: Usage of Product blocks”	DO-178C/DO-331: “Check safety-related diagnostic settings for signal data”
“hisl_0029: Usage of Assignment blocks”	DO-178C/DO-331: “Check usage of Math Operations blocks” IEC 61508, EN 50128 and ISO 26262: “Check usage of Math Operations blocks”
“hisl_0006: Usage of While Iterator blocks”	DO-178C/DO-331: “Check usage of Ports and Subsystems blocks” IEC 61508, EN 50128 and ISO 26262: “Check usage of Ports and Subsystems blocks”
“hisl_0007: Usage of While Iterator subsystems”	DO-178C/DO-331: “Check usage of Ports and Subsystems blocks” IEC 61508, EN 50128 and ISO 26262: “Check usage of Ports and Subsystems blocks”
“hisl_0008: Usage of For Iterator Blocks”	DO-178C/DO-331: “Check usage of Ports and Subsystems blocks” IEC 61508, EN 50128 and ISO 26262: “Check usage of Ports and Subsystems blocks”
“hisl_0009: Usage of For Iterator Subsystem blocks”	DO-178C/DO-331: “Check usage of Ports and Subsystems blocks” IEC 61508, EN 50128 and ISO 26262: “Check usage of Ports and Subsystems blocks”
“hisl_0010: Usage of If blocks and If Action Subsystem blocks”	Not applicable

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
"hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks"	Not applicable
"hisl_0012: Usage of conditionally executed subsystems"	Not applicable
"hisl_0024: Inport interface definition"	IEC 61508, EN 50128 and ISO 26262: "Check for root Inports with missing properties"
"hisl_0025: Design min/max specification of input interfaces"	IEC 61508, EN 50128 and ISO 26262: "Check for root Inports with missing range definitions"
"hisl_0026: Design min/max specification of output interfaces"	IEC 61508, EN 50128 and ISO 26262: "Check for root Outports with missing range definitions"
"hisl_0013: Usage of data store blocks"	DO-178C/DO-331: "Check safety-related diagnostic settings for data store memory"
"hisl_0015: Usage of Merge blocks"	Not applicable
"hisl_0021: Consistent vector indexing method"	DO-178C/DO-331: "Check for inconsistent vector indexing methods" IEC 61508, EN 50128 and ISO 26262: "Check for inconsistent vector indexing methods"
"hisl_0022: Data type selection for index signals"	Not applicable
"hisl_0023: Verification of model and subsystem variants"	Not applicable
"hisl_0016: Usage of blocks that compute relational operators"	DO-178C/DO-331: "Check usage of Logic and Bit Operations blocks" IEC 61508, EN 50128 and ISO 26262: "Check usage of Logic and Bit Operations blocks"

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
"hisl_0017: Usage of blocks that compute relational operators (2)"	DO-178C/DO-331: "Check usage of Logic and Bit Operations blocks" IEC 61508, EN 50128 and ISO 26262: "Check usage of Logic and Bit Operations blocks"
"hisl_0018: Usage of Logical Operator block"	DO-178C/DO-331: "Check usage of Logic and Bit Operations blocks" and "Check safety-related optimization settings" IEC 61508, EN 50128 and ISO 26262: "Check usage of Logic and Bit Operations blocks"
"hisl_0019: Usage of Bitwise Operator block"	Not applicable
"hisf_0001: Mealy and Moore semantics"	DO-178C/DO-331: "Check state machine type of Stateflow charts" IEC 61508, EN 50128 and ISO 26262: "Check state machine type of Stateflow charts"
"hisf_0002: User-specified state/transition execution order"	DO-178C/DO-331: "Check Stateflow charts for ordering of states and transitions" IEC 61508, EN 50128 and ISO 26262: "Check usage of Stateflow constructs"
"hisf_0009: Strong data typing (Simulink and Stateflow boundary)"	IEC 61508, EN 50128 and ISO 26262: "Check usage of Stateflow constructs"
"hisf_0011: Stateflow debugging settings"	DO-178C/DO-331: "Check Stateflow debugging options" IEC 61508, EN 50128 and ISO 26262: "Check usage of Stateflow constructs"

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders:
"hisf_0003: Usage of bitwise operations"	In Modeling Standards for MAAB folder, "Check for bitwise operations in Stateflow charts"
"hisf_0004: Usage of recursive behavior"	Not applicable
"hisf_0007: Usage of junction conditions (maintaining mutual exclusion)"	Not applicable
"hisf_0010: Usage of transition paths (looping out of parent of source and destination objects)"	Not applicable
"hisf_0012: Chart comments"	Not applicable
"hisf_0013: Usage of transition paths (crossing parallel state boundaries)"	Not applicable
"hisf_0014: Usage of transition paths (passing through states)"	Not applicable
"hisf_0015: Strong data typing (casting variables and parameters in expressions)"	Not applicable
"himl_0001: Usage of standardized MATLAB function headers"	Not applicable
"himl_0002: Strong data typing at MATLAB function boundaries"	DO-178C/DO-331: "Check for MATLAB Function interfaces with inherited properties" IEC 61508, EN 50128 and ISO 26262: "Check for MATLAB Function interfaces with inherited properties"

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
“himl_0003: Limitation of MATLAB function complexity”	DO-178C/DO-331: “Check MATLAB Function metrics” IEC 61508, EN 50128 and ISO 26262: “Check MATLAB Function metrics”
“himl_0004: MATLAB Code Analyzer recommendations for code generation”	DO-178C/DO-331: “Check MATLAB Code Analyzer messages” IEC 61508, EN 50128 and ISO 26262: “Check MATLAB Code Analyzer messages”
“himl_0005: Usage of global variables in MATLAB functions”	DO-178C/DO-331: “Check MATLAB code for global variables” IEC 61508, EN 50128 and ISO 26262: “Check MATLAB code for global variables”
“himl_0006: MATLAB code if / elseif / else patterns”	Not applicable
“himl_0007: MATLAB code switch / case / otherwise patterns”	Not applicable
“himl_0008: MATLAB code relational operator data types”	Not applicable
“himl_0009: MATLAB code with equal / not equal relational operators”	Not applicable
“himl_0010: MATLAB code with logical operators and functions”	Not applicable
“hisl_0040: Configuration Parameters > Solver > Simulation time”	Not applicable

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
"hisl_0041: Configuration Parameters > Solver > Solver options"	Not applicable
"hisl_0042: Configuration Parameters > Solver > Tasking and sample time options"	Not applicable
"hisl_0043: Configuration Parameters > Diagnostics > Solver"	DO-178C/DO-331: "Check safety-related diagnostic settings for solvers"
"hisl_0044: Configuration Parameters > Diagnostics > Sample Time"	DO-178C/DO-331: "Check safety-related diagnostic settings for sample time"
"hisl_0301: Configuration Parameters > Diagnostics > Compatibility"	DO-178C/DO-331: "Check safety-related diagnostic settings for compatibility"
"hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters"	DO-178C/DO-331: "Check safety-related diagnostic settings for parameters"
"hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block"	Not applicable
"hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization"	DO-178C/DO-331: "Check safety-related diagnostic settings for model initialization"
"hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging"	Not applicable

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
"hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals"	DO-178C/DO-331: "Check safety-related diagnostic settings for signal connectivity"
"hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses"	DO-178C/DO-331: "Check safety-related diagnostic settings for bus connectivity"
"hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls"	DO-178C/DO-331: "Check safety-related diagnostic settings that apply to function-call connectivity"
"hisl_0309: Configuration Parameters > Diagnostics > Type Conversion"	DO-178C/DO-331: "Check safety-related diagnostic settings for type conversions"
"hisl_0310: Configuration Parameters > Diagnostics > Model Referencing"	DO-178C/DO-331: "Check safety-related diagnostic settings for model referencing"
"hisl_0311: Configuration Parameters > Diagnostics > Stateflow"	Not applicable
"hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)"	DO-178C/DO-331: "Check safety-related optimization settings"
"hisl_0046: Configuration Parameters > Optimization > Block reduction"	DO-178C/DO-331: "Check safety-related optimization settings"
"hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)"	DO-178C/DO-331: "Check safety-related optimization settings"

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
“hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold”	Not applicable
“hisl_0052: Configuration Parameters > Optimization > Data initialization”	DO-178C/DO-331: “Check safety-related optimization settings”
“hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values”	DO-178C/DO-331: “Check safety-related optimization settings”
“hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions”	DO-178C/DO-331: “Check safety-related optimization settings”
“hisl_0055: Prioritization of code generation objectives for high-integrity systems”	Not applicable
“hisl_0061: Unique identifiers for clarity”	DO-178C/DO-331: “Check Stateflow charts for uniquely defined data objects” IEC 61508, EN 50128 and ISO 26262: “Check usage of Stateflow constructs”
“hisl_0062: Global variables in graphical functions”	Not applicable

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
“hisl_0063: Length of user-defined function names to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0064: Length of user-defined type object names to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0065: Length of signal and parameter names to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0020: Blocks not recommended for MISRA-C:2004 compliance”	In Modeling Guidelines for MISRA-C:2004 folder: “Check for blocks not recommended for MISRA-C:2004 compliance”
“hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance”	In Modeling Guidelines for MISRA-C:2004 folder: “Check configuration parameters for MISRA-C:2004 compliance”

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders: <ul style="list-style-type: none"> • Modeling Standards for DO-178C/DO-331 • Modeling Standards for IEC 61508 • Modeling Standards for EN 50128 • Modeling Standards for ISO 26262
“hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance”	Not applicable
“hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance”	Not applicable
“hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance”	Not applicable
“hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance”	Not applicable
“hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance”	Not applicable
“hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance”	Not applicable
“hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance”	Not applicable
“hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance”	Not applicable

High-Integrity Modeling Guideline	Checks available in Model Advisor By Task folders:
"hisl_0402: Use of custom #pragma to improve MISRA-C:2004 compliance"	Not applicable
"hisl_0403: Use of char data type improve MISRA-C:2004 compliance"	Not applicable

Simulink Block Considerations

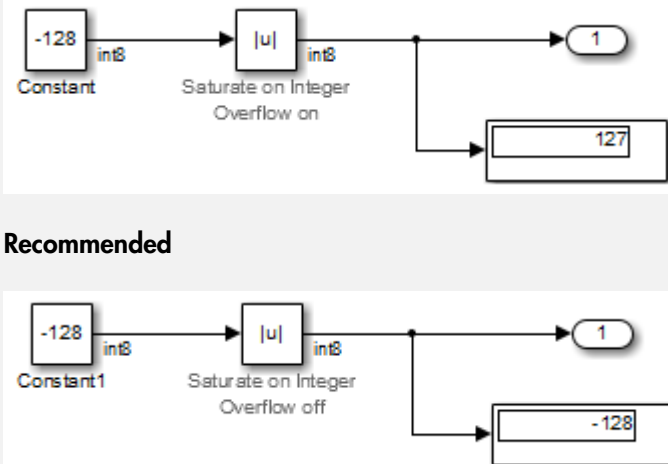
- “Math Operations” on page 2-2
- “Ports & Subsystems” on page 2-20
- “Signal Routing” on page 2-40
- “Logic and Bit Operations” on page 2-51

Math Operations

In this section...
“hisl_0001: Usage of Abs block” on page 2-3
“hisl_0002: Usage of Math Function blocks (rem and reciprocal)” on page 2-5
“hisl_0003: Usage of Square Root blocks” on page 2-7
“hisl_0028: Usage of Reciprocal Square Root blocks” on page 2-8
“hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)” on page 2-10
“hisl_0005: Usage of Product blocks” on page 2-13
“hisl_0029: Usage of Assignment blocks” on page 2-15

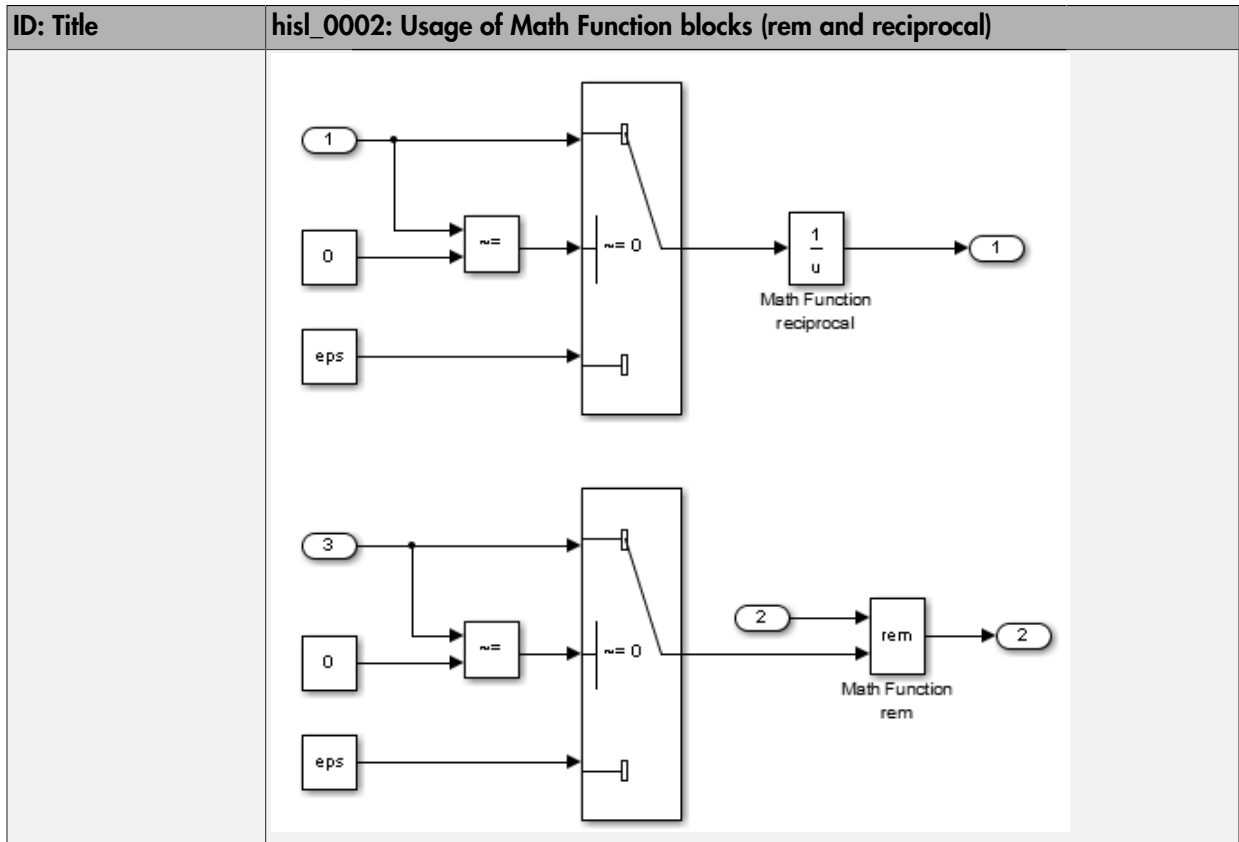
hisl_0001: Usage of Abs block

ID: Title	hisl_0001: Usage of Abs block	
Description	To support robustness of generated code, when using the Abs block,	
	A	Avoid Boolean and unsigned integer data types as inputs to the Abs block.
	B	In the Abs block parameter dialog box, select Saturate on integer overflow .
Notes	<p>The Abs block does not support Boolean data types. Specifying an unsigned input data type, might optimize the Abs block out of the generated code, resulting in a block you cannot trace to the generated code.</p> <p>For signed data types, Simulink does not represent the absolute value of the most negative value. When you select Saturate on integer overflow, the absolute value of the data type saturates to the most positive representable value. When you clear Saturate on integer overflow, absolute value calculations in the simulation and generated code might not be consistent or expected.</p>	
Rationale	A	Support generation of traceable code.
	B	Achieve consistent and expected behavior of model simulation and generated code.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Math Operations blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Math Operations blocks • By Task > Modeling Standards for EN 50128 > Check usage of Math Operations blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Math Operations blocks <p>For DO-178C/DO-331 check details, see “Check usage of Math Operations blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Math Operations blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' 	

ID: Title	hisl_0001: Usage of Abs block
	<p>IEC 61508-3, Table A.4 (3) 'Defensive programming' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' IEC 61508-3, Table B.8 (3) 'Control Flow Analysis'</p> <ul style="list-style-type: none"> • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' ISO 26262-6, Table 7 (f) 'Control flow analysis' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.19 (3) 'Control Flow Analysis' • DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' • MISRA-C:2004, Rule 14.1 MISRA-C:2004, Rule 21.1
Last Changed	R2013b
Examples	 <p>The first diagram, labeled 'Recommended', shows an Abs block with the input '-128' (Constant) and 'inB'. The block is configured with 'Saturate on Integer Overflow on'. The output is split: one path goes to a scope showing '1', and the other path goes to a scope showing '127'.</p> <p>The second diagram, labeled 'Not Recommended', shows the same Abs block with the input '-128' (Constant1) and 'inB'. The block is configured with 'Saturate on Integer Overflow off'. The output is split: one path goes to a scope showing '1', and the other path goes to a scope showing '-128'.</p>

hisl_0002: Usage of Math Function blocks (rem and reciprocal)

ID: Title	hisl_0002: Usage of Math Function blocks (rem and reciprocal)	
Description	To support robustness of generated code, when using the Math Function block with remainder-after-division (<code>rem</code>) or reciprocal (<code>reciprocal</code>) functions:	
	A	Protect the input of the <code>reciprocal</code> function from going to zero.
	B	Protect the second input of the <code>rem</code> function from going to zero.
Note	You can get a divide-by-zero operation, resulting in an infinite (<code>Inf</code>) output value for the <code>reciprocal</code> function, or a Not-a-Number (<code>NaN</code>) output value for the <code>rem</code> function. To avoid overflows or undefined values, protect the corresponding input from going to zero.	
Rationale	A, B	Protect against overflows and undefined numerical results.
Model Advisor Checks	<p>By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Math Operations blocks</p> <p>For check details, see “Check usage of Math Operations blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 21.1 	
Last Changed	R2014a	
Examples	In the following example, when the input signal oscillates around zero, the output exhibits a large change in value. You need further protection against the large change in value.	

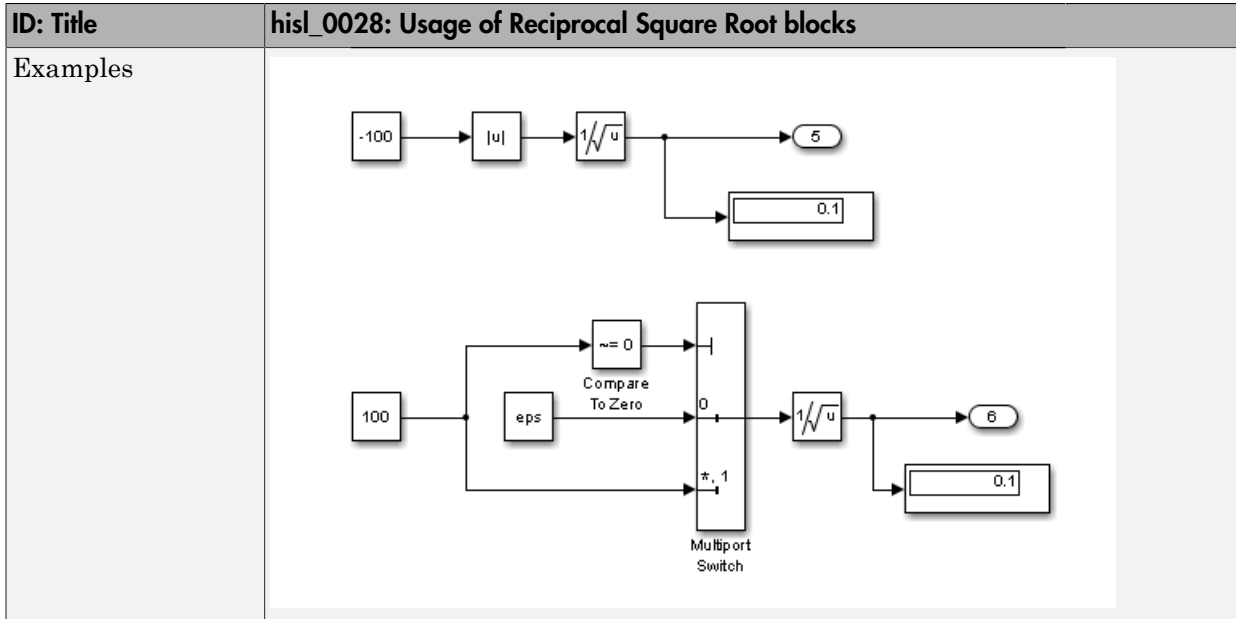


hisl_0003: Usage of Square Root blocks

ID: Title	hisl_0003: Usage of Square Root blocks	
Description	To support robustness of generated code, when using the Square Root block, do one of the following:	
	A	Account for complex numbers as the output.
Rationale	A, B	Avoid undesirable results in generated code.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' 	
Last Changed	R2013b	
Examples	<p>The top diagram shows a square root block with input -100 and output data type 'Complex'. The output is $0 + 10i$.</p> <p>The bottom diagram shows a square root block with input -100, preceded by an absolute value block. The output is 10.</p>	

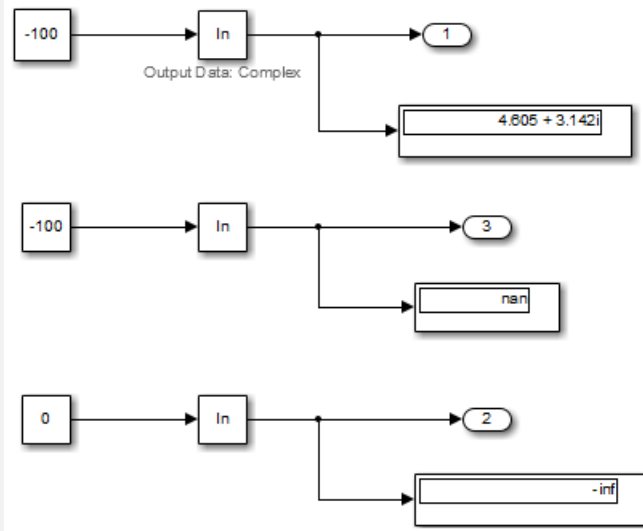
hisl_0028: Usage of Reciprocal Square Root blocks

ID: Title	hisl_0028: Usage of Reciprocal Square Root blocks	
Description	To support robustness of generated code, when using the Reciprocal Square Root block, do one of the following:	
	A	Protect the input from going negative.
	B	Protect the input from going to zero.
Note	You can get a divide-by-zero operation, resulting in an (Inf) output value for the reciprocal function. To avoid overflows or undefined values, protect the corresponding input from going to zero.	
Rationale	A, B	Avoid undesirable results in generated code.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' 	
Last Changed	R2013b	



hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)

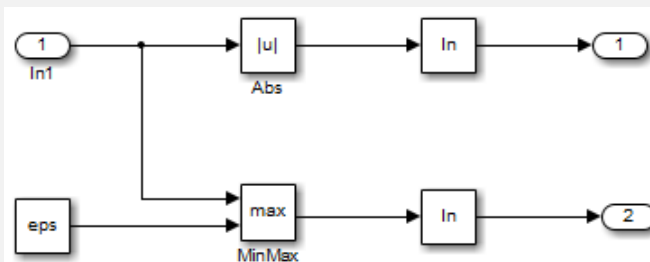
ID: Title	hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)	
Description	To support robustness of generated code, when using the Math Function block with natural logarithm (<code>log</code>) or base 10 logarithm (<code>log10</code>) function parameters,	
	A	Protect the input from going negative.
	B	Protect the input from equaling zero.
	C	Account for complex numbers as the output value.
Notes	If you set the output data type to complex, the natural logarithm and base 10 logarithm functions output complex values for negative input values. If you set the output data type to real, the functions output NAN for negative numbers, and minus infinity (<code>-inf</code>) for zero values.	
Rationale	A, B, C	Support generation of robust code.
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Math Operations blocks For check details, see “Check usage of Math Operations blocks”.	
References	<ul style="list-style-type: none"> IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' 	
Last Changed	R2013b	
Examples		

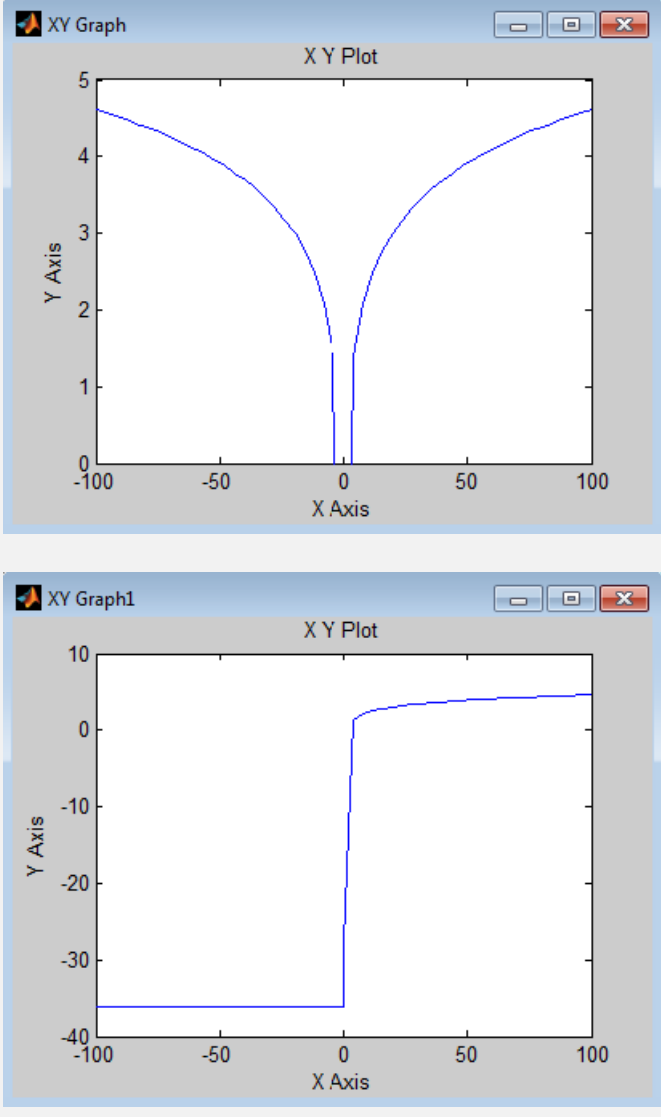
ID: Title hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)


You can protect against:

- Negative numbers using an Abs block.
- Zero values using a combination of the MinMax block and a Constant block, with **Constant value** set to **eps** (epsilon).

The following example displays the resulting output for input values ranging from -100 to 100.



ID: Title	hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)
	 <p>The image displays two Simulink XY Graph blocks. The top block, titled 'XY Graph', shows a plot of a function with a vertical asymptote at X=0. The curve is symmetric about the Y-axis, with Y values ranging from 0 to 5 and X values from -100 to 100. The bottom block, titled 'XY Graph1', shows a plot of a step function. The Y-axis ranges from -40 to 10, and the X-axis ranges from -100 to 100. The function is constant at approximately -35 for X < 0 and jumps to approximately 5 for X > 0.</p>

hisl_0005: Usage of Product blocks

ID: Title	hisl_0005: Usage of Product blocks	
Description	To support robustness of generated code, when using the Product block with divisor inputs,	
	A	In Element-wise (.*) mode, protect divisor inputs from going to zero.
	B	In Matrix (*) mode, protect divisor inputs from becoming singular input matrices.
	C	Set the model configuration parameter Diagnostics > Data Validity > Signals > Division by singular matrix to error.
Notes	<p>When using Product blocks for element-wise divisions, you might get a divide by zero, resulting in a NaN output. To avoid overflows, protect divisor inputs from going to zero.</p> <p>When using Product blocks to compute the inverse of a matrix, or a matrix division, you might get a divide by a singular matrix. This division results in a NaN output. To avoid overflows, protect divisor inputs from becoming singular input matrices.</p> <p>During simulation, while the software inverts one of the input values of a Product block that is in matrix multiplication mode, the Division by singular matrix diagnostic can detect a singular matrix.</p>	
Rationale	A, B, C	Protect against overflows.
Model Advisor Checks	<p>By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for signal data</p> <p>For check details, see “Check safety-related diagnostic settings for signal data”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.4.2.2 'Robustness Test Cases' 	

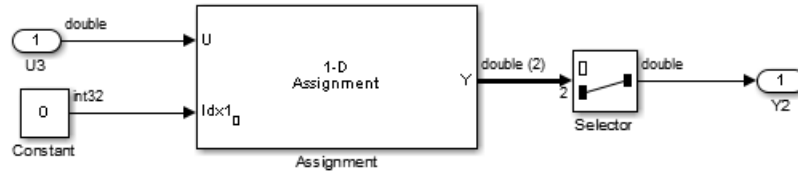
ID: Title	hisl_0005: Usage of Product blocks
	DO-331, Section MB.6.4.3 'Requirements-Based Testing Methods' <ul style="list-style-type: none">• MISRA-C:2004, Rule 21.1
Last Changed	R2013b

hisl_0029: Usage of Assignment blocks

ID: Title	hisl_0029: Usage of Assignment blocks
Description	To support robustness of generated code, when using the Assignment block, initialize array fields before their first use.
Notes	<p>If the output vector of the Assignment block is not initialized with an input to the block, elements of the vector might not be initialized in the generated code.</p> <p>When the Assignment block is used iteratively and all array field are assigned during one simulation time step, you do not need initialization input to the block.</p> <p>Accessing uninitialized elements of block output can result in unexpected behavior.</p>
Rationale	Avoid undesirable results in generated code.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Math Operations blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Math Operations blocks • By Task > Modeling Standards for EN 50128 > Check usage of Math Operations blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Math Operations blocks <p>For DO-178C/DO-331 check details, see “Check usage of Math Operations blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Math Operations blocks”.</p>
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262–6, Table 1(b) 'Use of language subsets' • ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' • DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' • MISRA-C:2004, Rule 9.1

ID: Title	hisl_0029: Usage of Assignment blocks
Last Changed	R2014a

ID: Title hisl_0029: Usage of Assignment blocks

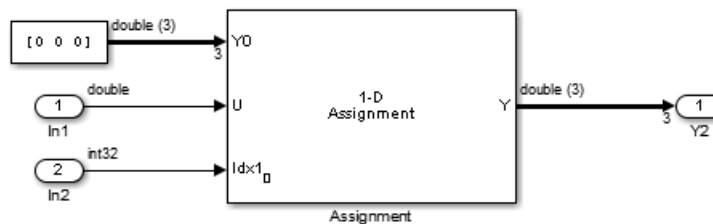
Examples


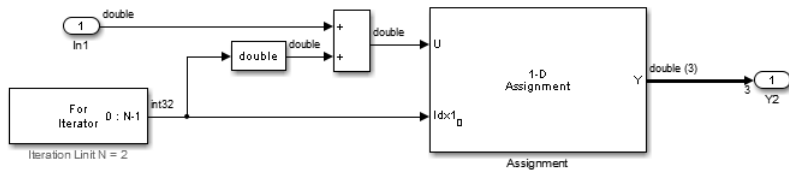
```

31 /* Model step function */
32 void Assignment1_step(void)
33 {
34     real T rtb_Assignment[2];
35
36     /* Assignment: '<Root>/Assignment' incorporates:
37      * Constant: '<Root>/Constant'
38      * Inport: '<Root>/U3'
39      */
40     rtb_Assignment[0] = Assignment1 U.U3;
41
42     /* Output: '<Root>/Y2' */
43     Assignment1 Y.Y2 = rtb_Assignment[1];
44 }

```

Not Recommended: No initialization input Y0 when block is not used iteratively



ID: Title	hisl_0029: Usage of Assignment blocks
	<pre data-bbox="333 303 1164 685"> /* Model step function */ 32 void Assignment2_step(void) 33 { 34 /* Assignment: '<Root>/Assignment' incorporates: 35 * Constant: '<Root>/Constant' 36 * Inport: '<Root>/In1' 37 * Inport: '<Root>/In2' 38 */ 39 Assignment2 Y.Y2[0] = 0.0; 40 Assignment2 Y.Y2[1] = 0.0; 41 Assignment2 Y.Y2[2] = 0.0; 42 Assignment2 Y.Y2[Assignment2 U.In2] = Assignment2 U.In1; 43 } </pre> <p data-bbox="326 720 1142 755">Recommended: Initialization input Y0 when block is not used iteratively</p> 

ID: Title	hisl_0029: Usage of Assignment blocks
	<pre data-bbox="333 298 1172 725">/* Model step function */ 32 void Assignment3_step(void) 33 { 34 <u>int32 T</u> s1_iter; 35 36 /* Outputs for Iterator SubSystem: '<Root>/For Iterator Subsystem' incorporates: 37 * ForIterator: '<Si>/For Iterator' 38 */ 39 for (s1_iter = 0; s1_iter < 2; s1_iter++) { 40 /* Assignment: '<Si>/Assignment' incorporates: 41 * DataTypeConversion: '<Si>/Data Type Conversion' 42 * Inport: '<Root>/In1' 43 * Sum: '<Si>/Add' 44 */ 45 <u>Assignment3 Y</u>.Out1[s1_iter] = <u>Assignment3 U</u>.In1 + ((<u>real T</u>)s1_iter); 46 } 47 48 /* End of Outputs for SubSystem: '<Root>/For Iterator Subsystem' */ 49 }</pre> <p data-bbox="326 760 1090 789">Recommended: Initialize array fields when block is used iteratively</p>

Ports & Subsystems

In this section...

“hisl_0006: Usage of While Iterator blocks” on page 2-21

“hisl_0007: Usage of While Iterator subsystems” on page 2-23

“hisl_0008: Usage of For Iterator Blocks” on page 2-26

“hisl_0009: Usage of For Iterator Subsystem blocks” on page 2-28

“hisl_0010: Usage of If blocks and If Action Subsystem blocks” on page 2-29

“hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks” on page 2-31

“hisl_0012: Usage of conditionally executed subsystems” on page 2-33

“hisl_0024: Inport interface definition” on page 2-35

“hisl_0025: Design min/max specification of input interfaces” on page 2-36

“hisl_0026: Design min/max specification of output interfaces” on page 2-38

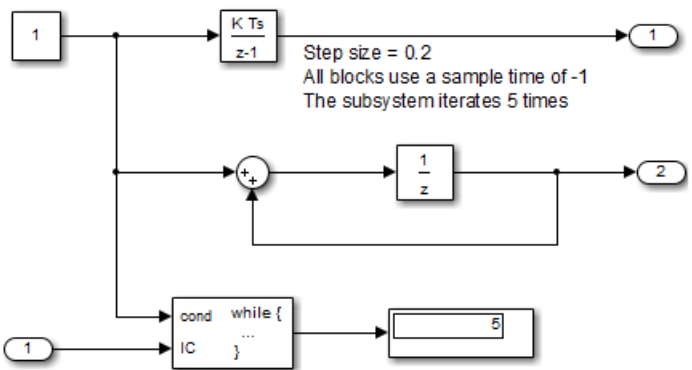
hisl_0006: Usage of While Iterator blocks

ID: Title	hisl_0006: Usage of While Iterator blocks	
Description	To support bounded iterative behavior in the generated code when using the While Iterator block, in the While Iterator block parameters dialog box:	
	A	Set Maximum number of iterations to a positive integer value; do not set value to -1 for unlimited.
	B	Consider selecting Show iteration number port to observe the iteration value during simulation.
Note	<p>When you use While Iterator subsystems, set the maximum number of iterations. If you use an unlimited number of iterations, the generated code might include infinite loops, which lead to execution-time overruns.</p> <p>To observe the iteration value during simulation and determine whether the loop reaches the maximum number of iterations, select the While Iterator block parameter Show iteration number port. If the loop reaches the maximum number of iterations, verify the output values of the While Iterator block.</p>	
Rationale	A, B	Support bounded iterative in the generated code.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for EN 50128 > Check usage of Ports and Subsystems blocks <p>For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Ports and Subsystems blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' 	

ID: Title	hisl_0006: Usage of While Iterator blocks
	ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' <ul style="list-style-type: none">• EN 50128, Table A.4 (11) 'Language Subset'• EN 50128, Table A.3 (1) 'Defensive Programming'• DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards'• DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'• MISRA-C:2004, Rule 21.1
Last Changed	R2013b

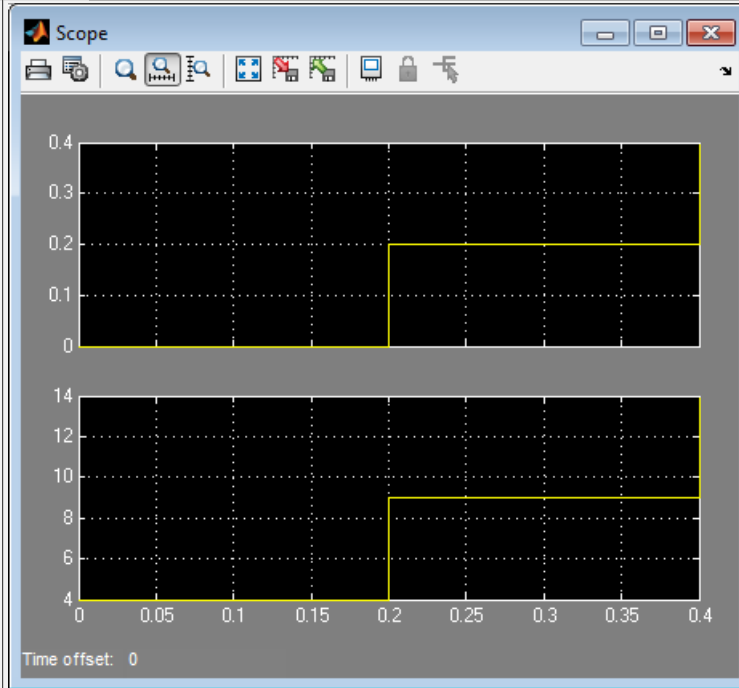
hisl_0007: Usage of While Iterator subsystems

ID: Title	hisl_0007: Usage of While Iterator subsystems	
Description	To support unambiguous behavior, when using While Iterator subsystems,	
	A	Specify inherited (-1) or constant (inf) sample times for the blocks within the subsystems.
	B	Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems.
Rationale	A, B	Avoid ambiguous behavior from the subsystem.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for EN 50128 > Check usage of Ports and Subsystems blocks <p>For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Ports and Subsystems blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' • DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' • MISRA-C:2004, Rule 21.1 	
Last Changed	R2013b	

ID: Title	hisl_0007: Usage of While Iterator subsystems
Examples	<p>For iterative subsystems, the value <code>delta T</code> is nonzero for the first iteration only. For subsequent iterations, the value is zero.</p> <p>In the following example, in the output of the Sum block calculation that uses the unit delay, the Sum block calculation does not require <code>delta T</code>. The output of the Discrete-Time Integrator block displays the result of having a zero <code>delta T</code> value.</p>  <p>The diagram illustrates a Simulink model within a while loop. The loop is controlled by a 'cond while' block with an initial condition (IC) and a final condition of 5. Inside the loop, the input signal splits into three paths: <ul style="list-style-type: none"> The top path goes through a block labeled $\frac{K T_s}{z-1}$ to an output port labeled '1'. The middle path goes through a summing junction (+) and then a discrete-time integrator block labeled $\frac{1}{z}$. The output of the integrator is fed back into the summing junction and also goes to an output port labeled '2'. The bottom path goes to a display block showing the value 5. Text annotations specify: 'Step size = 0.2', 'All blocks use a sample time of -1', and 'The subsystem iterates 5 times'. </p>

ID: Title

hisl_0007: Usage of While Iterator subsystems



hisl_0008: Usage of For Iterator Blocks

ID: Title	hisl_0008: Usage of For Iterator blocks	
Description	To support bounded iterative behavior in the generated code when using the For Iterator block, do one of the following:	
	A	In the For Iterator block parameters dialog box, set Iteration limit source to internal .
	B	If Iteration limit source must be external , use a block that has a constant value, such as a Width, Probe, or Constant.
	C	In the For Iterator block parameters dialog box, clear Set next i (iteration variable) externally .
	D	In the For Iterator block parameters dialog box, consider selecting Show iteration variable to observe the iteration value during simulation.
Notes	When you use the For Iterator block, feed the loop control variable with fixed (nonvariable) values to get a predictable number of loop iterations. Otherwise, a loop can result in unpredictable execution times and, in the case of external iteration variables, infinite loops that can lead to execution-time overruns.	
Rationale	A, B, C, D	Support bounded iterative behavior in generated code.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for EN 50128 > Check usage of Ports and Subsystems blocks <p>For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Ports and Subsystems blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' 	

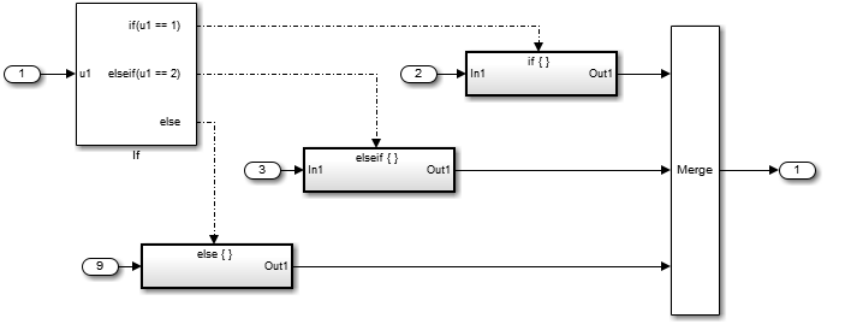
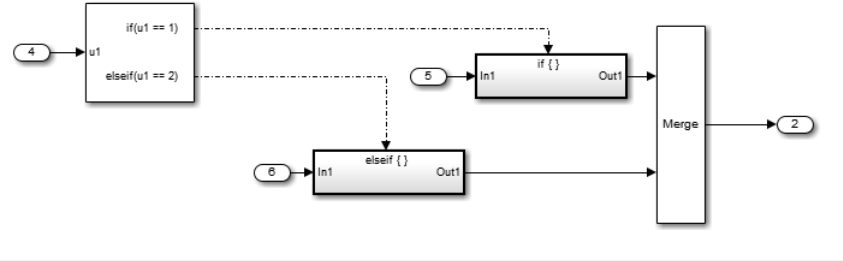
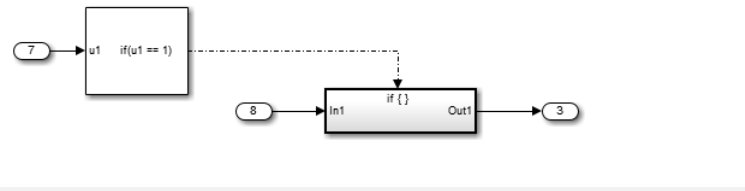
ID: Title	hisl_0008: Usage of For Iterator blocks
	IEC 61508-3, Table A.4 (3) 'Defensive programming' <ul style="list-style-type: none">• ISO 26262-6, Table 1 (b) 'Use of language subsets'ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques'• EN 50128, Table A.4 (11) 'Language Subset'EN 50128, Table A.3 (1) 'Defensive Programming'• DO-331, MB.Section 6.3.1.e 'High-level requirements conform to standards'DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'• MISRA-C:2004, Rule 13.6
Last Changed	R2013b

hisl_0009: Usage of For Iterator Subsystem blocks

ID: Title	hisl_0009: Usage of For Iterator Subsystem blocks	
Description	To support unambiguous behavior, when using the For Iterator Subsystem block,	
	A	Specify inherited (-1) or constant (inf) sample times for blocks within the subsystem.
	B	Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystem.
Rationale	A, B	Avoid ambiguous behavior from the subsystem.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Ports and Subsystems blocks • By Task > Modeling Standards for EN 50128 > Check usage of Ports and Subsystems blocks <p>For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Ports and Subsystems blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset'; IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 13.6 	
Last Changed	R2013b	
Examples	See “hisl_0007: Usage of While Iterator subsystems” on page 2-23.	

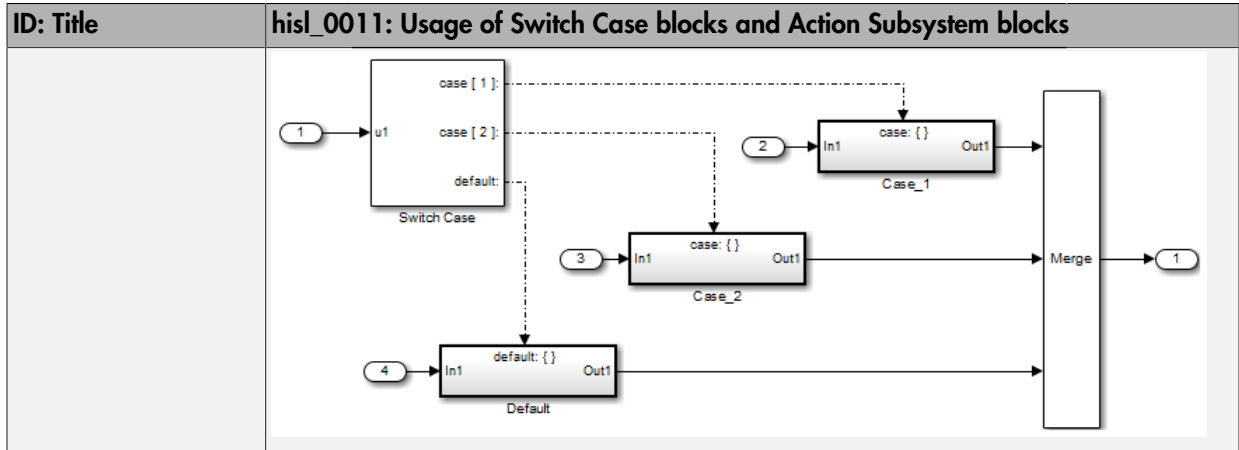
hisl_0010: Usage of If blocks and If Action Subsystem blocks

ID: Title	hisl_0010: Usage of If blocks and If Action Subsystem blocks	
Description	To support verifiable generated code, when using the If block with nonempty <code>Elseif</code> expressions,	
	A	In the block parameter dialog box, select Show else condition .
	B	Connect the outports of the If block to If Action Subsystem blocks.
Prerequisites	"hisl_0016: Usage of blocks that compute relational operators" on page 2-52	
Notes	The combination of If and If Action Subsystem blocks enable conditional execution based on input conditions. When there is only an <code>if</code> branch, you do not need to include an <code>else</code> branch.	
Rationale	A, B	Support generation of verifiable code.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA-C:2004, Rule 14.10 	
See Also	na_0012: Use of Switch vs. If-Then-Else Action Subsystem in the Simulink documentation	
Last Changed	R2013b	

ID: Title	hisl_0010: Usage of If blocks and If Action Subsystem blocks
Examples	 <p>Recommended: Elseif with Else</p>  <p>Not Recommended: No Else Path</p>  <p>Recommended: Only an If, no Else required</p>

hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks

ID: Title	hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks	
Description	To support verifiable generated code, when using the Switch Case block:	
	A	In the Switch Case block parameter dialog box, select Show default case .
	B	Connect the outports of the Switch Case block to a Switch Case Action Subsystem block.
	C	Use an integer data type or an enumeration value for the inputs to Switch Case blocks.
Prerequisites	“hisl_0016: Usage of blocks that compute relational operators” on page 2-52	
Notes	The combination of Switch Case and If Action Subsystem blocks enable conditional execution based on input conditions. Provide a default path of execution in the form of a “Default” block.	
Rationale	A, B, C	Support generation of verifiable code.
References	<ul style="list-style-type: none"> IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' ISO 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' MISRA-C:2004, Rule 15.3 	
See Also	db_0115: Simulink patterns for case constructs in the Simulink documentation.	
Last Changed	R2014b	
Examples	The following graphic displays an example of providing a default path of execution using a “Default” block.	



hisl_0012: Usage of conditionally executed subsystems

ID: Title	hisl_0012: Usage of conditionally executed subsystems	
Description	To support unambiguous behavior, when using conditionally executed subsystems:	
	A	Specify inherited (-1) sample times for all blocks in the subsystem, except Constant. Constant blocks can use infinite (<code>inf</code>) sample time.
	B	If the subsystem is called asynchronously, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystem.
Notes	<p>Conditionally executed subsystems include:</p> <ul style="list-style-type: none"> • If Action • Switch Case Action • Function-Call • Triggered • Enabled <p>Sample time-dependent blocks include:</p> <ul style="list-style-type: none"> • Discrete State-Space • Discrete-Time Integrator • Discrete FIR Filter • Discrete Filter • Discrete Transfer Fcn • Discrete Zero-Pole • Transfer Fcn First Order • Transfer Fcn Real Zero • Transfer Fcn Lead or Lag 	
Rationale	A, B	Support unambiguous behavior.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' 	

ID: Title	hisl_0012: Usage of conditionally executed subsystems
	<ul style="list-style-type: none">• EN 50128, Table A.4 (11) 'Language Subset'• EN 50128, Table A.3 (1) 'Defensive Programming'
Last Changed	R2013b
Examples	When using discrete blocks, the behavior depends on the operation across multiple contiguous time steps. When the blocks are called intermittently, the results may not conform to your expectations.

hisl_0024: Inport interface definition

ID: Title	hisl_0024: Inport interface definition
Description	<p>To support strong data typing and unambiguous behavior of the model and the generated code, for each root-level Inport block, explicitly set the following block parameters:</p> <ul style="list-style-type: none"> • Data type • Port dimensions • Sample time
Note	<p>Using root-level Inport blocks without fully defined dimensions, sample times, or data type can lead to ambiguous simulation results. If you do not explicitly define these parameters, Simulink back-propagates dimensions, sample times, and data types from downstream blocks.</p>
Rationale	<ul style="list-style-type: none"> • Avoid unambiguous behavior. • Support full specification of software interface.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for IEC 61508 > Check for root Inports with missing properties • By Task > Modeling Standards for ISO 26262 > Check for root Inports with missing properties • By Task > Modeling Standards for EN 50128 > Check for root Inports with missing properties <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check for root Inports with missing properties”.</p>
References	<ul style="list-style-type: none"> • IEC 61508-3, Table B.9 (5) ‘Fully defined interface’ • ISO 26262-4, Table 2 (2) ‘Precisely defined interfaces’ • ISO 26262-6, Table 1 (1f) ‘Use of unambiguous graphical representation’ • EN 50128, Table A.3 (19) ‘Fully Defined Interface’
Last Changed	R2014b

hisl_0025: Design min/max specification of input interfaces

ID: Title	hisl_0025: Design min/max specification of input interfaces
Description	Provide design min/max information for root-level Inport blocks to specify the input interface ranges.
Notes	<ul style="list-style-type: none"> • Specifying the range of Inport blocks on the root level enables additional capabilities^a. Examples include: <ul style="list-style-type: none"> • Detection of overflows through simulation range checking. • Code optimizations using Embedded Coder. • Design model verification using Simulink Design Verifier™. • Fixed-point autoscaling using Fixed-Point Designer™. • Specified design ranges can be used by Embedded Coder to optimize the generated code. If you want to use design ranges for optimization, in the Configuration Parameters dialog box, on the Code Generation pane, consider selecting Optimize using the specified minimum and maximum values. • Ranges for bus-type Inport blocks are specified with the bus elements of the defining bus object. Simulink ignores range specifications provided directly at Inport blocks that are bus-type.
Rationale	Support precise specification of the input interface.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for IEC 61508 > Check for fully defined interface range (Inports) • By Task > Modeling Standards for ISO 26262 > Check for fully defined interface range (Inports) • By Task > Modeling Standards for EN 50128 > Check for fully defined interface range (Inports) <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check for root Inports with missing range definitions”.</p>
References	<ul style="list-style-type: none"> • IEC 61508-3, Table B.9 (5) ‘Fully defined interface’ • ISO 26262-4, Table 2 (2) ‘Precisely defined interfaces’ • EN 50128, Table A.1(11) – Software Interface Specifications, Table A.3 (19) ‘Fully Defined Interface’

ID: Title	hisl_0025: Design min/max specification of input interfaces
Last Changed	R2013b

- a. These capabilities leverage design range information for different purposes. For more information, refer to the documentation for the tools you intend to use.

hisl_0026: Design min/max specification of output interfaces

ID: Title	hisl_0026: Design min/max specification of output interfaces
Description	Provide design min/max information for root-level Outport blocks to specify the output interface ranges.
Notes	<ul style="list-style-type: none"> • Specifying the range of Outport blocks on the root level enables additional capabilities^a. Examples include: <ul style="list-style-type: none"> • Detection of overflows through simulation range checking. • Code optimizations using Embedded Coder. • Design model verification using Simulink Design Verifier. • Fixed-point autoscaling using Fixed-Point Designer. • Specified design ranges can be used by Embedded Coder to optimize the generated code. If you want to use design ranges for optimization, in the Configuration Parameters dialog box, on the Code Generation pane, consider selecting Optimize using the specified minimum and maximum values. • Ranges for bus-type Outport blocks are specified with the bus elements of the defining bus object. Simulink ignores range specifications provided directly at Outport blocks that are bus-type.
Rationale	Support precise specification of the output interface.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for IEC 61508 > Check for fully defined interface range (Outports) • By Task > Modeling Standards for ISO 26262 > Check for fully defined interface range (Outports) • By Task > Modeling Standards for EN 50128 > Check for fully defined interface range (Outports) <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check for root Outports with missing range definitions”.</p>
References	<ul style="list-style-type: none"> • IEC 61508-3, Table B.9 (5) ‘Fully defined interface’ • ISO 26262-4, Table 2 (2) ‘Precisely defined interfaces’ • EN 50128, Table A.1(11) – Software Interface Specifications, Table A.3 (19) ‘Fully Defined Interface’

ID: Title	hisl_0026: Design min/max specification of output interfaces
Last Changed	R2013b

- a. These capabilities leverage design range information for different purposes. For more information, refer to the documentation for the tools you intend to use.

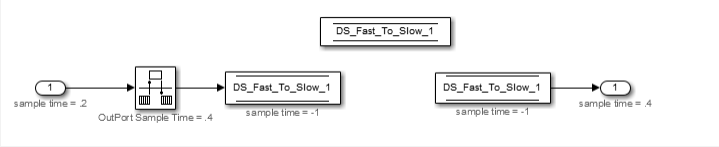
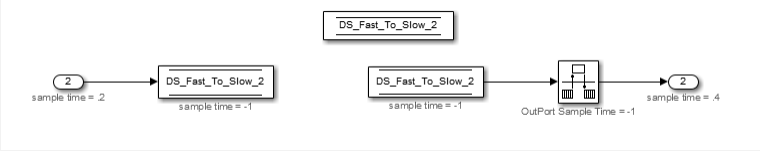
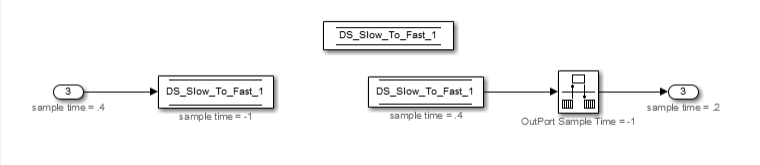
Signal Routing

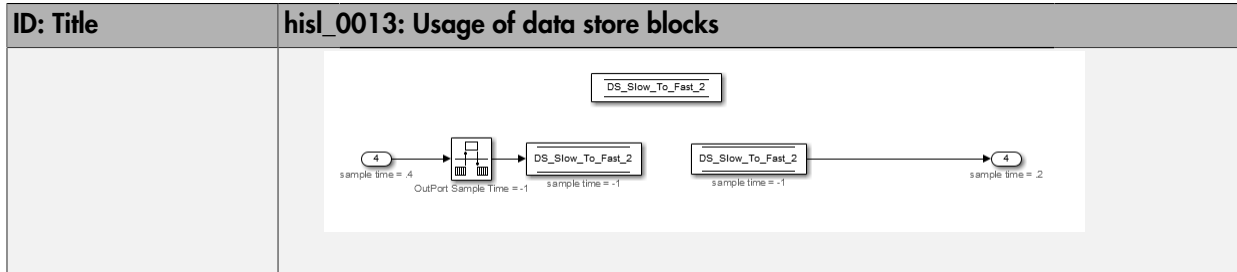
In this section...
“hisl_0013: Usage of data store blocks” on page 2-41
“hisl_0015: Usage of Merge blocks” on page 2-45
“hisl_0021: Consistent vector indexing method” on page 2-47
“hisl_0022: Data type selection for index signals” on page 2-49
“hisl_0023: Verification of model and subsystem variants” on page 2-50

hisl_0013: Usage of data store blocks

ID: Title	hisl_0013: Usage of data store blocks	
Description	To support deterministic behavior across different sample times or models when using data store blocks, including Data Store Memory, Data Store Read, and Data Store Write:	
	A	In the Configuration Parameters dialog box, on the Diagnostics > Data Validity pane, under Data Store Memory Block , set the following parameters to error : <ul style="list-style-type: none"> • Detect read before write • Detect write after read • Detect write after write • Multitask data store • Duplicate data store names
	B	Avoid data store reads and writes that occur across model and atomic subsystem boundaries.
Notes	C	Avoid using data stores to write and read data at different rates, because different rates can result in inconsistent exchanges of data. To provide deterministic data coupling in multirate systems, use Rate Transition blocks before Data Store Write blocks, or after Data Store Read blocks.
	<p>The sorting algorithm in Simulink does not take into account data coupling between models and atomic subsystems.</p> <p>Using data store memory blocks can have significant impact on your software verification effort. Models and subsystems that use only inports and outports to pass data provide a directly traceable interface, simplifying the verification process.</p>	
Rationale	A, B, C	Support consistent data values across different sample times or models.
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for data store memory	

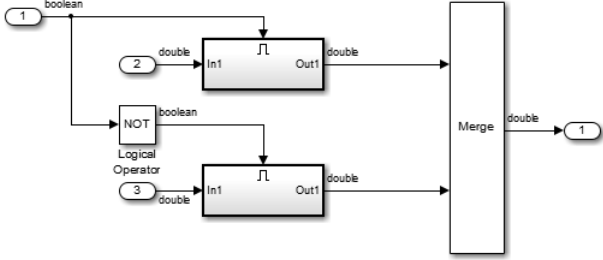
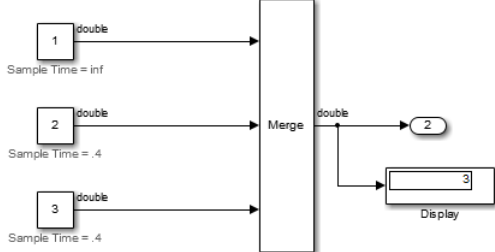
ID: Title	hisl_0013: Usage of data store blocks
	For check details, see “Check safety-related diagnostic settings for data store memory”.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.3.b 'Software architecture is consistent'
Last Changed	R2013b

ID: Title	hisl_0013: Usage of data store blocks
Examples	<p>The following examples use Rate Transition blocks to provide deterministic data coupling in multirate systems</p> <ul style="list-style-type: none"> For fast-to-slow transitions: <p>Set the rate of the slow sample time on either the Rate Transition block or the Data Store Write block.</p>  <p>Do not place the Rate Transition block after the Data Store Read block.</p>  For slow-to-fast transitions: <p>If the Rate Transition block is after the Data Store Read block, specify the slow rate on the Data Store Read block.</p>  <p>If the Rate Transition block is before the Data Store Write block, use the inherited sample time for the blocks.</p>



hisl_0015: Usage of Merge blocks

ID: Title	hisl_0015: Usage of Merge blocks	
Description	To support unambiguous behavior from Merge blocks,	
	A	Use Merge blocks only with conditionally executed subsystems.
	B	Specify execution of the conditionally executed subsystems such that only one subsystem executes during a time step.
	C	Clear the Merge block parameter Allow unequal port widths .
Notes	<p>Simulink combines the inputs of the Merge block into a single output. The output value at any time is equal to the most recently computed output of the blocks that drive the Merge block. Therefore, the Merge block output is dependent upon the execution order of the input computations.</p> <p>To provide predictable behavior of the Merge block output, you must have mutual exclusion between the conditionally executed subsystems feeding a Merge block. If the inputs are not mutually exclusive, Simulink uses the last input port.</p>	
Rationale	A, B, C	Avoid unambiguous behavior.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.3.b 'Software architecture is consistent' 	
Last Changed	R2013b	

ID: Title	hisl_0015: Usage of Merge blocks
Examples	<div data-bbox="452 326 1053 633" style="border: 1px solid black; padding: 10px; margin-bottom: 10px;"> <p style="text-align: center;">Recommended</p>  </div> <div data-bbox="372 690 546 720" style="text-align: center;"> <p>Recommended</p> </div> <div data-bbox="486 784 979 1076" style="border: 1px solid black; padding: 10px; margin-bottom: 10px;"> <p style="text-align: center;">Not Recommended</p>  </div> <div data-bbox="372 1135 593 1164" style="text-align: center;"> <p>Not Recommended</p> </div>

hisl_0021: Consistent vector indexing method

ID: Title	hisl_0021: Consistent vector indexing method			
Description	<p>Within a model, use:</p> <table border="1" data-bbox="372 409 1337 713"> <tr> <td data-bbox="372 409 442 713">A</td> <td data-bbox="442 409 1337 713"> <p>A consistent vector indexing method for all blocks. Blocks for which you should set the indexing method include:</p> <ul style="list-style-type: none"> • Index Vector • Multiport Switch • Assignment • Selector • For Iterator </td> </tr> </table>		A	<p>A consistent vector indexing method for all blocks. Blocks for which you should set the indexing method include:</p> <ul style="list-style-type: none"> • Index Vector • Multiport Switch • Assignment • Selector • For Iterator
A	<p>A consistent vector indexing method for all blocks. Blocks for which you should set the indexing method include:</p> <ul style="list-style-type: none"> • Index Vector • Multiport Switch • Assignment • Selector • For Iterator 			
Rationale	A	Reduce the risk of introducing errors due to inconsistent indexing.		
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check for inconsistent vector indexing methods • By Task > Modeling Standards for IEC 61508 > Check for inconsistent vector indexing methods • By Task > Modeling Standards for ISO 26262 > Check for inconsistent vector indexing methods • By Task > Modeling Standards for EN 50128 > Check for inconsistent vector indexing methods <p>For DO-178C/DO-331 check details, see “Check for inconsistent vector indexing methods”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check for inconsistent vector indexing methods”.</p>			
References	<ul style="list-style-type: none"> • IEC 61508–3, Table A.3 (3) 'Language subset' • IEC 61508–3, Table A.4 (5) 'Design and coding standards' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (f) 'Use of unambiguous graphical representation' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.12 (1) 'Coding Standard' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' 			

ID: Title	hisl_0021: Consistent vector indexing method
See Also	“cgsl_0101: Zero-based indexing”
Last Changed	R2013b

hisl_0022: Data type selection for index signals

ID: Title	hisl_0022: Data type selection for index signals	
Description	For index signals, use:	
	A	An integer or enumerated data type
	B	A data type that covers the range of indexed values.
Rationale	A	Prevent unexpected results that can occur with rounding operations for floating-point data types.
References	B	Enable access to data in a vector.
Last Changed	R2013b	

hisl_0023: Verification of model and subsystem variants

ID: Title	hisl_0023: Verification of model and subsystem variants	
Description	When verifying that a model is consistent with generated code, do one of the following:	
	A	In the Configuration Parameters dialog box, on the Code Generation > Interface pane, disable variants in generated code by setting Generate preprocessor conditionals to Disable all .
	B	Verify all combinations of model variants that might be active in the generated code.
Rationale	A	Simplify consistency testing between the model and generated code by restricting the code base to a single variant.
	B	Make sure that consistency testing between the model and generated code is complete for all variants.
References	<ul style="list-style-type: none"> • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table A.4 (7) 'Use of trusted / verified software modules and components' 	
Last Changed	R2012b	

Logic and Bit Operations

In this section...
“hisl_0016: Usage of blocks that compute relational operators” on page 2-52
“hisl_0017: Usage of blocks that compute relational operators (2)” on page 2-54
“hisl_0018: Usage of Logical Operator block” on page 2-55
“hisl_0019: Usage of Bitwise Operator block” on page 2-57

hisl_0016: Usage of blocks that compute relational operators

ID: Title	hisl_0016: Usage of blocks that compute relational operators	
Description	To support the robustness of the operations, when using blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare to Zero, and Detect Change	
	A	Avoid comparisons using the == or ~= operator on floating-point data types.
Notes	<p>Due to floating-point precision issues, do not test floating-point expressions for equality (==) or inequality (~=).</p> <p>When the model contains a block computing a relational operator with the == or ~= operators, the inputs to the block must not be single, double, or any custom storage class that is a floating-point type. Change the data type of the input signals, or rework the model to eliminate using the == or ~= operators within blocks that compute relational operators.</p>	
Rationale	A	Improve model robustness.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for EN 50128 > Check usage of Logic and Bit Operations blocks <p>For DO-178C/DO-331 check details, see “Check usage of Logic and Bit Operations blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Logic and Bit Operations blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' 	

ID: Title	hisl_0016: Usage of blocks that compute relational operators
	EN 50128, Table A.3 (1) 'Defensive Programming' <ul style="list-style-type: none"> • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 13.3
See Also	“hisl_0017: Usage of blocks that compute relational operators (2)” on page 2-54
Last Changed	R2014b
Examples	<p>Positive Pattern: To test whether two floating-point variables or expressions are equal, compare the difference of the two variables against a threshold that takes into account the floating-point relative accuracy (ϵ) and the magnitude of the numbers.</p> <p>The following pattern shows how to test two double-precision input signals, In1 and In2, for equality.</p> <div data-bbox="372 829 1326 1109" style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre> graph LR In1((1)) -- double --> Subtract[Subtract] In2((2)) -- double --> Subtract Subtract -- double --> Abs[Abs] threshold[threshold Constant] -- double --> Abs Abs -- double --> RelOp[Relational Operator] threshold -- double --> RelOp RelOp -- boolean --> Out1((1 Out1)) </pre> </div>

hisl_0017: Usage of blocks that compute relational operators (2)

ID: Title	hisl_0017: Usage of blocks that compute relational operators (2)	
Description	To support unambiguous behavior in the generated code, when using blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare to Zero, and Detect Change	
	A	Set the block Output data type parameter to Boolean .
Rationale	A	Support generation of code that produces unambiguous behavior.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for IEC 61508 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for EN 50128 > Check usage of Logic and Bit Operations blocks <p>For DO-178C/DO-331 check details, see “Check usage of Logic and Bit Operations blocks”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Logic and Bit Operations blocks”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset'; • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 12.6 	
See Also	“hisl_0016: Usage of blocks that compute relational operators” on page 2-52	
Last Changed	R2013b	

hisl_0018: Usage of Logical Operator block

ID: Title	hisl_0018: Usage of Logical Operator block	
Description	To support unambiguous behavior of generated code, when using the Logical Operator block,	
	A	Set the Output data type block parameter to Boolean .
Prerequisites	“hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)” on page 5-25	
Rationale	A	Avoid ambiguous behavior of generated code.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for IEC 61508 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for ISO 26262 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for EN 50128 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for DO-178C/DO-331 > Check usage of Logic and Bit Operations blocks • By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related optimization settings <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Logic and Bit Operations blocks”.</p> <p>For DO-178C/DO-331 check details, see “Check usage of Logic and Bit Operations blocks” or “Check safety-related optimization settings”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 12.6 	
Last Changed	R2013b	

hisl_0019: Usage of Bitwise Operator block

ID: Title	hisl_0019: Usage of Bitwise Operator block	
Description	To support unambiguous behavior, when using the Bitwise Operator block,	
	A	Avoid signed integer data types as input to the block.
	B	Choose an output data type that represents zero exactly.
Notes	Bitwise operations on signed integers are not meaningful. If a shift operation moves a signed bit into a numeric bit, or a numeric bit into a signed bit, unpredictable and unwanted behavior can result.	
Rationale	A, B	Support unambiguous behavior of generated code.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • MISRA-C:2004, Rule 12.7 	
See Also	"hisf_0003: Usage of bitwise operations" on page 3-12 in the Simulink documentation	
Last Changed	R2013b	

Stateflow Chart Considerations

- “Chart Properties” on page 3-2
- “Chart Architecture” on page 3-11

Chart Properties

In this section...
“hisf_0001: Mealy and Moore semantics” on page 3-3
“hisf_0002: User-specified state/transition execution order” on page 3-5
“hisf_0009: Strong data typing (Simulink and Stateflow boundary)” on page 3-7
“hisf_0011: Stateflow debugging settings” on page 3-9

hisf_0001: Mealy and Moore semantics

ID: Title	hisf_0001: Mealy and Moore semantics	
Description	To create Stateflow charts that implement a subset of Stateflow semantics,	
	A	In the Chart properties dialog box, set State Machine Type to Mealy or Moore .
	B	Apply consistent settings to the Stateflow charts in a model.
Note	<p>Setting State Machine Type restricts the Stateflow semantics to pure Mealy or Moore semantics. Mealy and Moore charts might be easier to understand and use in high-integrity applications.</p> <p>In Mealy charts, actions are associated with transitions. In the Moore charts, actions are associated with states.</p> <p>At compile time, the Stateflow software verifies that the chart semantics comply with the formal definitions and rules of the selected type of state machine. If the chart semantics are not in compliance, the software provides a diagnostic message.</p>	
Rationale	A, B	Promote a clear modeling style.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check state machine type of Stateflow charts • By Task > Modeling Standards for IEC 61508 > Check state machine type of Stateflow charts • By Task > Modeling Standards for ISO 26262 > Check state machine type of Stateflow charts • By Task > Modeling Standards for EN 50128 > Check state machine type of Stateflow charts <p>For DO-178C/DO-331 check details, see “Check state machine type of Stateflow charts”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check state machine type of Stateflow charts”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.7 (2) 'Simulation/modeling' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' 	

ID: Title	hisf_0001: Mealy and Moore semantics
	EN 50128, Table A.11 (3) 'Simulation' <ul style="list-style-type: none">• DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent'DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards'DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'DO-331, Section MB.6.3.3.b 'Software architecture is consistent'DO-331, Section MB.6.3.3.e 'Software architecture conform to standards'
See Also	“Create Mealy and Moore Charts” in the Stateflow documentation
Last Changed	R2013b

hisf_0002: User-specified state/transition execution order

ID: Title	hisf_0002: User-specified state/transition execution order	
Description	Do the following to explicitly set the execution order for active states and valid transitions in Stateflow charts:	
	A	In the Chart Properties dialog box, select User specified state/transition execution order .
	B	In the Stateflow Editor View menu, select Show Transition Execution Order .
	C	Set default transition to evaluate last.
Note	<p>Selecting User specified state/transition execution order restricts the dependency of a Stateflow chart semantics on the geometric position of parallel states and transitions.</p> <p>Specifying the execution order of states and transitions allows you to enforce determinism in the search order for active states and valid transitions. You have control of the order in which parallel states are executed and transitions originating from a source are tested for execution. If you do not explicitly set the execution order, the Stateflow software determines the execution order following a deterministic algorithm.</p> <p>Selecting Show Transition Execution Order displays the transition testing order.</p>	
Rationale	A, B, C	Promote an unambiguous modeling style.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check Stateflow charts for ordering of states and transitions • By Task > Modeling Standards for IEC 61508 > Check usage of Stateflow constructs • By Task > Modeling Standards for ISO 26262 > Check usage of Stateflow constructs • By Task > Modeling Standards for EN 50128 > Check usage of Stateflow constructs <p>For DO-178C/DO-331 check details, see “Check Stateflow charts for ordering of states and transitions”.</p>	

ID: Title	hisf_0002: User-specified state/transition execution order
	For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Stateflow constructs”.
References	This guideline supports adhering to: <ul style="list-style-type: none">• IEC 61508-3, Table A.3 (3) 'Language subset'• ISO 26262-6, Table 1 (b) 'Use of language subsets'• ISO 26262-6, Table 1 (f) 'Use of unambiguous graphical representation'• EN 50128, Table A.4 (11) 'Language Subset'• DO-331, Section MB.6.3.3.b 'Software architecture is consistent'• DO-331, Section MB.6.3.3.e 'Software architecture conform to standards '
See Also	The following topics in the Stateflow documentation <ul style="list-style-type: none">• “Transition Testing Order in Multilevel State Hierarchy”• “Execution Order for Parallel States”
Last Changed	R2013b

hisf_0009: Strong data typing (Simulink and Stateflow boundary)

ID: Title	hisf_0009: Strong data typing (Simulink and Stateflow boundary)	
Description	To support strong data typing between Simulink and Stateflow ,	
	A	Select Use Strong Data Typing with Simulink I/O .
Notes	By default, input to and output from Stateflow charts are of type double . To interface directly with Simulink signals of data types other than double , select Use Strong Data Typing with Simulink I/O . In this mode, data types between the Simulink and Stateflow boundary are strongly typed, and the Simulink software does not treat the data types as double . The Stateflow chart accepts input signals of any data type supported by the Simulink software, provided that the type of the input signal matches the type of the corresponding Stateflow input data object. Otherwise, the software reports a type mismatch error.	
Rationale	A	Support strongly typed code.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for IEC 61508 > Check usage of Stateflow constructs • By Task > Modeling Standards for ISO 26262 > Check usage of Stateflow constructs • By Task > Modeling Standards for EN 50128 > Check usage of Stateflow constructs <p>For check details, see “Check usage of Stateflow constructs”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' • DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' 	

ID: Title	hisf_0009: Strong data typing (Simulink and Stateflow boundary)
	• MISRA-C:2004, Rules 10.1, 10.2, 10.3 and 10.4
Last Changed	R2013b

hisf_0011: Stateflow debugging settings

ID: Title	hisf_0011: Stateflow debugging settings					
Description	<p>To protect against unreachable code and indeterminate execution time,</p> <table border="1" data-bbox="372 413 1337 857"> <tr> <td data-bbox="372 413 454 718">A</td> <td data-bbox="458 413 1337 718"> <p>Select the following run-time diagnostics:</p> <ul style="list-style-type: none"> • In the Configuration Parameters dialog box, on the Simulation Target pane, select: Detect wrap on overflow • In the Stateflow Debugging window, select Transition Conflict Data Range Detect Cycles </td> </tr> <tr> <td data-bbox="372 723 454 857">B</td> <td data-bbox="458 723 1337 857"> <p>For each truth table in the model, in the Settings menu of the Truth Table Editor, set the following parameters to Error: Underspecified Overspecified</p> </td> </tr> </table>		A	<p>Select the following run-time diagnostics:</p> <ul style="list-style-type: none"> • In the Configuration Parameters dialog box, on the Simulation Target pane, select: Detect wrap on overflow • In the Stateflow Debugging window, select Transition Conflict Data Range Detect Cycles 	B	<p>For each truth table in the model, in the Settings menu of the Truth Table Editor, set the following parameters to Error: Underspecified Overspecified</p>
A	<p>Select the following run-time diagnostics:</p> <ul style="list-style-type: none"> • In the Configuration Parameters dialog box, on the Simulation Target pane, select: Detect wrap on overflow • In the Stateflow Debugging window, select Transition Conflict Data Range Detect Cycles 					
B	<p>For each truth table in the model, in the Settings menu of the Truth Table Editor, set the following parameters to Error: Underspecified Overspecified</p>					
Notes	<p>Run-time diagnostics are only triggered during simulation. If the error condition is not reached during simulation, the error message is not triggered for code generation.</p>					
Rationale	<p>A, B Protect against unreachable code and unpredictable execution time.</p>					
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check Stateflow debugging settings • By Task > Modeling Standards for IEC 61508 > Check usage of Stateflow constructs • By Task > Modeling Standards for ISO 26262 > Check usage of Stateflow constructs • By Task > Modeling Standards for EN 50128 > Check usage of Stateflow constructs <p>For DO-178C/DO-331 check details, see “Check Stateflow debugging options”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Stateflow constructs”.</p>					
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.7 (2) 'Simulation/modeling' • ISO 26262 Table 1 (d) 'Use of defensive implementation techniques' 					

ID: Title	hisf_0011: Stateflow debugging settings
	<ul style="list-style-type: none"><li data-bbox="379 305 1338 361">• EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.11 (3) 'Simulation'<li data-bbox="379 368 1338 597">• DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'
Last Changed	R2015a

Chart Architecture

In this section...

“hisf_0003: Usage of bitwise operations” on page 3-12

“hisf_0004: Usage of recursive behavior” on page 3-13

“hisf_0007: Usage of junction conditions (maintaining mutual exclusion)” on page 3-15

“hisf_0010: Usage of transition paths (looping out of parent of source and destination objects)” on page 3-16

“hisf_0012: Chart comments” on page 3-18

“hisf_0013: Usage of transition paths (crossing parallel state boundaries)” on page 3-19

“hisf_0014: Usage of transition paths (passing through states)” on page 3-22

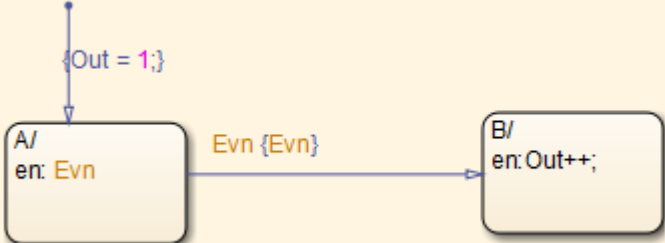
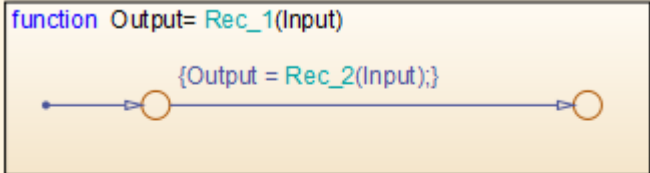
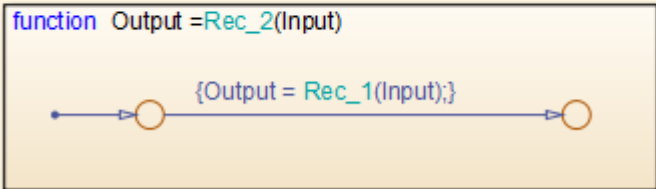
“hisf_0015: Strong data typing (casting variables and parameters in expressions)” on page 3-23

hisf_0003: Usage of bitwise operations

ID: Title	hisf_0003: Usage of bitwise operations	
Description	When using bitwise operations in Stateflow blocks,	
	A	Avoid signed integer data types as operands to the bitwise operations.
Notes	Normally, bitwise operations are not meaningful on signed integers. Undesired behavior can occur. For example, a shift operation might move the sign bit into the number, or a numeric bit into the sign bit.	
Rationale	A	Promote unambiguous modeling style.
Model Advisor Checks	<p>By Task > Modeling Standards for MAAB > Stateflow > Check for bitwise operations in Stateflow charts</p> <p>For check details, see “Check for bitwise operations in Stateflow charts”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' • DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' • DO-331, Section 6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 12.7 'Bitwise operators shall not be applied to operands whose underlying type is signed' 	
See Also	“hisl_0019: Usage of Bitwise Operator block”	
Last Changed	R2013b	

hisf_0004: Usage of recursive behavior

ID: Title	hisf_0004: Usage of recursive behavior	
Description	To support bounded function call behavior, avoid using design patterns that include unbounded recursive behavior. Recursive behavior is bound if you do the following:	
	A	Use an explicit termination condition that is local to the recursive call.
	B	Make sure the termination condition is reached.
Notes	This rule only applies if a chart is a classic Stateflow chart. If “hisf_0001: Mealy and Moore semantics” on page 3-3 is followed, recursive behavior is prevented due to restrictions in the chart semantics. Additionally, you can detect the error during simulation by enabling the Stateflow diagnostic Detect Cycles .	
Rationale	A, B	Promote bounded function call behavior.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table B.1 (6) 'Limited use of recursion' • ISO 26262-6, Table 9 (j) 'No recursions' • EN 50128, Table A.12 (6) 'Limited Use of Recursion' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 16.2 	
Last Changed	R2013b	
Examples	There are multiple patterns in Stateflow that can result in unbounded recursion.	

ID: Title	hisf_0004: Usage of recursive behavior
	
	<p>Recursive Function Calls</p>
	<p>When the default state A is entered, event EVN is broadcast in the entry action of A. EVN results in a recursive call of the interpretation algorithm. Since A is active, the outgoing transition of A is tested. Since the current event EVN matches the transition event (and because of the absence of condition) the condition action is executed, broadcasting EVN again. This results in a new call of the interpretation algorithm which repeats the same sequence of steps until stack overflow.</p>
	
	
	<p>Recursive Function Calls</p>

hisf_0007: Usage of junction conditions (maintaining mutual exclusion)

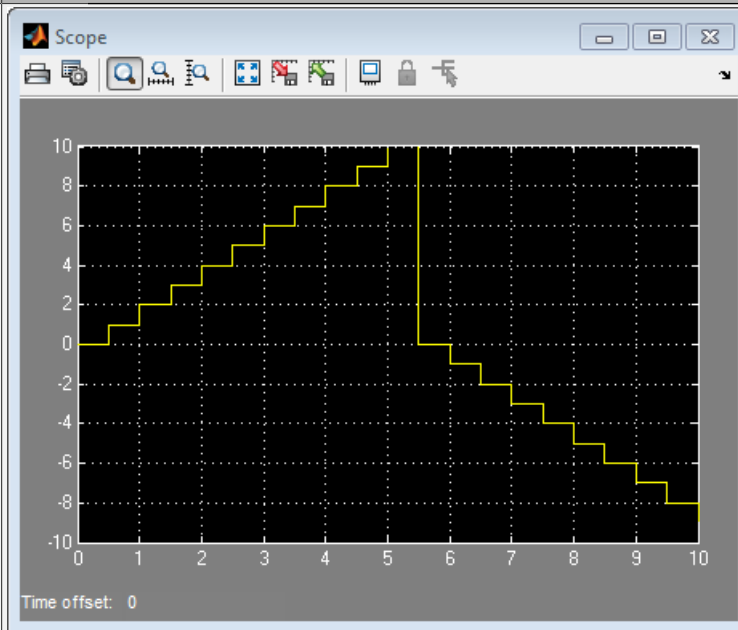
ID: Title	hisf_0007: Usage of junction conditions (maintaining mutual exclusion)
Description	To enhance clarity and prevent the generation of unreachable code,
	A Make junction conditions mutually exclusive.
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.
Rationale	A Enhance clarity and prevent generation of unreachable code.
References	<ul style="list-style-type: none"> • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.d 'High-level requirements are verifiable' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'
Last Changed	R2012b

hisf_0010: Usage of transition paths (looping out of parent of source and destination objects)

ID: Title	hisf_0010: Usage of transition paths (looping out of parent of source and destination objects)
Description	<p>Transitions that loop out of the parent of the source and destination objects are typically unintentional and cause the parent to deactivate.</p> <p>A Avoid using these transitions.</p>
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.
Rationale	A Promote a clear modeling style.
References	<ul style="list-style-type: none"> • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate'
Last Changed	R2012b
Examples	<p>The diagram illustrates a Stateflow chart with a parent state <code>A_Parent/</code> containing two sub-states: <code>A_sub_1/</code> and <code>A_sub_2/</code>. The parent state's event-driven (en) block contains the code <code>Out = 0;</code>. The <code>A_sub_1/</code> state's data-driven (du) block contains <code>Out++;</code>. The <code>A_sub_2/</code> state's data-driven (du) block contains <code>Out--;</code>. A transition path loops from the <code>A_sub_1/</code> state back to the <code>A_sub_2/</code> state, guarded by the condition <code>[Out >= 10]</code>. This transition path loops out of the parent state.</p>

ID: Title

hisf_0010: Usage of transition paths (looping out of parent of source and destination objects)

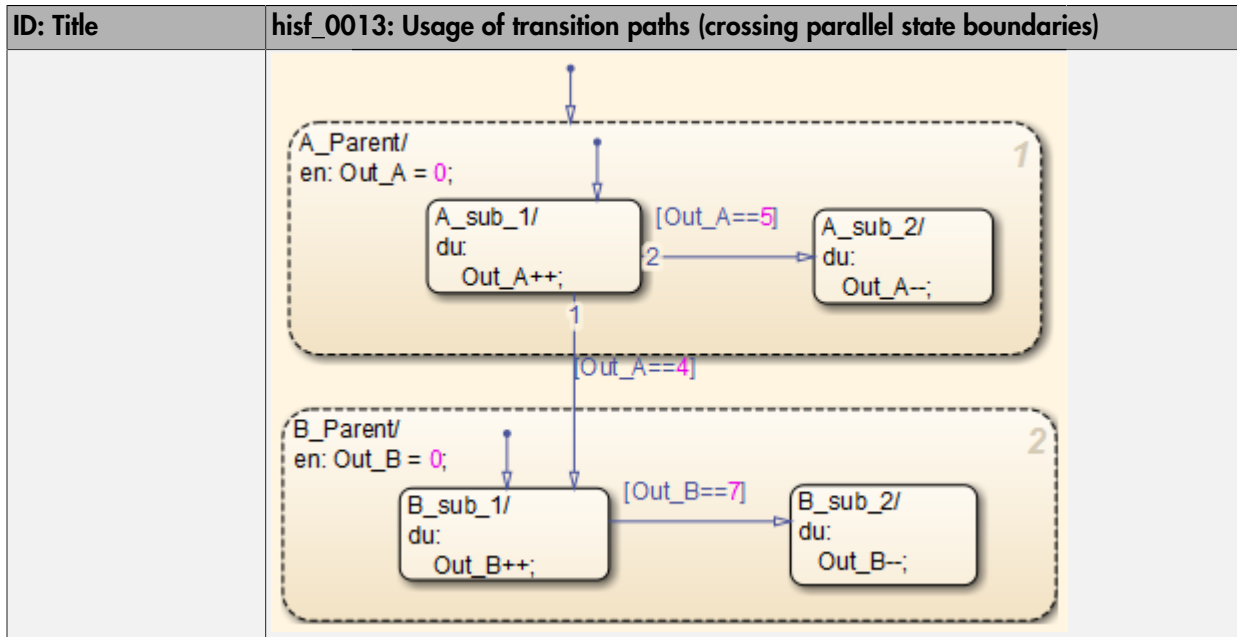


hisf_0012: Chart comments

ID: Title	hisf_0012: Chart comments	
Description	To enhance traceability between generated code and a model,	
	A	Add comments to the following Stateflow objects: <ul style="list-style-type: none">• Transitions
Rationale	A	Enhance traceability between generated code and the corresponding model.
References	• DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements'	
Last Changed	R2012b	

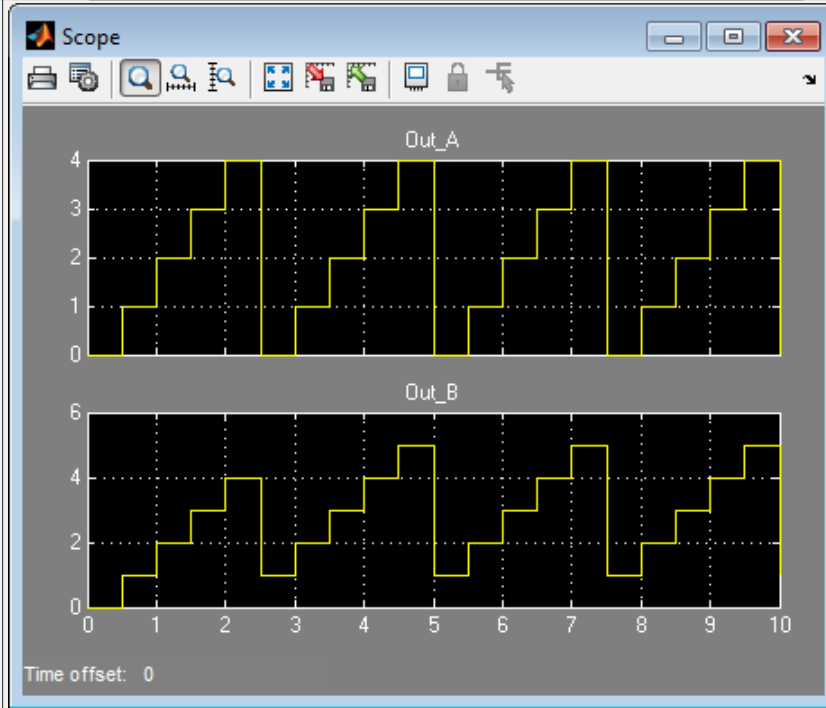
hisf_0013: Usage of transition paths (crossing parallel state boundaries)

ID: Title	hisf_0013: Usage of transition paths (crossing parallel state boundaries)	
Description	To avoid creating diagrams that are hard to understand,	
	A	Avoid creating transitions that cross from one parallel state to another.
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.	
Rationale	A	Enhance model readability.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' 	
Last Changed	R2014b	
Example	In the following example, when Out_A is 4, both parent states (A_Parent and B_Parent) are reentered. Reentering the parent states resets the values of Out_A and Out_B to zero.	

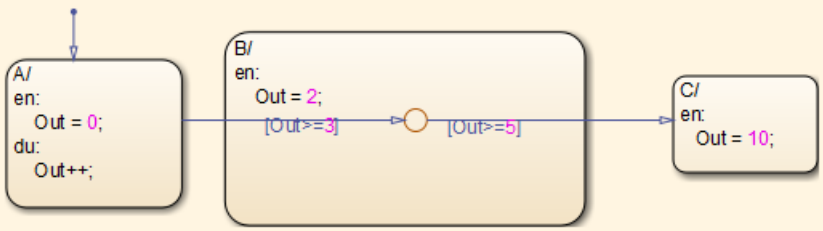


ID: Title

hisf_0013: Usage of transition paths (crossing parallel state boundaries)



hisf_0014: Usage of transition paths (passing through states)

ID: Title	hisf_0014: Usage of transition paths (passing through states)	
Description	To avoid creating diagrams that are confusing and include transition paths without benefit,	
	A	Avoid transition paths that go into and out of a state without ending on a substate.
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.	
Rationale	A	Enhance model readability.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' • DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' 	
Last Changed	R2014b	
Examples	 <p>The diagram illustrates a stateflow chart with three states: A, B, and C. State A is on the left, State B is in the middle, and State C is on the right. State A has an entry event (A/), an initialization 'Out = 0;', and a duration 'Out++;'. State B has an entry event (B/), an initialization 'Out = 2;', and a transition to state C with guard '[Out >= 3]'. State C has an entry event (C/), an initialization 'Out = 10;', and a transition back to state B with guard '[Out >= 5]'. The transition from B to C is marked with a small circle.</p>	

hisf_0015: Strong data typing (casting variables and parameters in expressions)

ID: Title	hisf_0015: Strong data typing (casting variables and parameters in expressions)	
Description	To facilitate strong data typing,	
	A	Explicitly type cast variables and parameters of different data types in: <ul style="list-style-type: none"> • Transition evaluations • Transition assignments • Assignments in states
Notes	The Stateflow software automatically casts variables of different type into the same data type. This guideline helps clarify data types of the intermediate variables.	
Rationale	A	Apply strong data typing.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' • DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rules 10.1, 10.2, 10.3 and 10.4 	
Last Changed	R2014b	

ID: Title	hisf_0015: Strong data typing (casting variables and parameters in expressions)
Examples	<div data-bbox="382 309 753 638"> <pre> stateDiagram-v2 [*] --> State_A/ State_A/ --> State_B/ : [uint16(uint_8_Var) < uint_16_Var] </pre> </div> <p data-bbox="373 673 546 708">Recommended</p> <div data-bbox="382 736 667 1013"> <pre> stateDiagram-v2 [*] --> State_A1/ State_A1/ --> State_B1/ : [int_8_Var < int_16_Var] </pre> </div> <p data-bbox="373 1048 593 1083">Not Recommended</p>

MATLAB Function and MATLAB Code Considerations

- “MATLAB Functions” on page 4-2
- “MATLAB Code” on page 4-11

MATLAB Functions

In this section...
“himl_0001: Usage of standardized MATLAB function headers” on page 4-3
“himl_0002: Strong data typing at MATLAB function boundaries” on page 4-4
“himl_0003: Limitation of MATLAB function complexity” on page 4-6
“himl_0005: Usage of global variables in MATLAB functions” on page 4-8

himl_0001: Usage of standardized MATLAB function headers

ID: Title	himl_0001: Usage of standardized MATLAB function headers
Description	When using MATLAB functions, use a standardized header to provide information about the purpose and use of the function.
Rationale	A standardized header improves the readability and documentation of MATLAB functions. The header should provide a function description and usage information.
See Also	<ul style="list-style-type: none"> • MathWorks Automotive Advisory Board (MAAB) guideline na_0025: MATLAB Function Header • Orion GN&C: MATLAB and Simulink Standards, jh_0073: eML Header • “MATLAB Function Block Editor”
Last Changed	R2014a
Examples	<p>A typical standardized function header includes:</p> <ul style="list-style-type: none"> • Function name • Description • Inputs and outputs (if possible, include size and type) • Assumptions and limitations • Revision history

himl_0002: Strong data typing at MATLAB function boundaries

ID: Title	himl_0002: Strong data typing at MATLAB function boundaries
Description	<p>To support strong data typing at the interfaces of MATLAB functions, explicitly define the interface for input signals, output signals, and parameters, by setting:</p> <ul style="list-style-type: none"> • Complexity • Type
Rationale	<p>Defined interfaces:</p> <ul style="list-style-type: none"> • Allow consistency checking of interfaces. • Prevent unintended generation of different functions for different input and output types. • Simplify testing of functions by limiting the number of test cases.
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check for MATLAB Function interfaces with inherited properties • By Task > Modeling Standards for IEC 61508 > Check for MATLAB Function interfaces with inherited properties • By Task > Modeling Standards for ISO 26262 > Check for MATLAB Function interfaces with inherited properties • By Task > Modeling Standards for EN 50128 > Check for MATLAB Function interfaces with inherited properties <p>For DO-178C/DO-331 check details, see “Check for MATLAB Function interfaces with inherited properties”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check for MATLAB Function interfaces with inherited properties”.</p>
References	<ul style="list-style-type: none"> • IEC 61508-3, Table B.9 (5) - Fully defined interface • ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation • EN 50128, Table A.1 (11) - Software Interface Specifications • DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
See Also	<ul style="list-style-type: none"> • MathWorks Automotive Advisory Board (MAAB) guideline na_0034: MATLAB Function block input/output settings

ID: Title	himl_0002: Strong data typing at MATLAB function boundaries
	<ul style="list-style-type: none"> Orion GN&C: MATLAB and Simulink Standards, jh_0063: eML block input / output settings “MATLAB Function Block Editor”
Last Changed	R2014a
Examples	<p>Recommended:</p> <p>In the “Ports and Data Manager”, specify the complexity and type of input u1 as follows:</p> <ul style="list-style-type: none"> Complexity to Off Type to uint16 <div data-bbox="353 713 1159 1020" data-label="Diagram"> </div> <p>Not Recommended:</p> <p>In the “Ports and Data Manager”, do <i>not</i> specify the complexity and type of input u1 as follows:</p> <ul style="list-style-type: none"> Complexity to Inherited Type to Inherit: Same as Simulink. <p>Note: To access the “Ports and Data Manager”, from the toolbar of the “MATLAB Function Block Editor”, select Edit Data.</p>

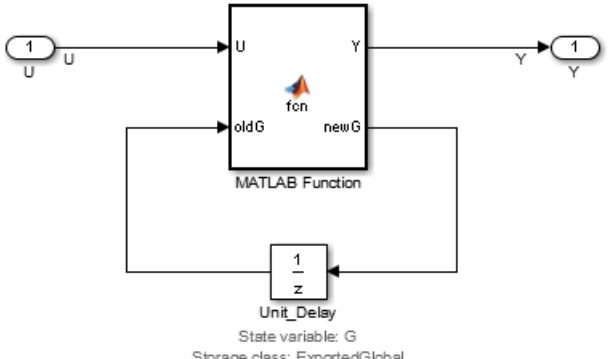
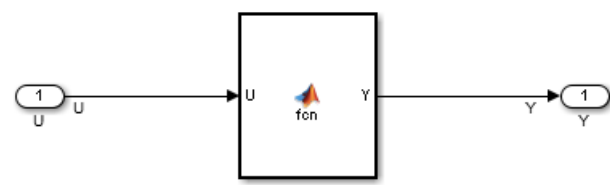
himl_0003: Limitation of MATLAB function complexity

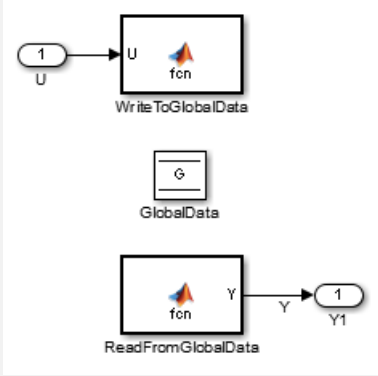
ID: Title	himl_0003: Limitation of MATLAB function complexity											
Description	<p>When using MATLAB functions, limit the size and complexity of MATLAB code. The size and complexity of MATLAB functions is characterized by:</p> <ul style="list-style-type: none"> • Lines of code • Nested function levels • Cyclomatic complexity • Density of comments (ratio of comment lines to lines of code) 											
Note	<p>Size and complexity limits can vary across projects. Typical limits might be as described in this table:</p> <table border="1" data-bbox="363 748 1326 973"> <thead> <tr> <th data-bbox="363 748 788 788">Metric</th> <th data-bbox="792 748 1326 788">Limit</th> </tr> </thead> <tbody> <tr> <td data-bbox="363 793 788 833">Lines of code</td> <td data-bbox="792 793 1326 833">60 per MATLAB function</td> </tr> <tr> <td data-bbox="363 838 788 878">Nested function levels</td> <td data-bbox="792 838 1326 878">3^{1,2}</td> </tr> <tr> <td data-bbox="363 883 788 923">Cyclomatic complexity</td> <td data-bbox="792 883 1326 923">15</td> </tr> <tr> <td data-bbox="363 928 788 968">Density of comments</td> <td data-bbox="792 928 1326 968">0.2 comment lines per line of code</td> </tr> </tbody> </table> <p>¹Pure Wrappers to external functions are not counted as separate levels.</p> <p>²Standard MATLAB library functions do not count as separate levels.</p>		Metric	Limit	Lines of code	60 per MATLAB function	Nested function levels	3 ^{1,2}	Cyclomatic complexity	15	Density of comments	0.2 comment lines per line of code
Metric	Limit											
Lines of code	60 per MATLAB function											
Nested function levels	3 ^{1,2}											
Cyclomatic complexity	15											
Density of comments	0.2 comment lines per line of code											
Rationale	<ul style="list-style-type: none"> • Readability • Comprehension • Traceability • Maintainability • Testability 											
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check for MATLAB Function metrics • By Task > Modeling Standards for IEC 61508 > Check for MATLAB Function metrics • By Task > Modeling Standards for ISO 26262 > Check for MATLAB Function metrics 											

ID: Title	himl_0003: Limitation of MATLAB function complexity
	<ul style="list-style-type: none"> • By Task > Modeling Standards for EN 50128 > Check for MATLAB Function metrics <p>For DO-178C/DO-331 check details, see “Check MATLAB Function metrics”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check MATLAB Function metrics”.</p>
References	<ul style="list-style-type: none"> • IEC 61508-3, Table B.9 (5) - Fully defined interface • ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation • EN 50128, Table A.1(11) - Software Interface Specifications • DO-331, Sections MB.6.3.1.e - High-level requirements conform to standards • DO-331, Sections MB.6.3.2.e - Low-level requirements conform to standards
See Also	<ul style="list-style-type: none"> • MathWorks Automotive Advisory Board (MAAB) guideline na_0016: Source lines of MATLAB Functions • MathWorks Automotive Advisory Board (MAAB) guideline na_0017: Number of called function levels • MathWorks Automotive Advisory Board (MAAB) guideline na_0018: Number of nested if/else and case statement • Orion GN&C: MATLAB and Simulink Standards, jh_0084: eML Comments • “MATLAB Function Block Editor”
Last Changed	R2014a

himl_0005: Usage of global variables in MATLAB functions

ID: Title	himl_0005: Usage of global variables in MATLAB functions
Description	Avoid using global variables in MATLAB functions. To access shared data, use signal lines or persistent data.
Notes	Using global data in MATLAB code requires the definition of Data Store Memory blocks or Custom Storage class objects. If the read and write access order is not specified correctly, usage of this type of storage can lead to unexpected results.
Rationale	<ul style="list-style-type: none"> • Readability • Maintainability • Deterministic Behavior
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check MATLAB code for global variables • By Task > Modeling Standards for IEC 61508 > Check MATLAB code for global variables • By Task > Modeling Standards for EN 50128 > Check MATLAB code for global variables • By Task > Modeling Standards for ISO 26262 > Check MATLAB code for global variables <p>For DO-178C/DO-331 check details, see “Check MATLAB code for global variables”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check MATLAB code for global variables”.</p>
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1(b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.3.b 'Consistency'
See Also	<ul style="list-style-type: none"> • na_0024: Global Variables • “hisl_0013: Usage of data store blocks”
Last Changed	R2014a
Examples	<ul style="list-style-type: none"> • Recommended

ID: Title	himl_0005: Usage of global variables in MATLAB functions
	<pre data-bbox="412 296 742 465"> function [Y,newG] = ... fcn(U,oldG) %#codegen Y = oldG * U; newG = oldG + 1; end </pre> <div data-bbox="412 499 1050 894" style="border: 1px solid black; padding: 10px; text-align: center;">  <p>State variable: G Storage class: ExportedGlobal</p> </div> <ul style="list-style-type: none"> <li data-bbox="378 911 620 940">• Recommended <pre data-bbox="412 968 704 1137"> function Y = fcn(U) %#codegen persistent G; if isempty(G) G = 1; end end </pre> <div data-bbox="412 1171 1050 1409" style="border: 1px solid black; padding: 10px; text-align: center;">  <p>MATLAB_Function</p> </div> <ul style="list-style-type: none"> <li data-bbox="378 1440 675 1470">• Not Recommended

ID: Title	himl_0005: Usage of global variables in MATLAB functions
	<p>Write to global data function:</p> <pre data-bbox="412 352 630 491">function fcn(U) %#codegen global G; G = U; End</pre> <p>Read from global data function:</p> <pre data-bbox="412 578 649 716">function Y = fcn %#codegen global G; Y = G; end</pre>  <p>The diagram illustrates the data flow between two MATLAB function blocks. The top block, labeled 'WriteToGlobalData', has an input port 'U' and an output port 'Y'. The bottom block, labeled 'ReadFromGlobalData', has an input port 'Y' and an output port 'Y1'. A central 'GlobalData' block, containing a variable 'G', is connected to both function blocks, representing the shared global variable.</p>

MATLAB Code

In this section...

“himl_0004: MATLAB Code Analyzer recommendations for code generation” on page 4-11

“himl_0006: MATLAB code if / elseif / else patterns” on page 4-15

“himl_0007: MATLAB code switch / case / otherwise patterns” on page 4-17

“himl_0008: MATLAB code relational operator data types” on page 4-19

“himl_0009: MATLAB code with equal / not equal relational operators” on page 4-20

“himl_0010: MATLAB code with logical operators and functions” on page 4-22

himl_0004: MATLAB Code Analyzer recommendations for code generation

ID: Title	himl_0004: MATLAB Code Analyzer recommendations for code generation	
Description	When using MATLAB code:	
	A	To activate MATLAB Code Analyzer messages for code generations, use the <code>%#codegen</code> directive in external MATLAB functions.
	B	Review the MATLAB Code Analyzer messages. Either: <ul style="list-style-type: none"> • Implement the recommendations or • Justify not following the recommendations with <code>%#ok<message-ID(S)></code> directives in the MATLAB function. Do not use <code>%#ok</code> without specific message-IDs.
Notes	The MATLAB Code Analyzer messages provide identifies potential errors, problems, and opportunities for improvement in the code.	
Rationale	A	In external MATLAB functions, the <code>%#codegen</code> directive activates MATLAB Code Analyzer messages for code generation.
	B	<ul style="list-style-type: none"> • Following MATLAB Code Analyzer recommendations helps to: <ul style="list-style-type: none"> • Generate efficient code. • Follow best code generation practices • Avoid using MATLAB features not supported for code generation.

ID: Title	himl_0004: MATLAB Code Analyzer recommendations for code generation	
		<ul style="list-style-type: none"> • Avoid code patterns which potentially influence safety. • Not following MATLAB Code Analyzer recommendations are justified with message id (e.g. %#ok<NOPRT>. <p>In the MATLAB function, using %#ok without a message id justifies the full line, potentially hiding issues.</p>
Model Advisor Checks	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check MATLAB Code Analyzer messages • By Task > Modeling Standards for IEC 61508 > Check MATLAB Code Analyzer messages • By Task > Modeling Standards for EN 50128 > Check MATLAB Code Analyzer messages • By Task > Modeling Standards for ISO 26262 > Check MATLAB Code Analyzer messages <p>For DO-178C/DO-331 check details, see “Check MATLAB Code Analyzer messages”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check MATLAB Code Analyzer messages”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.b 'Accuracy and consistency' • DO-331, Section MB.6.3.2.b 'Accuracy and consistency' 	
See Also	“Check Code for Errors and Warnings”	
Last Changed	R2014a	

ID: Title	himl_0004: MATLAB Code Analyzer recommendations for code generation
Examples	<p>Recommended</p> <ul style="list-style-type: none"> • Activate MATLAB Code Analyzer messages for code generations: <pre> %#codegen function y = function(u) y = inc_u(u)); end function yy = inc_u(uu) yy = uu + 1; end </pre> • Justify missing ; and value assigned might be unused: <pre> y = 2*u %#ok<NOPRT,NAGSU> output for debugging ... y = 3*u; </pre> • If output is not desired and assigned value is unused, remove the line <code>y = 2*u ...</code>: <pre> y = 3*u; </pre> <p>Not Recommended</p> <ul style="list-style-type: none"> • External MATLAB file used in Simulink with missing <code>%#codegen</code> directive: <pre> function y = function(u) % nested functions can't be used for code generation function yy = inc_u(uu) yy = uu + 1; end y = inc_u(u)); end </pre> • All messages in line are justified by using <code>%#ok</code> without a message ID: <pre> % missing ';' and the value might be unused y = 2*u %#ok ... y = 3*u; </pre> • No justification:

ID: Title	himl_0004: MATLAB Code Analyzer recommendations for code generation
	<pre>% missing justification for missing ';' and unnecessary '['..''] y= [2*u]</pre>

himl_0006: MATLAB code if / elseif / else patterns

ID: Title	himl_0006: MATLAB code if / elseif / else patterns
Description	For MATLAB code with <code>if / elseif / else</code> constructs, terminate the constructs with an <code>else</code> statement that includes at least a meaningful comment. A final <code>else</code> statement is not required if there is no <code>elseif</code> .
Rationale	<ul style="list-style-type: none"> • Defensive programming • Readability • Traceability
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.e 'Conformance to standards' • DO-331, Section MB.6.3.2.e 'Conformance to standards' • DO-331, Section MB.6.3.3.e 'Conformance to standards' • MISRA-C:2004, Rule 14.10 • MISRA-C:2012, Rule 15.7
See Also	<ul style="list-style-type: none"> • “hisl_0010: Usage of If blocks and If Action Subsystem blocks”
Last Changed	R2014a
Examples	<p>Recommended</p> <ul style="list-style-type: none"> • <pre>if u > 0 y = 1; end</pre> • <pre>if u > 0 y = 1; elseif u < 0 y = -1; else y = 0; end</pre> • <pre>y = 0; if u > 0</pre>

ID: Title	himl_0006: MATLAB code if / elseif / else patterns
	<pre> y = 1; elseif u < 0 y = -1; else % handled before if end</pre> <p>Not Recommended</p> <ul style="list-style-type: none">• % empty else <pre>y = 0; if u > 0 y = 1; elseif u < 0 y = -1; else end</pre>• % missing else <pre>y = 0; if u > 0 y = 1; elseif u < 0 y = -1; end</pre>

himl_0007: MATLAB code switch / case / otherwise patterns

ID: Title	himl_0007: MATLAB code switch / case / otherwise patterns
Description	<p>For MATLAB code with <code>switch</code> statements, include:</p> <ul style="list-style-type: none"> • At least two <code>case</code> statements. • An <code>otherwise</code> statement that at least includes a meaningful comment.
Note	If there is only one <code>case</code> and one <code>otherwise</code> statement, consider using an <code>if / else</code> statement.
Rationale	<ul style="list-style-type: none"> • Defensive programming • Readability • Traceability
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.e 'Conformance to standards' • DO-331, Section MB.6.3.2.e 'Conformance to standards' • DO-331, Section MB.6.3.3.e 'Conformance to standards' • MISRA-C:2004, Rule 15.3 • MISRA-C:2004, Rule 15.5 • MISRA-C:2012, Rule 16.4 • MISRA-C:2012, Rule 16.6
See Also	<ul style="list-style-type: none"> • na_0022: Recommended patterns for Switch/Case statements • "hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks"
Last Changed	R2014a
Examples	<p>Recommended</p> <ul style="list-style-type: none"> • <pre>switch u case 1 y = 3; case 3 y = 1;</pre>

ID: Title	himl_0007: MATLAB code switch / case / otherwise patterns
	<pre> otherwise y = 1; end • y = 0; switch u case 1 y = 3; case 3 y = 1; otherwise % handled before switch end </pre> <p>Not Recommended</p> <ul style="list-style-type: none"> • % no case statements <pre> switch u otherwise y = 1; end </pre> • % empty otherwise statement <pre> switch u case 1 y = 3; case 3 y = 1; otherwise end </pre> • % no otherwise statement <pre> switch u case 1 y = 3; end </pre>

himl_0008: MATLAB code relational operator data types

ID: Title	himl_0008: MATLAB code relational operator data types
Description	For MATLAB code with relational operators, use the same data type for the left and right operands.
Note	If the two operands have different data types, MATLAB will promote both operands to a common data type. This can lead to unexpected results.
Rationale	<ul style="list-style-type: none"> • Prevent implicit casts • Prevent unexpected results
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1(c) 'Enforcement of strong typing' • ISO 26262-6, Table 1(b) 'Use of language subsets' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate'
See Also	<ul style="list-style-type: none"> • “himl_0016: Usage of blocks that compute relational operators” • “himl_0017: Usage of blocks that compute relational operators (2)”
Last Changed	R2014a
Examples	<p>Recommended</p> <ul style="list-style-type: none"> • <code>myBool == true</code> • <code>myInt8 == int8(1)</code> <p>Not Recommended</p> <ul style="list-style-type: none"> • <code>myBool == 1</code> • <code>myInt8 == true</code> • <code>myInt8 == 1</code> • <code>myInt8 == int16(1)</code> • <code>myEnum1.EnumVal == int32(1)</code>

himl_0009: MATLAB code with equal / not equal relational operators

ID: Title	himl_0009: MATLAB code with equal / not equal relational operators
Description	<p>For MATLAB code with equal or not equal relational operators, avoid using the following data types:</p> <ul style="list-style-type: none"> • Single • Double • Types derived from single or double data types
Note	<p>Consider the following code fragments:</p> <pre>1 sqrt(2)^2 == 2 2 sqrt(2^2) == 2</pre> <p>Mathematically, both fragments are true. However, because of floating point rounding effects, the results are:</p> <pre>1 false 2 true</pre>
Rationale	<ul style="list-style-type: none"> • Prevent unexpected results
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • EN 50128, MB.6.3.2.g ' 'Defensive Programming' • MISRA-C:2004, Rule 13.3
See Also	<ul style="list-style-type: none"> • jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow • "himl_0016: Usage of blocks that compute relational operators"
Last Changed	R2014a
Examples	Recommended

ID: Title	himl_0009: MATLAB code with equal / not equal relational operators
	<ul style="list-style-type: none"><li data-bbox="378 296 1135 326">• <code>myDouble >= 0.99 && myDouble <= 1.01; % test range</code> <p data-bbox="372 355 635 385">Not Recommended</p> <ul style="list-style-type: none"><li data-bbox="378 413 645 472">• <code>myDouble == 1.0</code> <code>mySingle ~= 15.0</code>

himl_0010: MATLAB code with logical operators and functions

ID: Title	himl_0010: MATLAB code with logical operators and functions
Description	For logical operators and logical functions in MATLAB code, use logical data types
Notes	Logical operators: <code>&&</code> , <code> </code> , <code>~</code> Logical functions: <code>and</code> , <code>or</code> , <code>not</code> , <code>xor</code>
Rationale	<ul style="list-style-type: none"> Prevent unexpected results
References	<ul style="list-style-type: none"> IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' IEC 61508-3, Table A.3 (3) 'Language subset' ISO 26262-6, Table 1(c) 'Enforcement of strong typing' ISO 26262-6, Table 1(b) 'Use of language subsets' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.4 (11) 'Language Subset' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate'
Last Changed	R2014a
Examples	<p>Recommended</p> <ul style="list-style-type: none"> <code>~myLogical</code> <code>(myInt8 > int8(4)) && myLogical</code> <code>xor(myLogical1,myLogical2)</code> <p>Not Recommended</p> <ul style="list-style-type: none"> <code>~myInt8</code> <code>myInt8 && myDouble</code>

Configuration Parameter Considerations

- “Solver” on page 5-2
- “Diagnostics” on page 5-7
- “Optimizations” on page 5-24

Solver

In this section...
“hisl_0040: Configuration Parameters > Solver > Simulation time” on page 5-3
“hisl_0041: Configuration Parameters > Solver > Solver options” on page 5-4
“hisl_0042: Configuration Parameters > Solver > Tasking and sample time options” on page 5-5

hisl_0040: Configuration Parameters > Solver > Simulation time

ID: Title	hisl_0040: Configuration Parameters > Solver > Simulation time	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Solver pane, set parameters for simulation time as follows:	
	A	Start time to 0.0.
	B	Stop time to a positive value that is less than the value of Application lifespan (days) .
Note	<p>Simulink allows nonzero start times for simulation. However, production code generation requires a zero start time.</p> <p>By default, Application lifespan (days) is <code>inf</code>. If you do not change this setting, any positive value for Stop time is valid.</p> <p>You specify Stop time in seconds and Application lifespan (days) is in days.</p>	
Rationale	A	Generate code that is valid for production code generation.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' 	
See Also	<ul style="list-style-type: none"> • “hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)” on page 5-27 • Solver Pane section of the Simulink documentation 	
Last Changed	R2013b	

hisl_0041: Configuration Parameters > Solver > Solver options

ID: Title	hisl_0041: Configuration Parameters > Solver > Solver options	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Solver pane, set parameters for solvers as follows:	
	A	Type to Fixed-step.
	B	Solver to discrete (no continuous states).
Note	Generating code for production requires a fixed-step, discrete solver.	
Rationale	A, B	Generate code that is valid for production code generation.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' 	
See Also	"Solver Pane" in the Simulink documentation	
Last Changed	R2013b	

hisl_0042: Configuration Parameters > Solver > Tasking and sample time options

ID: Title	hisl_0042: Configuration Parameters > Solver > Tasking and sample time options	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Solver pane, set parameters for tasking and sample time as follows:	
	A	<p>Periodic sample time constraint to Specified and assign values to Sample time properties.</p> <hr/> <p>Caution If you use a referenced model as a reusable function, set Periodic sample time constraint to Ensure sample time independent.</p>
	B	Tasking mode for periodic sample times to SingleTasking or MultiTasking.
	C	Clear the parameter Automatically handle data transfers between tasks .
Notes	<p>Selecting the Automatically handle data transfers between tasks check box might result in inserting rate transition code without a corresponding model construct. This might impede establishing full traceability or showing that unintended functions are not introduced.</p> <p>You can select or clear the Higher priority value indicates higher task priority check box . Selecting this check box determines whether the priority for Sample time properties uses the lowest values as highest priority, or the highest values as highest priority.</p>	
Rationale	A, B, C	Support fully specified models and unambiguous code.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' 	
See Also	"Solver Pane" in the Simulink documentation	

ID: Title	hisl_0042: Configuration Parameters > Solver > Tasking and sample time options
Last Changed	R2013b

Diagnostics

In this section...

“hisl_0043: Configuration Parameters > Diagnostics > Solver” on page 5-8

“hisl_0044: Configuration Parameters > Diagnostics > Sample Time” on page 5-10

“hisl_0301: Configuration Parameters > Diagnostics > Compatibility” on page 5-13

“hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters” on page 5-14

“hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block” on page 5-15

“hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization” on page 5-16

“hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging” on page 5-17

“hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals” on page 5-18

“hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses” on page 5-19

“hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls” on page 5-20

“hisl_0309: Configuration Parameters > Diagnostics > Type Conversion” on page 5-21

“hisl_0310: Configuration Parameters > Diagnostics > Model Referencing” on page 5-22

“hisl_0311: Configuration Parameters > Diagnostics > Stateflow” on page 5-23

hisl_0043: Configuration Parameters > Diagnostics > Solver

ID: Title	hisl_0043: Configuration Parameters > Diagnostics > Solver													
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Solver section to:													
	Compile-Time	<ul style="list-style-type: none"> • Algebraic loop to error. • Minimize algebraic loop to error. • Unspecified inheritability of sample times to error. • Automatic solver parameter selection to error. • State name clash to warning. 												
	Run-Time	<ul style="list-style-type: none"> • Block priority violation to error if you are using block priorities. 												
Note	Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.													
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th data-bbox="379 930 851 970">If Diagnostic Parameter...</th> <th data-bbox="857 930 1337 970">Is Not Set As Indicated, Then ...</th> </tr> </thead> <tbody> <tr> <td data-bbox="379 977 851 1072">Algebraic loop</td> <td data-bbox="857 977 1337 1072">Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.</td> </tr> <tr> <td data-bbox="379 1079 851 1175">Minimize algebraic loop</td> <td data-bbox="857 1079 1337 1175">Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.</td> </tr> <tr> <td data-bbox="379 1182 851 1317">Block priority violation</td> <td data-bbox="857 1182 1337 1317">Block execution order can include undetected conflicts that might result in unpredictable block order execution.</td> </tr> <tr> <td data-bbox="379 1324 851 1459">Unspecified inheritability of sample times</td> <td data-bbox="857 1324 1337 1459">An S-function that is not explicitly set to inherit sample time can go undetected and result in unpredictable behavior.</td> </tr> <tr> <td data-bbox="379 1466 851 1524">Automatic solver parameter selection</td> <td data-bbox="857 1466 1337 1524">An automatic change to the solver, step size, or simulation stop time</td> </tr> </tbody> </table>		If Diagnostic Parameter...	Is Not Set As Indicated, Then ...	Algebraic loop	Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.	Minimize algebraic loop	Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.	Block priority violation	Block execution order can include undetected conflicts that might result in unpredictable block order execution.	Unspecified inheritability of sample times	An S-function that is not explicitly set to inherit sample time can go undetected and result in unpredictable behavior.	Automatic solver parameter selection	An automatic change to the solver, step size, or simulation stop time
If Diagnostic Parameter...	Is Not Set As Indicated, Then ...													
Algebraic loop	Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.													
Minimize algebraic loop	Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.													
Block priority violation	Block execution order can include undetected conflicts that might result in unpredictable block order execution.													
Unspecified inheritability of sample times	An S-function that is not explicitly set to inherit sample time can go undetected and result in unpredictable behavior.													
Automatic solver parameter selection	An automatic change to the solver, step size, or simulation stop time													

ID: Title	hisl_0043: Configuration Parameters > Diagnostics > Solver	
	If Diagnostic Parameter...	Is Not Set As Indicated, Then ...
		can go undetected and might the operation of generated code.
	State name clash	A name being used for more than one state might go undetected.
Rationale	Support generation of robust and unambiguous code.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for solvers For check details, see “Check safety-related diagnostic settings for solvers”.	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, MB.6.3.3.e 'Software architecture conforms to standards' 	
See Also	<ul style="list-style-type: none"> • “Diagnostics Pane: Solver” in the Simulink documentation • jc_0021: Model diagnostic settings in the Simulink documentation 	
Last Changed	R2013b	

hisl_0044: Configuration Parameters > Diagnostics > Sample Time

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time							
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Sample Time section to error :							
	Compile-Time	<ul style="list-style-type: none"> • Source block specifies -1 sample time • Discrete used as continuous • Multitask rate transition • Single task rate transition • Multitask conditionally executed subsystem • Tasks with equal priority • Enforce sample times specified by Signal Specification blocks <p>If the target system does not allow preemption between tasks that have equal priority, set Tasks with equal priority to none.</p>						
	Run-Time	Not applicable						
Note	<p>Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.</p> <table border="1" data-bbox="373 1098 1328 1512"> <thead> <tr> <th data-bbox="373 1098 851 1138">If Diagnostic Parameter...</th> <th data-bbox="857 1098 1328 1138">Is Not Set As Indicated, Then ...</th> </tr> </thead> <tbody> <tr> <td data-bbox="373 1145 851 1312">Source block specifies -1 sample time</td> <td data-bbox="857 1145 1328 1312">Use of inherited sample times for a source block, such as Sine Wave, can go undetected and result in unpredictable execution rates for source and downstream blocks.</td> </tr> <tr> <td data-bbox="373 1319 851 1512">Discrete used as continuous</td> <td data-bbox="857 1319 1328 1512">Input signals with continuous sample times for a discrete block, such as Unit Delay, can go undetected. You cannot use signals with continuous sample times for embedded real-time software applications</td> </tr> </tbody> </table>		If Diagnostic Parameter...	Is Not Set As Indicated, Then ...	Source block specifies -1 sample time	Use of inherited sample times for a source block, such as Sine Wave, can go undetected and result in unpredictable execution rates for source and downstream blocks.	Discrete used as continuous	Input signals with continuous sample times for a discrete block, such as Unit Delay, can go undetected. You cannot use signals with continuous sample times for embedded real-time software applications
If Diagnostic Parameter...	Is Not Set As Indicated, Then ...							
Source block specifies -1 sample time	Use of inherited sample times for a source block, such as Sine Wave, can go undetected and result in unpredictable execution rates for source and downstream blocks.							
Discrete used as continuous	Input signals with continuous sample times for a discrete block, such as Unit Delay, can go undetected. You cannot use signals with continuous sample times for embedded real-time software applications							

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time	
	If Diagnostic Parameter...	Is Not Set As Indicated, Then ...
	Multitask rate transition	Invalid rate transitions between two blocks operating in multitasking mode can go undetected. You cannot use invalid rate transitions for embedded real-time software applications.
	Single task rate transition	A rate transition between two blocks operating in single-tasking mode can go undetected. You cannot use single-tasking rate transitions for embedded real-time software applications.
	Multitask conditionally executed subsystems	A conditionally executed multirate subsystem, operating in multitasking mode, might go undetected and corrupt data or show unexpected behavior in a target system that allows preemption.
	Tasks with equal priority	Two asynchronous tasks with equal priority might go undetected and show unexpected behavior in target systems that allow preemption.
	Enforce sample times specified by Signal Specification blocks	Inconsistent sample times for a Signal Specification block and the connected destination block might go undetected and result in unpredictable execution rates.
Rationale	A	Support generation of robust and unambiguous code.
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for sample time For check details, see “Check safety-related diagnostic settings for sample time”.	

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time
References	<ul style="list-style-type: none">• IEC 61508-3, Table A.3 (3) 'Language subset'• ISO 26262-6, Table 1 (b) 'Use of language subsets'• EN 50128, Table A.4 (11) 'Language Subset'• DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent'• DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'• DO-331, Section MB.6.3.3.b 'Software architecture is consistent'
See Also	“Diagnostics Pane: Sample Time” in the Simulink documentation
Last Changed	R2013b

hisl_0301: Configuration Parameters > Diagnostics > Compatibility

ID: Title	hisl_0301: Configuration Parameters > Diagnostics > Compatibility	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Compatibility section to:	
	Compile-Time	S—function upgrades needed > error
	Run-Time	Not applicable
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for compatibility For check details, see “Check safety-related diagnostic settings for compatibility”.	
See Also	“Diagnostics Pane: Compatibility” in the Simulink documentation	
Last Changed	R2012b	

hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters

ID: Title	hisl_0302: Configuration Parameters > Diagnostics > Data Validity >Parameters	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Data Validity > Parameters section to:	
	Compile-Time	Detect downcast> error Detect precision loss> error
	Run-Time	Detect overflow> error Detect underflow> error
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for parameters For check details, see “Check safety-related diagnostic settings for parameters”.	
See Also	“Diagnostics Pane: Data Validity” in the Simulink documentation	
Last Changed	R2012b	

hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block

ID: Title	hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Data Validity > Merge block section to:	
	Compile-Time	Not applicable
	Run-Time	Detect multiple driving blocks executing at the same time step > error
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
See Also	“Diagnostics Pane: Data Validity” in the Simulink documentation	
Last Changed	R2011b	

hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization

ID: Title	hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Data Validity > Model Initialization section to:	
	Compile-Time	Not applicable
	Run-Time	Underspecified initialization detection > Simplified
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for model initialization For check details, see “Check safety-related diagnostic settings for model initialization”.	
See Also	“Diagnostics Pane: Data Validity” in the Simulink documentation	
Last Changed	R2012b	

hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging

ID: Title	hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Data Validity > Debugging section to:	
	Compile-Time	Model Verification block enabling > Disable All
	Run-Time	Not applicable
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
See Also	“Diagnostics Pane: Data Validity” in the Simulink documentation	
Last Changed	R2011b	

hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals

ID: Title	hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Connectivity > Signals section to:	
	Compile-Time	Not applicable
	Run-Time	Signal label mismatch> error Unconnected block input ports> error Unconnected block output ports> error Unconnected line> error
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for signal connectivity For check details, see “Check safety-related diagnostic settings for signal connectivity”.	
See Also	“Diagnostics Pane: Connectivity” in the Simulink documentation	
Last Changed	R2012b	

hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses

ID: Title	hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Connectivity > Buses section to:	
	Compile-Time	Not applicable
	Run-Time	<p>Unspecified bus object at root Output block > error</p> <p>Element name mismatch > error</p> <p>Mux blocks used to create bus signals > error</p> <p>Non-bus signals treated as bus signals > error</p> <p>Repair bus selection > Warn and repair</p>
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	<p>By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for bus connectivity</p> <p>For check details, see “Check safety-related diagnostic settings for bus connectivity”.</p>	
See Also	“Diagnostics Pane: Connectivity” in the Simulink documentation	
Last Changed	R2012b	

hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls

ID: Title	hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Connectivity > Function calls section to:	
	Compile-Time	Invalid function-call connection > error
	Run-Time	Context—dependent inputs > Enable all
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings that apply to function-call connectivity For check details, see “Check safety-related diagnostic settings that apply to function-call connectivity”.	
See Also	“Diagnostics Pane: Connectivity” in the Simulink documentation	
Last Changed	R2012b	

hisl_0309: Configuration Parameters > Diagnostics > Type Conversion

ID: Title	hisl_0309: Configuration Parameters > Diagnostics > Type Conversion	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Type Conversion section to:	
	Compile-Time	Vector / matrix block input conversion > error
	Run-Time	Not applicable
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for type conversions For check details, see “Check safety-related diagnostic settings for type conversions”.	
See Also	“Diagnostics Pane: Type Conversion” in the Simulink documentation	
Last Changed	R2012b	

hisl_0310: Configuration Parameters > Diagnostics > Model Referencing

ID: Title	hisl_0310: Configuration Parameters > Diagnostics > Model Referencing	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Model Referencing section to:	
	Compile-Time	Model block version mismatch > error Port and parameter mismatch > error Invalid root Inport / Outport block connection > error Unsupported data logging > error
	Run-Time	Not applicable
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related diagnostic settings for model referencing For check details, see “Check safety-related diagnostic settings for model referencing”.	
See Also	“Diagnostics Pane: Model Referencing” in the Simulink documentation	
Last Changed	R2012b	

hisl_0311: Configuration Parameters > Diagnostics > Stateflow

ID: Title	hisl_0311: Configuration Parameters > Diagnostics > Stateflow	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set the parameters of the Stateflow section to:	
	Compile-Time	Unexpected backtracking > error Invalid input data access in chart initialization > error No unconditional default transitions > error Transitions outside natural parent > error Transition shadowing > error
	Run-Time	Not applicable
Note	There are two categories of diagnostics — compile-time and run-time. Prior to a simulation, compile-time diagnostics run once. During a simulation, run-time diagnostics are active at every time step. Because run-time diagnostics are active during a simulation, they impact the simulation speed. For simulations outside of a verification and validation context, consider disabling run-time diagnostics.	
Rationale	Improve robustness of design.	
See Also	“Diagnostics Pane: Stateflow” in the Simulink documentation	
Last Changed	R2012b	

Optimizations

In this section...

“hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)” on page 5-25

“hisl_0046: Configuration Parameters > Optimization > Block reduction” on page 5-26

“hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)” on page 5-27

“hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold” on page 5-28

“hisl_0052: Configuration Parameters > Optimization > Data initialization” on page 5-29

“hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values” on page 5-30

“hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions” on page 5-31

“hisl_0055: Prioritization of code generation objectives for high-integrity systems” on page 5-32

hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)

ID: Title	hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)	
Description	To support unambiguous behavior when using logical operators, relational operators, and the Combinatorial Logic block,	
	A	Select Implement logic signals as Boolean data (vs. double) in the Optimization pane of the Configuration Parameters dialog box.
Notes	Selecting the Implement logic signals as Boolean data (vs. double) parameter, enables Boolean type checking, which produces an error when blocks that prefer Boolean inputs connect to double signals. This checking results in generating code that requires less memory.	
Rationale	A	Avoid ambiguous model behavior and optimize memory for generated code.
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related optimization settings For check details, see “Check safety-related optimization settings”.	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, MB.6.3.1.e 'High-level requirements conform to standards' • DO-331, MB.6.3.2.e 'Low-level requirements conform to standards' • MISRA-C:2004, Rule 12.6 	
Last Changed	R2013b	

hisl_0046: Configuration Parameters > Optimization > Block reduction

ID: Title	hisl_0046: Configuration Parameters > Optimization > Block reduction	
Description	To support unambiguous presentation of the generated code and support traceability between a model and generated code,	
	A	Clear the Block reduction parameter on the Optimization pane of the Configuration Parameters dialog box.
Notes	Selecting Block reduction might optimize blocks out of the code generated for a model. This results in requirements without associated code and violates traceability objectives.	
Rationale	A	Support unambiguous presentation of generated code.
	A	Support traceability between a model and generated code.
Model Advisor Checks	<p>By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related optimization settings</p> <p>For check details, see “Check safety-related optimization settings”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Clauses 7.4.7.2, 7.4.8.3, and 7.7.2.8 which require to demonstrate that no unintended functionality has been introduced • DO-331, Section MB.6.3.4.e ‘Source code is traceable to low-level requirements’ 	
See Also	“Block reduction” in the Simulink documentation	
Last Changed	R2012b	

hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)

ID: Title	hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)	
Description	To support the robustness of systems that run continuously, in the Configuration Parameters dialog box, on the Optimization pane:	
	A	Set Application lifespan (days) to inf .
Notes	Embedded applications might run continuously. Do not assume a limited lifespan for timers and counters. . When you set Application lifespan (days) to inf , the simulation time is less than the application lifespan.	
Rationale	A	Support robustness of systems that run continuously.
Model Advisor Checks	<p>By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related optimization settings</p> <p>For check details, see “Check safety-related optimization settings”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' 	
See Also	<ul style="list-style-type: none"> • “Application lifespan (days)” in the Simulink documentation • “hisl_0040: Configuration Parameters > Solver > Simulation time” on page 5-3 	
Last Changed	R2013b	

hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold

ID: Title	hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold	
Description	To support unambiguous code, set the minimum signal or parameter width for generating a for loop. In the Configuration Parameters dialog box, on the Optimization > Signals and Parameters pane,	
	A	Set Loop unrolling threshold to 2 or greater.
	B	If Pack Boolean data into bitfields is selected, set Bitfield declarator type specifier to <code>uint_T</code> .
Notes	The Loop unrolling threshold parameter specifies the array size at which the code generator begins to use a for loop, instead of separate assignment statements, to assign values to the elements of a signal or parameter array. The default value is 5.	
Rationale	A	Support unambiguous generated code.
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language Subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • MISRA-C:2004, Rule 6.4 	
See Also	“Loop unrolling threshold ” in the Simulink documentation	
Last Changed	R2013b	

hisl_0052: Configuration Parameters > Optimization > Data initialization

ID: Title	hisl_0052: Configuration Parameters > Optimization > Data initialization	
Description	To support complete definition of data and initialize internal and external data to zero, in the Configuration Parameters dialog box, on the Optimization pane,	
	A	Clear Remove root level I/O zero initialization .
	B	Clear Remove internal data zero initialization .
Note	Explicitly initialize all variables. If the run-time environment of the target system provides mechanisms to initialize all I/O and state variables, consider using the initialization of the target as an alternative to the suggested settings.	
Rationale	A, B	Support fully defined data in generated code.
Model Advisor Checks	<p>By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related optimization settings</p> <p>For check details, see “Check safety-related optimization settings”.</p>	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA-C:2004, Rule 9.1 • DO-331, Section MB.6.3.3.b 'Software architecture is consistent' 	
See Also	<p>Information about the following parameters in the Simulink documentation:</p> <ul style="list-style-type: none"> • “Remove root level I/O zero initialization” • “Remove internal data zero initialization” 	
Last Changed	R2013b	

hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values

ID: Title	hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values	
Description	To support verifiable code, In the Configuration Parameters dialog box, on the Optimization pane,	
	A	Consider selecting Remove code from floating-point to integer conversions that wraps out-of-range values .
Notes	Avoid overflows as opposed to handling them with wrapper code. For blocks that have the parameter Saturate on overflow cleared, clearing Remove code from floating-point to integer conversions that wraps out-of-range values might add code that wraps out of range values, resulting in unreachable code that cannot be tested.	
Rationale	A	Support generation of code that can be verified.
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related optimization settings For check details, see “Check safety-related optimization settings”.	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA-C:2004, Rule 14.1 • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' 	
See Also	“Remove code from floating-point to integer conversions that wraps out-of-range values” in the Simulink documentation	
Last Changed	R2013b	

hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions

ID: Title	hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions	
Description	To support the robustness of the operations, in the Configuration Parameters dialog box, on the Optimization pane,	
	A	Clear Remove code that protects against division arithmetic exceptions .
Note	Avoid division-by-zero exceptions. If you clear Remove code that protects against division arithmetic exceptions , the code generator produces code that guards against division by zero for fixed-point data.	
Rationale	A	Protect against divide-by-zero exceptions for fixed-point code.
Model Advisor Checks	By Task > Modeling Standards for DO-178C/DO-331 > Check safety-related optimization settings For check details, see “Check safety-related optimization settings”.	
References	<ul style="list-style-type: none"> • IEC 61508-3, Table A.3 (3) 'Language Subset' • IEC 61508-3 Table A.4 (3) 'Defensive Programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' • ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA-C:2004, Rule 21.1 • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' 	
See Also	“Remove code that protects against division arithmetic exceptions” in the Simulink documentation	
Last Changed	R2013b	

hisl_0055: Prioritization of code generation objectives for high-integrity systems

ID: Title	hisl_0055: Prioritized configuration objectives for high-integrity systems	
Description	Prioritize objectives for high-integrity systems using the Code Generation Advisor by:	
	A	Assigning the highest priority to the high-integrity and traceability objectives (Safety precaution and Traceability)
	B	Configuring the Code Generation Advisor to run before generating code by setting Check model before generating code to On (proceed with warnings) or On (stop for warnings) .
Notes	<p>Model configuration parameters provide control over many aspects of generated code. The prioritization of objectives specifies how configuration parameters are set when conflicts between objectives occur.</p> <p>Including the ROM, RAM, and Execution efficiency objectives with a lower priority in the list enables efficiency optimizations that do not conflict with Safety precaution and Traceability in the active configuration.</p> <p>Review the resulting parameter configurations to verify that safety requirements are met.</p>	
Rationale	A, B	When you use the Code Generation Advisor, configuration parameters conform to the objectives that you want and they are consistently enforced.
References	<ul style="list-style-type: none"> • DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' • IEC61508–3, Table A.3 (3) 'Language Subset' • IEC 61508–3, Table A.4 (3) 'Defensive Programing' • ISO 26262–6, Table 1(b) 'Use of language subsets' • ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.3 (1) 'Defensive Programming' 	
See also	<ul style="list-style-type: none"> • “Set Objectives — Code Generation Advisor Dialog Box” • “Manage a Configuration Set” • “cgsl_0301: Prioritization of code generation objectives for code efficiency” 	

ID: Title	hisl_0055: Prioritized configuration objectives for high-integrity systems
Last Changed	R2014a

MISRA-C:2004 Compliance Considerations

- “Modeling Style” on page 6-2
- “Block Usage” on page 6-17
- “Configuration Settings” on page 6-22
- “Stateflow Chart Considerations” on page 6-26
- “System Level” on page 6-36

Modeling Style

In this section...

“hisl_0061: Unique identifiers for clarity” on page 6-3

“hisl_0062: Global variables in graphical functions” on page 6-9

“hisl_0063: Length of user-defined function names to improve MISRA-C:2004 compliance” on page 6-11

“hisl_0064: Length of user-defined type object names to improve MISRA-C:2004 compliance” on page 6-12

“hisl_0065: Length of signal and parameter names to improve MISRA-C:2004 compliance” on page 6-13

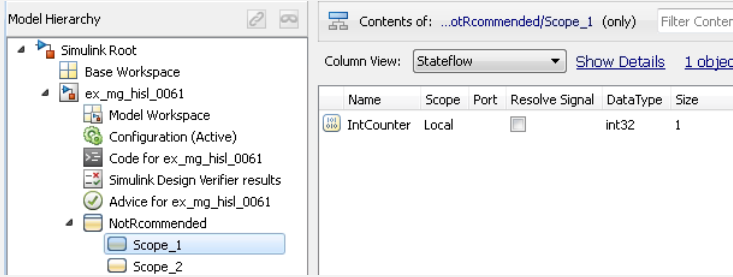
“hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance” on page 6-14

“hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance” on page 6-15

hisl_0061: Unique identifiers for clarity

ID: Title	hisl_0061: Unique identifiers for clarity	
Description	When developing a model:	
	A	Use unique identifiers for Simulink signals.
	B	Define unique identifiers across multiple scopes within a chart.
Notes	The code generator resolves conflicts between identifiers so that symbols in the generated code are unique. The process is called name mangling.	
Rationale	A, B	Improve readability of a graphical model and mapping between identifiers in the model and generated code.
References	<ul style="list-style-type: none"> • MISRA-C: 2004 5.6 • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508–3, Table A.3 (3) 'Language subset' • IEC 61508–3, Table A.4 (5) 'Design and coding standards' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (e) 'Use of established design principles' • ISO 26262-6, Table 1 (h) 'Use of naming conventions' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.12 (1) 'Coding Standard' 	
Model Advisor Check	<ul style="list-style-type: none"> • By Task > Modeling Standards for DO-178C/DO-331 > Check Stateflow charts for uniquely defined data objects • By Task > Modeling Standards for IEC 61508 > Check usage of Stateflow constructs • By Task > Modeling Standards for ISO 26262 > Check usage of Stateflow constructs • By Task > Modeling Standards for EN 50128 > Check usage of Stateflow constructs <p>For DO-178C/DO-331 check details, see “Check Stateflow charts for uniquely defined data objects”.</p> <p>For IEC 61508, EN 50128 and ISO 26262 check details, see “Check usage of Stateflow constructs”.</p>	
See Also	“Code Appearance” in the Simulink Coder™ documentation	

ID: Title	hisl_0061: Unique identifiers for clarity
Last Changed	R2015a

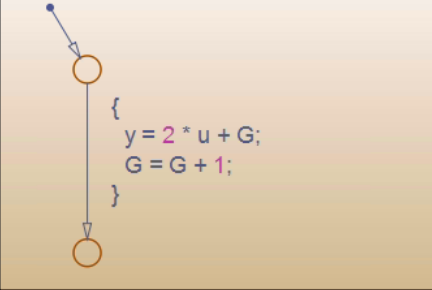
ID: Title	hisl_0061: Unique identifiers for clarity												
Examples	<p data-bbox="372 302 635 331">Not Recommended</p> <p data-bbox="372 361 1225 421">In the following example, two states <code>Scope_1</code> and <code>Scope_2</code> use local identifier <code>IntCounter</code>.</p> <div data-bbox="372 453 1262 996" style="border: 1px dashed gray; padding: 10px; margin: 10px 0;"> <pre data-bbox="406 508 970 690"> Scope_1 % IntCounter is defined at this scope entry: IntCounter = int32(0); during: Chart_Level_Output_S1 = Chart_Level_Input + IntCounter; IntCounter = IntCounter + int32(1); </pre> <pre data-bbox="406 751 970 933"> Scope_2 % IntCounter is defined at this scope entry: IntCounter = int32(0); during: Chart_Level_Output_S2 = Chart_Level_Input + IntCounter; IntCounter = IntCounter + int32(1); </pre> </div> <p data-bbox="372 1032 1319 1062">The identifier <code>IntCounter</code> is defined for two states, <code>Scope_1</code> and <code>Scope_2</code>.</p> <div data-bbox="372 1095 1104 1373">  <table border="1" data-bbox="688 1182 1104 1373"> <thead> <tr> <th>Name</th> <th>Scope</th> <th>Port</th> <th>Resolve Signal</th> <th>Data Type</th> <th>Size</th> </tr> </thead> <tbody> <tr> <td>IntCounter</td> <td>Local</td> <td></td> <td><input type="checkbox"/></td> <td>int32</td> <td>1</td> </tr> </tbody> </table> </div>	Name	Scope	Port	Resolve Signal	Data Type	Size	IntCounter	Local		<input type="checkbox"/>	int32	1
Name	Scope	Port	Resolve Signal	Data Type	Size								
IntCounter	Local		<input type="checkbox"/>	int32	1								

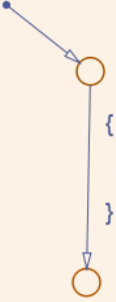
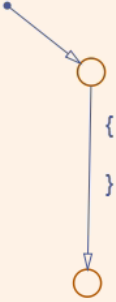
ID: Title	hisl_0061: Unique identifiers for clarity													
	<p>Model Hierarchy</p> <ul style="list-style-type: none">Simulink Root<ul style="list-style-type: none">Base Workspace<ul style="list-style-type: none">ex_mg_hisl_0061<ul style="list-style-type: none">Model Workspace<ul style="list-style-type: none">Configuration (Active)Code for ex_mg_hisl_0061Simulink Design Verifier resultsAdvice for ex_mg_hisl_0061NotRecommended<ul style="list-style-type: none">Scope_1Scope_2	<p>Contents of: ...otRecommended/Scope_2 (only) Filter Contents</p> <p>Column View: Stateflow Show Details 1 object(s)</p> <table border="1"><thead><tr><th>Name</th><th>Scope</th><th>Port</th><th>Resolve Signal</th><th>DataType</th><th>Size</th></tr></thead><tbody><tr><td>IntCounter</td><td>Local</td><td></td><td><input type="checkbox"/></td><td>int32</td><td>1</td></tr></tbody></table>	Name	Scope	Port	Resolve Signal	DataType	Size	IntCounter	Local		<input type="checkbox"/>	int32	1
Name	Scope	Port	Resolve Signal	DataType	Size									
IntCounter	Local		<input type="checkbox"/>	int32	1									

ID: Title	hisl_0061: Unique identifiers for clarity
	<p data-bbox="372 302 580 331">Recommended</p> <p data-bbox="372 361 1332 456">To clarify the model, create unique identifiers. In the following example, state <code>Scope_1</code> uses local identifier <code>IntCounter_Scope_1</code>. State <code>Scope_2</code> uses local identifier <code>IntCounter_Scope_2</code>.</p> <div data-bbox="422 539 1262 760" style="border: 1px dashed black; border-radius: 10px; padding: 10px;"><pre data-bbox="422 539 1262 760">Scope_1 % IntCounter_Scope_1 is defined at this scope entry: IntCounter_Scope_1 = int32(0); during: Chart_Level_Output_S1 = Chart_Level_Input + IntCounter_Scope_1; IntCounter_Scope_1 = IntCounter_Scope_1 + int32(1);</pre></div> <div data-bbox="422 788 1262 1008" style="border: 1px dashed black; border-radius: 10px; padding: 10px;"><pre data-bbox="422 788 1262 1008">Scope_2 % IntCounter_Scope_2 is defined at this scope entry: IntCounter_Scope_2 = int32(0); during: Chart_Level_Output_S2 = Chart_Level_Input + IntCounter_Scope_2; IntCounter_Scope_2 = IntCounter_Scope_2 + int32(1);</pre></div> <p data-bbox="372 1098 1332 1159">The identifier <code>IntCounter_Scope_1</code> is defined for state <code>Scope_1</code>. Identifier <code>IntCounter_Scope_2</code> is defined for <code>Scope_2</code>.</p>

ID: Title	hisl_0061: Unique identifiers for clarity													
	<p>Model Hierarchy</p> <ul style="list-style-type: none"> Simulink Root <ul style="list-style-type: none"> Base Workspace ex_mg_hisl_0061 <ul style="list-style-type: none"> Model Workspace Configuration (Active) Code for ex_mg_hisl_0061 Simulink Design Verifier results Advice for ex_mg_hisl_0061 NotRecommended <ul style="list-style-type: none"> Scope_1 Scope_2 Recommended <ul style="list-style-type: none"> Scope_1 Scope_2 	<p>Contents of: .../Recommended/Scope_1 (only) Filter Contents</p> <p>Column View: Stateflow Show Details 1 object(s)</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Scope</th> <th>Port</th> <th>Resolve Signal</th> <th>Data Type</th> <th>Size</th> </tr> </thead> <tbody> <tr> <td>IntCounter_Scope_1</td> <td>Local</td> <td></td> <td><input type="checkbox"/></td> <td>int32</td> <td>1</td> </tr> </tbody> </table>	Name	Scope	Port	Resolve Signal	Data Type	Size	IntCounter_Scope_1	Local		<input type="checkbox"/>	int32	1
Name	Scope	Port	Resolve Signal	Data Type	Size									
IntCounter_Scope_1	Local		<input type="checkbox"/>	int32	1									
	<p>Model Hierarchy</p> <ul style="list-style-type: none"> Simulink Root <ul style="list-style-type: none"> Base Workspace ex_mg_hisl_0061 <ul style="list-style-type: none"> Model Workspace Configuration (Active) Code for ex_mg_hisl_0061 Simulink Design Verifier results Advice for ex_mg_hisl_0061 NotRecommended <ul style="list-style-type: none"> Scope_1 Scope_2 Recommended <ul style="list-style-type: none"> Scope_1 Scope_2 	<p>Contents of: .../Recommended/Scope_2 (only) Filter Contents</p> <p>Column View: Stateflow Show Details 1 object(s)</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Scope</th> <th>Port</th> <th>Resolve Signal</th> <th>Data Type</th> <th>Size</th> </tr> </thead> <tbody> <tr> <td>IntCounter_Scope_2</td> <td>Local</td> <td></td> <td><input type="checkbox"/></td> <td>int32</td> <td>1</td> </tr> </tbody> </table>	Name	Scope	Port	Resolve Signal	Data Type	Size	IntCounter_Scope_2	Local		<input type="checkbox"/>	int32	1
Name	Scope	Port	Resolve Signal	Data Type	Size									
IntCounter_Scope_2	Local		<input type="checkbox"/>	int32	1									

hisl_0062: Global variables in graphical functions

ID: Title	hisl_0062: Global variables in graphical functions
Description	For data with a global scope used in a function, do not use the data in the calling expression if a value is assigned to the data in that function.
Rationale	Enhance readability of a model by removing ambiguity in the values of global variables.
References	<ul style="list-style-type: none"> • IEC 61508–3, Table A.3 (3) 'Language subset' • IEC 61508–3, Table A.4 (4) 'Modular approach' • IEC 61508–3, A.4 (5) 'Design and coding standards' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • ISO 26262-6, Table 1 (f) 'Use of unambiguous graphical representation' • ISO 26262-6, Table 1 (h) 'Use of naming conventions' • EN 50128, Table A.4 (11) 'Language Subset' • EN 50128, Table A.12 (1) 'Coding Standard' • EN 50128, Table A.12 (2) 'Coding Style Guide' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004 12.2 • MISRA-C:2004 12.4
Last Changed	R2015a
Examples	<p data-bbox="452 1055 1332 1117">Consider a graphical function <code>graphicalFunction</code> that modifies the global data <code>G</code>.</p> <div data-bbox="452 1152 931 1529" style="border: 1px solid black; padding: 10px; background-color: #f9f9f9;"> <pre data-bbox="482 1177 838 1204">function y = graphicalFunction(u)</pre>  </div>

ID: Title	hisl_0062: Global variables in graphical functions
	<p data-bbox="451 302 657 328">Recommended</p> <div data-bbox="451 361 1083 715"><p data-bbox="673 395 1061 482">function y = graphicalFunction(u)</p><pre data-bbox="575 510 853 621">{ Y = graphicalFunction(U); Y = Y + G; }</pre><p>The diagram shows a control flow graph with two nodes. The top node is a circle with an arrow pointing to it from the top-left. The bottom node is a circle with an arrow pointing to it from the top node. A box to the right contains the function signature 'function y = graphicalFunction(u)'. A code block between the nodes shows a function body with two lines: 'Y = graphicalFunction(U);' and 'Y = Y + G;' enclosed in curly braces.</p></div> <p data-bbox="451 753 713 779">Not Recommended</p> <div data-bbox="451 812 1090 1183"><p data-bbox="658 864 1046 951">function y = graphicalFunction(u)</p><pre data-bbox="575 979 897 1065">{ Y = graphicalFunction(U) + G; }</pre><p>The diagram shows a control flow graph with two nodes. The top node is a circle with an arrow pointing to it from the top-left. The bottom node is a circle with an arrow pointing to it from the top node. A box to the right contains the function signature 'function y = graphicalFunction(u)'. A code block between the nodes shows a function body with one line: 'Y = graphicalFunction(U) + G;' enclosed in curly braces.</p></div>

hisl_0063: Length of user-defined function names to improve MISRA-C:2004 compliance

ID: Title	hisl_0063: Length of user-defined function names to improve MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code when working with Subsystem blocks with the block parameter Function name options set to <code>User specified</code> :	
	A	Limit the length of data object names to 31 characters or fewer.
	For this rule, Subsystem blocks include standard Simulink Subsystems, MATLAB Function blocks, and Stateflow blocks.	
Rationale	A	Function names longer than 31 characters might result in a MISRA-C:2004 violation.
References	<ul style="list-style-type: none"> MISRA-C:2004 Rule 5.1 	
Prerequisites	"hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance"	
Last Changed	R2011a	

hisl_0064: Length of user-defined type object names to improve MISRA-C:2004 compliance

ID: Title	hisl_0064: Length of user-defined type object names to improve MISRA-C:2004 compliance
Description	<p>To improve MISRA-C:2004 compliance of the generated code, limit the length of data object names to 31 characters or fewer for:</p> <ul style="list-style-type: none"> • Simulink.AliasType • Simulink.NumericType • Simulink.Variant • Simulink.Bus • Simulink.BusElement • Simulink.IntEnumType
Rationale	The length of the type definitions in the generated code name might result in a MISRA-C:2004 violation.
References	<ul style="list-style-type: none"> • MISRA-C:2004 Rule 5.1
Prerequisites	“hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance”
Last Changed	R2011a

hisl_0065: Length of signal and parameter names to improve MISRA-C:2004 compliance

ID: Title	hisl_0065: Length of signal and parameter names to improve MISRA-C:2004 compliance
Description	<p>To improve MISRA-C:2004 compliance of the generated code, limit the length of signal and parameter names to 31 characters or fewer when using any of the following storage classes:</p> <ul style="list-style-type: none"> • Exported global • Imported Extern • Imported Extern Pointer • Custom storage class
Rationale	The length of the signal and parameter name might result in a MISRA-C:2004 violation.
References	<ul style="list-style-type: none"> • MISRA-C:2004 Rule 5.1
Prerequisites	“hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance”
Last Changed	R2011a

hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance

ID: Title	hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance	
Description	To improve MISRA-C: 2004 compliance of the generated code, define reserved keywords to prevent identifier clashes within the project namespace.	
	A	In the Configuration Parameters dialog box, on the Simulation Target > Symbols > Reserved names pane, define reserved identifiers.
	B	Use a consistent set of reserved identifiers for all models.
Notes	Simulink Coder checks models for standard C language key words. Expand the list of reserved identifiers to include project specific identifiers. Examples include target-specific clashes, standard and custom library clashes, and other identified clashes.	
Rationale	Improve MISRA-C:2004 compliance of the generated code.	
See Also	<ul style="list-style-type: none"> • “Simulation Target Pane: Symbols” in the Simulink documentation • “Reserved Keywords” in the Simulink Coder documentation • “Reserved names” in the Simulink Coder documentation 	
References	MISRA-C:2004, Rule 20.2	
Last Changed	R2011b	

hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance

ID: Title	hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance
Description	<p>To improve MISRA-C:2004 compliance of generated code, insert a data type conversion block when using signals of type single (<code>real32_T</code>) as inputs to the following blocks:</p> <ul style="list-style-type: none"> • Math • Trigonometry • Sqrt <p>The data type conversion block to changes the data type to double (<code>real_T</code>)</p>
Rationale	Improve MISRA-C:2004 compliance of the generated code.
Notes	The function prototypes for many math functions require an input of type double. To accommodate the function prototype, you can add a data type conversion block. As an alternative to the data type conversion block, you could define a new function interface using the Target Function Library (TFL).
References	<ul style="list-style-type: none"> • MISRA-C: 2004 Rule 10.2
Last Changed	R2012a

ID: Title	hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance
Example	<p>Recommended</p> <p>Add a data type conversion block to the input signal of the block. Convert the output signal back to single.</p>

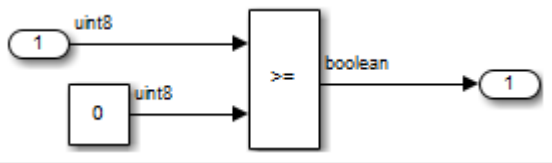
Block Usage

In this section...
“hisl_0020: Blocks not recommended for MISRA-C:2004 compliance” on page 6-17
“hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance” on page 6-18
“hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance” on page 6-21

hisl_0020: Blocks not recommended for MISRA-C:2004 compliance

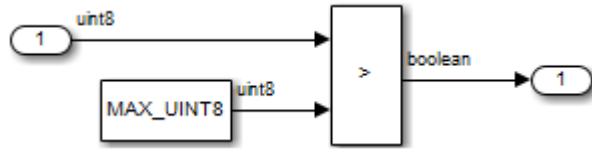
ID: Title	hisl_0020: Blocks not recommended for MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code,	
	A	Use only blocks that support code generation, as documented in the Simulink Block Support Table
	B	Do not use blocks that are listed as “Not recommended for production code” in the Simulink Block Support Table
Notes	<p>If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard.</p> <p>Choose Simulink Help > Block Support Table > Simulink to view the block support table.</p> <p>Blocks with the footnote (4) in the Block Support Table are classified as “Not Recommended for production code.”</p>	
Rationale	A,B	Improve MISRA-C:2004 compliance of the generated code.
Model Advisor Checks	<p>By Product > Embedded Coder > Check for blocks not recommended for MISRA-C:2004 compliance</p> <p>For check details, see “Check for blocks not recommended for MISRA-C:2004 compliance”.</p>	
References	MISRA-C:2004	
Last Changed	R2011a	

hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance

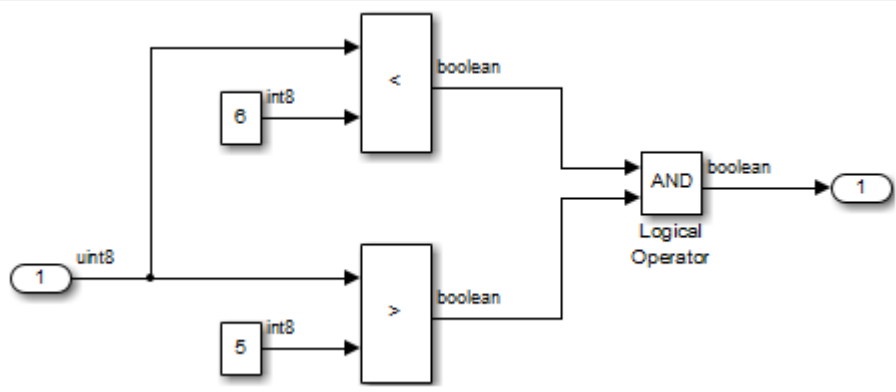
ID: Title	hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance
Description	<p>To improve MISRA-C:2004 compliance of generated code, avoid comparison operations with invariant results. Comparison operations are performed by the following blocks:</p> <ul style="list-style-type: none"> • If • Logic • Relational Operator • Switch • Switch Case • Compare to Constant
Rationale	Improve MISRA-C:2004 compliance of the generated code.
References	<ul style="list-style-type: none"> • MISRA-C: 2004 Rule 13.7 • MISRA-C: 2004 Rule 14.1
Last Changed	R2012a
Example	<p>Invariant comparisons can occur in simple or compound comparison operations. In compound comparison operations, the individual components can be variable when the full calculation is invariant.</p> <p>Simple: A uint8 is always greater than or equal to 0.</p>  <pre> graph LR A([1]) -- uint8 --> B[>=] C[0] -- uint8 --> B B -- boolean --> D([1]) </pre> <p>Simple: A uint8 cannot have a value greater than 256</p>

ID: Title

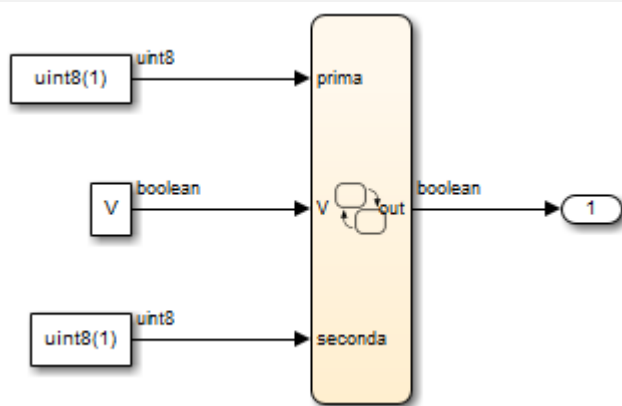
hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance

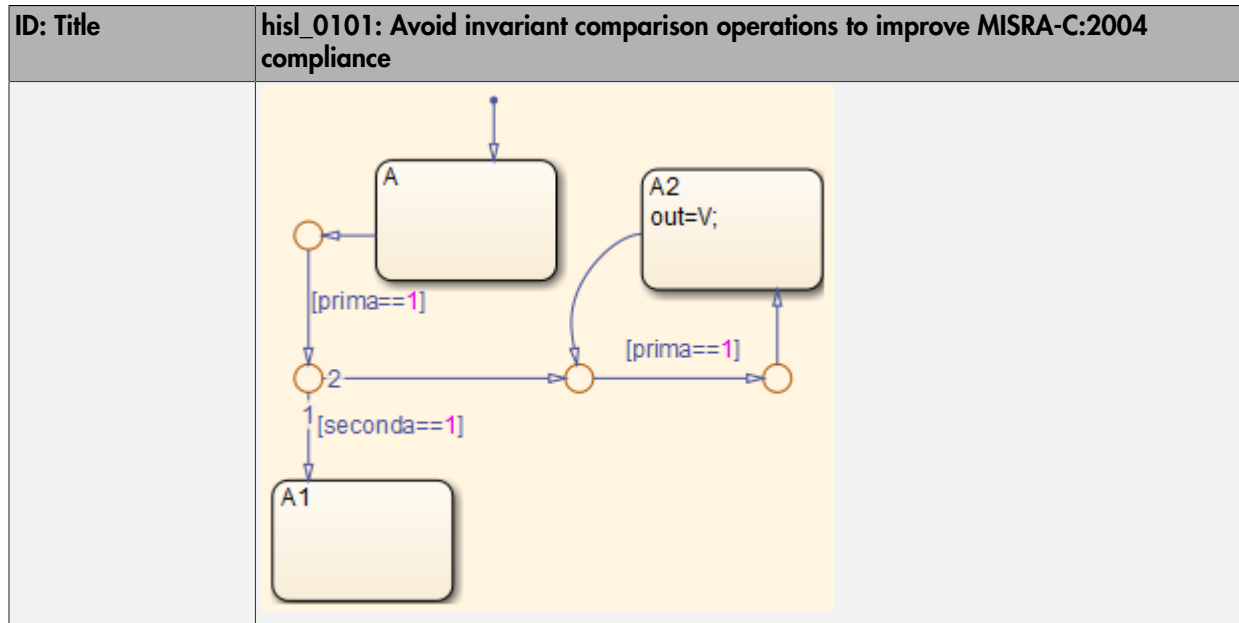


Compound: The comparison operations are mutually exclusive



Stateflow:





hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance

ID: Title	hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance
Description	To improve MISRA-C:2004 compliance of generated code, use integer data type for variables that are used as loop control counter variables in: <ul style="list-style-type: none">• For and while loops constructed in Stateflow and MATLAB.• While Iterator and For Iterator blocks.
Rationale	Improve MISRA-C:2004 compliance of the generated code.
References	<ul style="list-style-type: none">• MISRA-C: 2004 Rule 13.7
Last Changed	R2012a

Configuration Settings

In this section...

“hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance” on page 6-22

“hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance” on page 6-24

“hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance” on page 6-25

hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance

ID: Title	hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code,	
	A	Set the following model configuration parameters as specified:
	Pane / Configuration Parameter	Value
	Diagnostics > Data Validity	
	Model Verification block enabling	Disable All
	Code Generation pane	
	System target file	ERT-based target
	Code Generation > Interface pane	
	Support: non-finite numbers	Cleared (off)
	Support: continuous time	Cleared (off)
	Support: non-inlined S-functions	Cleared (off)
	MAT-file logging	Cleared (off)
	Standard math library	C89/C90 (ANSI)
	Code replacement library	None

ID: Title	hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance											
		<table border="1"> <thead> <tr> <th>Pane / Configuration Parameter</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Code Generation > Code Style pane</td> <td></td> </tr> <tr> <td> Parenthesis level</td> <td>Maximum (Specify precedence with parentheses)</td> </tr> <tr> <td>Code Generation > Symbols pane</td> <td></td> </tr> <tr> <td> Maximum identifier length</td> <td>31</td> </tr> </tbody> </table>	Pane / Configuration Parameter	Value	Code Generation > Code Style pane		Parenthesis level	Maximum (Specify precedence with parentheses)	Code Generation > Symbols pane		Maximum identifier length	31
Pane / Configuration Parameter	Value											
Code Generation > Code Style pane												
Parenthesis level	Maximum (Specify precedence with parentheses)											
Code Generation > Symbols pane												
Maximum identifier length	31											
Note	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard.											
Rationale	A	Improve MISRA-C:2004 compliance of the generated code.										
Model Advisor Checks	<p>By Product > Embedded Coder > Check configuration parameters for MISRA-C:2004 compliance</p> <p>For check details, see “Check configuration parameters for MISRA-C:2004 compliance”.</p>											
References	<ul style="list-style-type: none"> MISRA-C:2004 											
Last Changed	R2011a											

hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance

ID: Title	hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of generated code, use a consistent set of model parameters. The parameters include, but are not limited to:	
	A	Explicitly setting model character encoding using the <code>slCharacterEncoding(encoding)</code> function.
	B	In the Configuration Parameters dialog box, explicitly selecting a Hardware Implementation > Production hardware > Signed integer division rounds to: parameter.
	C	If complex numbers are not required, deselecting the Code Generation > Interface > Software Environment > complex numbers parameter.
Notes	<p>Base the selection of the integer division method on the target hardware and compiler. When available, in the Configuration Parameters dialog box, specify both of these parameters:</p> <ul style="list-style-type: none"> • Hardware Implementation > Production hardware > Device vendor • Hardware Implementation > Production hardware > Device type 	
Rationale	Improve MISRA-C:2004 compliance of the generated code.	
See Also	<ul style="list-style-type: none"> • “Configure Test and Production Target Hardware” in the Simulink Coder documentation. • <code>slCharacterEncoding</code> in the Simulink documentation. • “hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance” 	
References	<ul style="list-style-type: none"> • MISRA-C: 2004 Rule 3.2 • MISRA-C: 2004 Rule 3.3 • MISRA-C: 2004 Rule 5.7 	
Last Changed	R2012a	

hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance

ID: Title	hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance
Description	To improve MISRA-C:2004 compliance of generated code when bitfields are used, in the Configuration Parameters dialog box, set Optimization > Signals and Parameters > Code generation > Bitfield declarator type specifier to <code>uint_T</code> .
Rationale	Improve MISRA-C:2004 compliance of the generated code.
Notes	Set Bitfield declarator type specifier to <code>uint_T</code> if any of the following Optimization parameters are enabled: <ul style="list-style-type: none"> • Optimization > Signals and Parameters > Code generation > Pack Boolean data into bitfields • Optimization > Stateflow > Code generation > Use bitsets for storing state configuration • Optimization > Stateflow > Code generation > Use bitsets for storing Boolean data
See Also	<ul style="list-style-type: none"> • “Optimization Pane: Signals and Parameters” in the Simulink documentation.
References	<ul style="list-style-type: none"> • MISRA-C: 2004 Rule 6.4
Last Changed	R2012a

Stateflow Chart Considerations

In this section...

“hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance” on page 6-27

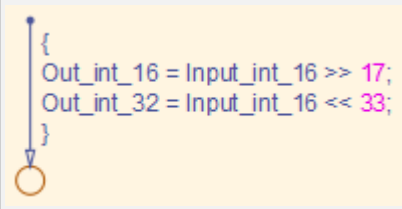
“hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance” on page 6-29

“hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance” on page 6-31

“hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance” on page 6-32

“hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance” on page 6-33

hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance

ID: Title	hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code with Stateflow bit-shifting operations, do not perform:	
	A	Right-shift operations greater than the bit-width of the input type, or by a negative value.
	B	Left-shift operations greater than the bit-width of the output type, or by a negative value.
Note	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard.	
Rationale	A,B	To avoid shift operations in the generated code that might be a MISRA-C:2004 violation.
References	<ul style="list-style-type: none"> MISRA-C:2004 Rule 12.8 	
Prerequisites	"hisf_0060: Configuration parameters that improve MISRA-C:2004 compliance"	
Last Changed	R2011a	
Example	<p>In the first equation, shifting 17 bits to the right pushes data stored in a 16-bit word out of range. The resulting output is zero. In the second equation, shifting the data 33 bits pushes data beyond the range of storage for a 32-bit word. Again, the resulting output is zero.</p>  <pre> { Out_int_16 = Input_int_16 >> 17; Out_int_32 = Input_int_16 << 33; } </pre>	

ID: Title	hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance
	<pre>void stateflow_shift_passed_step(void) { <u>Out_int_16</u> = (<u>int16_T</u>) (<u>Input_int_16</u> >> 17); <u>Out_int_32</u> = <u>Input_int_16</u> << 33; }</pre>

hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance

ID: Title	hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code, protect against Stateflow casting integer and fixed-point calculations to wider data types than the input data types by:	
	A	Explicitly type casting the calculations
	B	Using the := notation in Stateflow
Note	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard.	
Rationale	A,B	To avoid shift operations in the generated code that might be a MISRA-C:2004 violation.
References	<ul style="list-style-type: none"> • MISRA-C:2004 Rule 10.1 • MISRA-C:2004 Rule 10.4 	
Prerequisites	"hisf_0060: Configuration parameters that improve MISRA-C:2004 compliance"	
Last Changed	R2011a	
Example	<p>The example shows the default behavior and both methods of controlling the casting (explicitly type casting and using the colon operator).</p> <pre> { Out_Default = First_16 - Second_16;... Out_Colon := First_16 - Second_16;... Out_Explicate = int32(First_16) - int32(Second_16); } </pre> <pre> void stateflow_wide_shift_step(void) { Out_Default = First_16 - Second_16; Out_Colon = (int32_T)First_16 - (int32_T)Second_16; Out_Explicate = (int32_T)First_16 - (int32_T)Second_16; } </pre>	

hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance

ID: Title	hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance
Description	To improve MISRA-C:2004 compliance of the generated code:
	A Do not use unary minus operators on unsigned data types
Note	The MATLAB and C action languages do not restrict the use of unary minus operators on unsigned expressions.
Rationale	A Improve MISRA-C:2004 compliance of the generated code.
References	• MISRA-C:2004 Rule 12.9
Last Changed	R2014b
Example	<p>Not Recommended:</p> <pre> /* Gateway: Chart */ /* During: Chart */ /* Transition: '<S1>:1' */ varOut_SF_uint8 = (uint8_I) (-varIn_SF_uint8 * 3); </pre> <p>Applying the unary minus operator to the unsigned integer results in a MISRA-C:2004 Rule 12.9 violation. The resulting output wraps around the maximum value of 256 (uint8). In this example, if the input variable In_SF_uint8 equals 7, then the output variable varOut_uint8 equals 256 - (7 * 3), or 235. The simulation and code generation values are in agreement.</p>

hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance

ID: Title	hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code:	
	A	Explicitly select an integer data type as the control variable in a Stateflow for loop
Note	The default data type in Simulink and Stateflow is double. Explicitly select an integer data type.	
Rationale	A	Improve MISRA-C:2004 compliance of the generated code
References	<ul style="list-style-type: none"> • MISRA-C:2004 Rule 13.4 	
Last Changed	R2011b	

hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance

ID: Title	hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code for floating point and integer-based operations, do one of the following:	
	A	Perform static analysis of the model to prove that division by zero is not possible
	B	Provide run-time error checking in the generated C code by explicitly modeling the error checking in Stateflow
	C	Modify the code generation process using Code Replacement Libraries (CRLs) to protect against division by zero
	D	For integer-based operations, in the Configuration Parameters dialog box, on the Optimization pane, clear Remove code that protects against division arithmetic exceptions
Note	<p>Using run-time error checking introduces additional computational and memory overhead in the generated code. It is preferable to use static analysis tools to limit errors in the generated code. You can use Simulink Design Verifier or Polyspace® Code Prover™ to perform the static analysis.</p> <p>If static analysis determines that sections of the code can have a division by zero, then add run-time protection into that section of the model (see example). Using a modified CRL or selecting the parameter Remove code that protects against division arithmetic exceptions protects division operations against divide-by-zero operations. However, this action does introduce additional computational and memory overhead.</p> <p>Use only one of the run-time protections (B, C or D) in a model. Using more than one option can result in redundant protection operations.</p>	
Rationale	A,B, C,D	Improve MISRA-C:2004 compliance of the generated code
References	<ul style="list-style-type: none"> • MISRA-C:2004 Rule 21.1 	
See Also	<ul style="list-style-type: none"> • “What Is Code Replacement?” and “Code Replacement Libraries” in the Simulink Coder documentation • “hisl_0002: Usage of Math Function blocks (rem and reciprocal)” 	

ID: Title	hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance
	<ul style="list-style-type: none">• “hisl_0005: Usage of Product blocks”• “hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions”
Last Changed	R2011b

ID: Title	hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance
Example	<p>Run-time divide by zero protection can be realized using a graphical function. Unique functions should be provided for each data type.</p> <div style="background-color: #fff9c4; padding: 10px; margin-bottom: 10px;"> <p style="text-align: right; margin-right: 20px;"><i>Graphical function to model divide-by-zero check</i></p> <pre style="margin: 0;">{d_int_pro = div_fun_int(b_int, c_int);... d_dbl_pro = div_fun_dbl(b_dbl, c_dbl, 10000.0, 0.001);}</pre> </div> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; width: 45%;"> <p>function result = div_fun_dbl(num, den, maxVal, eps)</p> </div> <div style="border: 1px solid black; padding: 5px; width: 45%;"> <p>function result = div_fun_int(num, den)</p> </div> </div> <div style="background-color: #fff9c4; padding: 10px; margin-top: 10px;"> <p style="text-align: right; margin-right: 20px;"><i>Graphical function to model divide-by-zero check</i></p> <pre style="margin: 0;">{d = div_fun(b,c);}</pre> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>function result = div_fun(num, den)</p> </div>

System Level

In this section...
“hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance” on page 6-36
“hisl_0402: Use of custom #pragma to improve MISRA-C:2004 compliance” on page 6-37
“hisl_0403: Use of char data type improve MISRA-C:2004 compliance” on page 6-38

hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance

ID: Title	hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance
Description	To improve the MISRA-C:2004 compliance of the generated code, encapsulate manually inserted code. This code includes, but is not limited to, C, Fortran, and assembly code.
Rationale	Improve MISRA-C:2004 compliance of the generated code
See Also	<ul style="list-style-type: none"> • “External Code Integration” in the Embedded Coder documentation. • “External Code Integration” in the Simulink Coder documentation.
Notes	<p>Simulink provides multiple methods for integrating existing code. The user is responsible for encapsulating the generated code.</p> <p>Encapsulation can be defined as “the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation”^a</p>
References	<ul style="list-style-type: none"> • MISRA-C: 2004 Rule 2.1
Last Changed	R2012a

^aBooch, Grady, R. Maksimchuk, M. Engle, B. Young, J. Conallen, K. Houston. *Object-Oriented Analysis and Design with Applications*. 3rd ed. Boston, MA: Addison-Wesley Professional, 2007.

hisl_0402: Use of custom #pragma to improve MISRA-C:2004 compliance

ID: Title	hisl_0402: Use of custom #pragma to improve MISRA-C:2004 compliance	
Description	To improve the MISRA-C:2004 compliance of the generated code, document user defined pragma. In the documentation, include:	
	A	Memory range (start and stop address)
	B	Intended use
	C	Justification for using a pragma
Rationale	Improve MISRA-C:2004 compliance of the generated code	
See Also	<ul style="list-style-type: none"> • “About Memory Sections” in the Embedded Coder documentation. • “Document Generated Code with Simulink Report Generator” in the Simulink Coder documentation. 	
Notes	The Simulink Report Generator™ documents pragmas.	
References	<ul style="list-style-type: none"> • MISRA-C: 2004 Rule 3.4 	
Last Changed	R2012a	

hisl_0403: Use of char data type improve MISRA-C:2004 compliance

ID: Title	hisl_0403: Use of char data type to improve MISRA-C:2004 compliance	
Description	To improve the MISRA-C:2004 compliance of the generated code with custom storage classes that use the Char data type, only use:	
	A	Plain char type for character values.
	B	Signed and unsigned char type for numeric values.
Rationale	Improve MISRA-C:2004 compliance of the generated code.	
See Also	<ul style="list-style-type: none"> • “Custom Storage Classes” in the Embedded Coder documentation. • “About Memory Sections” in the Embedded Coder documentation. • “Document Generated Code with Simulink Report Generator” in the Simulink Coder documentation. 	
References	<ul style="list-style-type: none"> • MISRA-C: 2004 Rule 6.1 • MISRA-C: 2004 Rule 6.2 	
Last Changed	R2012a	