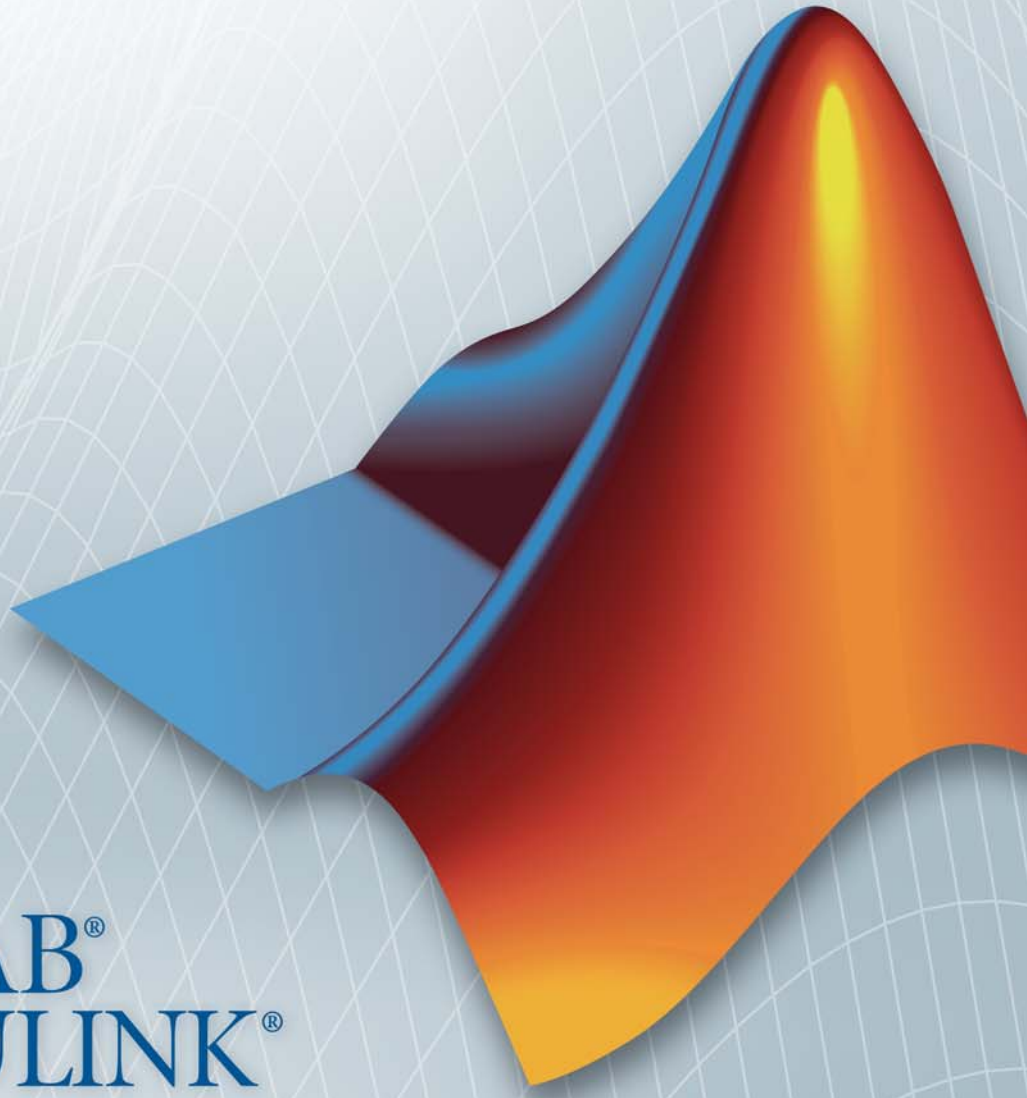
Simulink®
Modeling Guidelines for High-Integrity Systems



MATLAB®
&SIMULINK®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Modeling Guidelines for High-Integrity Systems

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)

Contents

1 Introduction

Motivation **1-2**

2 Block Considerations

Math Operations **2-2**
hisl_0001: Usage of Abs block **2-3**
hisl_0002: Usage of Math Function blocks (remainder and reciprocal) **2-5**
hisl_0003: Usage of Math Function blocks (square root) . . **2-7**
hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm) **2-9**
hisl_0005: Usage of Product blocks **2-12**

Ports & Subsystems **2-14**
hisl_0006: Usage of While Iterator blocks **2-15**
hisl_0007: Usage of While Iterator subsystems **2-17**
hisl_0008: Usage of For Iterator Blocks **2-20**
hisl_0009: Usage of For Iterator Subsystem blocks **2-22**
hisl_0010: Usage of If blocks and If Action Subsystem blocks **2-23**
hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks **2-25**
hisl_0012: Usage of triggered subsystems **2-27**
hisl_0012_b: Usage of function-call subsystems **2-28**

Signal Routing **2-29**
hisl_0013: Usage of data store blocks **2-30**
hisl_0015: Usage of Merge blocks **2-33**
hisl_0021: Consistent vector indexing method **2-35**
hisl_0022: Data type selection for index signals **2-36**
hisl_0023: Verification of model and subsystem variants . . **2-37**

Logic and Bit Operations **2-38**
hisl_0016: Usage of blocks that compute relational operators **2-39**
hisl_0017: Usage of blocks that compute relational operators (2) **2-41**
hisl_0018: Usage of Logical Operator block **2-42**
hisl_0019: Usage of Bitwise Operator block **2-43**

3 Configuration Parameter Considerations

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

Diagnostics **3-7**
hisl_0043: Configuration Parameters > Diagnostics > Solver **3-8**
hisl_0044: Configuration Parameters > Diagnostics > Sample Time **3-10**

Optimizations **3-13**
hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double) . . **3-14**
hisl_0046: Configuration Parameters > Optimization > Block reduction **3-15**
hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution **3-16**
hisl_0048: Configuration Parameters > Optimization > Application lifespan (days) **3-17**
hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold . . . **3-18**
hisl_0052: Configuration Parameters > Optimization > Data initialization **3-19**

hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values 3-20
hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions 3-21
hisl_0055: Prioritization of code generation objectives for high-integrity systems 3-22

4 Stateflow Chart Considerations

Chart Properties 4-2
hisf_0001: Mealy and Moore semantics 4-3
hisf_0002: User-specified state/transition execution order 4-5
hisf_0009: Strong data typing (Simulink and Stateflow boundary) 4-7
hisf_0011: Stateflow debugging settings 4-8

Chart Architecture 4-10
hisf_0003: Usage of bitwise operations 4-11
hisf_0004: Usage of recursive behavior 4-12
hisf_0007: Usage of junction conditions (maintaining mutual exclusion) 4-15
hisf_0010: Usage of transition paths (looping out of parent of source and destination objects) 4-16
hisf_0012: Chart comments 4-18
hisf_0013: Usage of transition paths (crossing parallel state boundaries) 4-19
hisf_0014: Usage of transition paths (passing through states) 4-21
hisf_0015: Strong data typing (casting variables and parameters in expressions) 4-22

5 MISRA-C:2004 Compliance Considerations

Modeling Style **5-2**
hisl_0061: Unique identifiers for clarity **5-3**
hisl_0062: Global variables in graphical functions **5-5**
hisl_0063: Length of user-defined function names to improve MISRA-C:2004 compliance **5-8**
hisl_0064: Length of user-defined type object names to improve MISRA-C:2004 compliance **5-9**
hisl_0065: Length of signal and parameter names to improve MISRA-C:2004 compliance **5-10**

Block Usage **5-11**
hisl_0020: Blocks not recommended for MISRA-C:2004 compliance **5-11**

Configuration Settings **5-12**
hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance **5-12**

Stateflow Chart Considerations **5-14**
hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance **5-14**
hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance **5-16**

1

Introduction

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-178B, IEC 61508, ISO 26262, and MISRA C®.

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 "MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB®, Simulink, and Stateflow®". 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-178B 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.

2

Block Considerations

- "Math Operations" on page 2-2
- "Ports & Subsystems" on page 2-14
- "Signal Routing" on page 2-29
- "Logic and Bit Operations" on page 2-38

Math Operations

In this section...

"hisl_0001: Usage of Abs block" on page 2-3

"hisl_0002: Usage of Math Function blocks (remainder and reciprocal)" on page 2-5

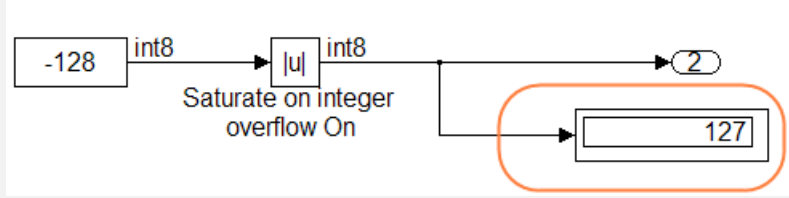
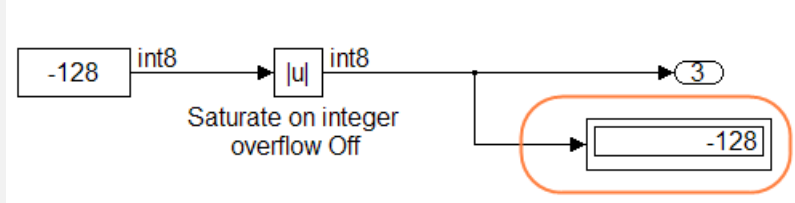
"hisl_0003: Usage of Math Function blocks (square root)" on page 2-7

"hisl_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)" on page 2-9

"hisl_0005: Usage of Product blocks" on page 2-12

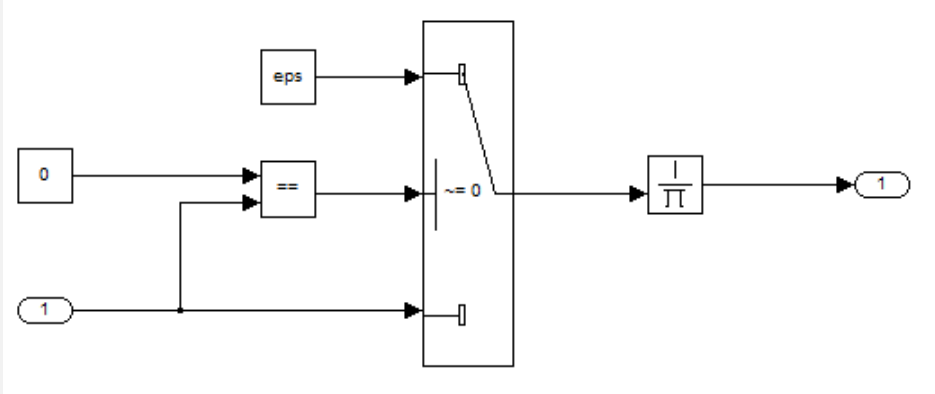
hisl_0001: Usage of Abs block

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	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. 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**, the absolute value of the most negative value represented by the data type has no affect.	
Rationale	A	Support generation of traceable code.
	B	Achieve consistent and expected behavior of model simulation and generated code.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178B > "Check usage of Math Operations blocks"** • **By Task > Modeling Standards for IEC-61508 > "Check usage of Math Operations blocks"** • **By Task > Modeling Standards for ISO-26262 > "Check usage of Math Operations blocks"**	

ID: Title	hisl_0001: Usage of Abs block
References	• IEC 61508-3, Table A.3 (3) 'Language subset' 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' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' ISO/DIS 26262-6, Table 7 (f) 'Control flow analysis' • DO-178B, Section 6.4.4.3c 'Structural Coverage Analysis Resolution (Dead Code)' • MISRA-C:2004, Rule 14.1 MISRA-C:2004, Rule 21.1
Last Changed	R2011a
Examples	 **Recommended**  **Not Recommended**

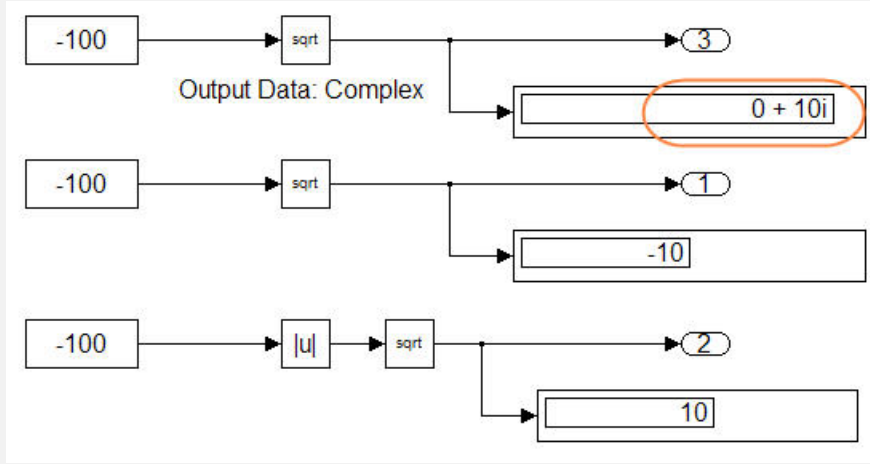
hisl_0002: Usage of Math Function blocks (remainder and reciprocal)

ID: Title	hisl_0002: Usage of Math Function blocks (remainder and reciprocal)	
Description	To support robustness of generated code, when using the Math Function block with remainder-after-division (`rem`) or array-reciprocal (`reciprocal`) functions,	
	A	Protect the input of the `reciprocal` function from going to zero.
	B	Protect the second input of the `rem` function from going to zero.
Note	You might get a divide-by-zero operation, resulting in an infinite (`Inf`) output value. To avoid overflows, protect the corresponding input from going to zero.	
Rationale	A, B	Protect against overflows.
Model Advisor Checks	**By Task > Modeling Standards for DO-178B > "Check for proper usage of Math blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1(b) 'Use of language subsets' ISO/DIS 26262-6, Table 1(d) 'Use of defensive implementation techniques' • DO-178B, Section 6.4.2.2 'Robustness Test Cases' DO-178B, Section 6.4.3 'Requirements-Based Testing Methods' • MISRA-C:2004, Rule 21.1	

ID: Title	hisl_0002: Usage of Math Function blocks (remainder and reciprocal)
Last Changed	R2011a
Examples	In the following example, when the input signal oscillates around zero, the output exhibits a large change in value. MathWorks recommends further protection against the large change in value. 

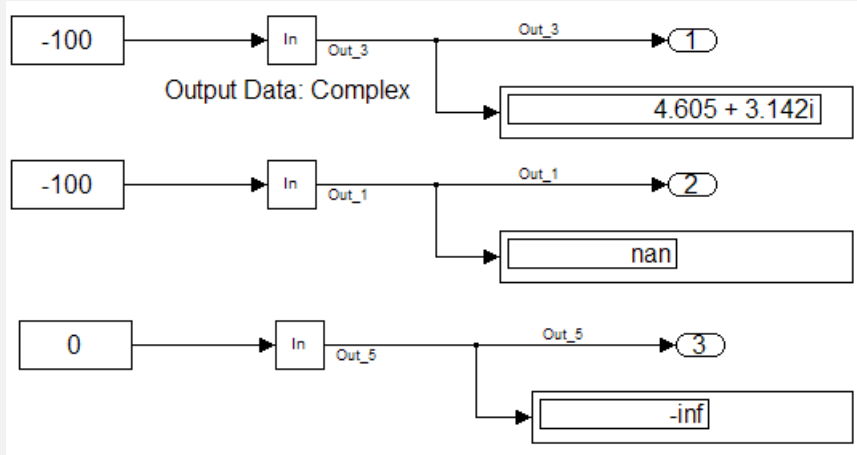
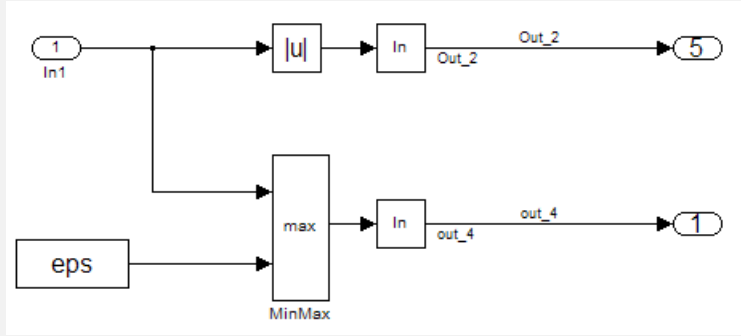
hisl_0003: Usage of Math Function blocks (square root)

ID: Title	hisl_0003: Usage of Math Function blocks (square root)	
Description	To support robustness of generated code, when using the Math Function block with the square root (`sqrt`) function parameter, do one of the following:	
	A	Account for complex numbers as the output.
	B	Account for negative values as the block output.
	C	Protect the input from going negative.
Notes	For negative input, the square root function takes the absolute value of the input and performs the square root operation. The square root function sets the sign of the output to negative, which might lead to undesirable results in the generated code.	
Rationale	A, B, C	Avoid undesirable results in generated code.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1(b) 'Use of language subsets' ISO/DIS 26262-6, Table 1(d) 'Use of defensive implementation techniques' • DO-178B, Section 6.4.2.2a 'Robustness Test Cases'	
Last Changed	R2011a	

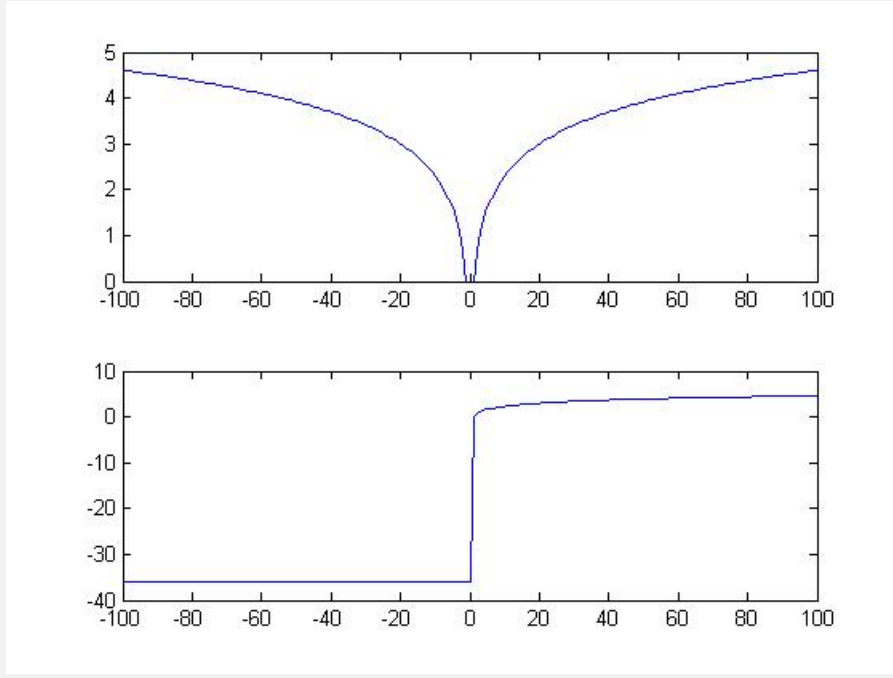
ID: Title	hisl_0003: Usage of Math Function blocks (square root)
Examples	

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 (`log`) or base 10 logarithm (`log10`) 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 (`-inf`) for zero values.	
Rationale	A, B, C	Support generation of robust code.
Model Advisor Checks	**By Task > Modeling Standards for DO-178B Checks > "Check for proper usage of Math blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1(b) 'Use of language subsets' ISO/DIS 26262-6, Table 1(d) 'Use of defensive implementation techniques' • DO-178B, Section 6.4.2.2a 'Robustness Test Cases' DO-178B, Sections 6.3.1g and 6.3.2g 'Algorithms are accurate'	
Last Changed	R2011a	

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



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 all divisor inputs from going to zero.
	B	In `Matrix(*)` mode, protect all divisor inputs from becoming singular input matrices.
	C	Set the model configuration parameter **Diagnostics > Data Validity > Signals > Division by singular matrix** to `error`.
Notes	When using Product blocks for element-wise divisions, you might get a divide by zero, resulting in a `NaN` output. To avoid overflows, protect all divisor inputs from going to zero. 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 all divisor inputs from becoming singular input matrices. 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.	
Rationale	A, B, C	Protect against overflows.
Model Advisor Checks	**By Task > Modeling Standards for DO-178B > "Check safety-related diagnostic settings for signal data"**	

ID: Title	hisl_0005: Usage of Product blocks
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262–6, Table 1(b) 'Use of language subsets' ISO/DIS 26262–6, Table 1(d) 'Use of defensive implementation techniques' • DO-178B, Section 6.4.2.2 'Robustness Test Cases' DO-178B, Section 6.4.3 'Requirements-Based Testing Methods' • MISRA-C:2004, Rule 21.1
Last Changed	R2011a

Ports & Subsystems

In this section...

"hisl_0006: Usage of While Iterator blocks" on page 2-15

"hisl_0007: Usage of While Iterator subsystems" on page 2-17

"hisl_0008: Usage of For Iterator Blocks" on page 2-20

"hisl_0009: Usage of For Iterator Subsystem blocks" on page 2-22

"hisl_0010: Usage of If blocks and If Action Subsystem blocks" on page 2-23

"hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks" on page 2-25

"hisl_0012: Usage of triggered subsystems" on page 2-27

"hisl_0012_b: Usage of function-call subsystems" on page 2-28

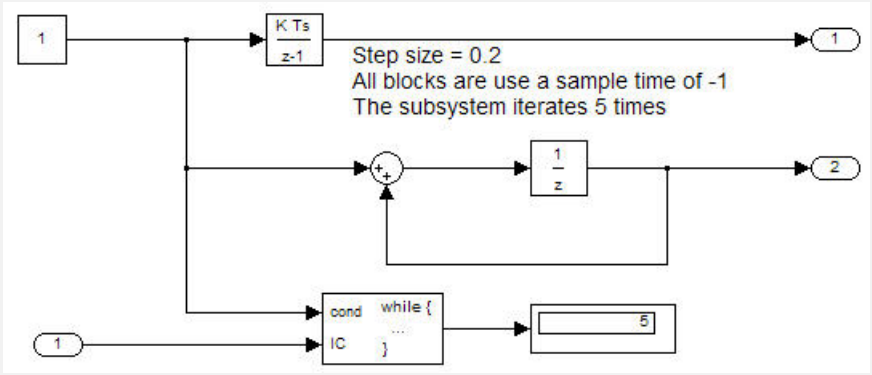
hisl_0006: Usage of While Iterator blocks

ID: Title	hisl_0006: Usage of While Iterator blocks	
Description	To support statistically deterministic 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.
	B	Consider selecting **Show iteration number port** to observe the iteration value during simulation.
Note	When you use While Iterator subsystems, MathWorks recommends setting 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. 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 whether the output values of the While Iterator block are correct.	
Rationale	A, B	Support generation of statistically deterministic code.
Model Advisor Checks	• **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 DO-178B > "Check usage of Ports and Subsystems blocks"**	

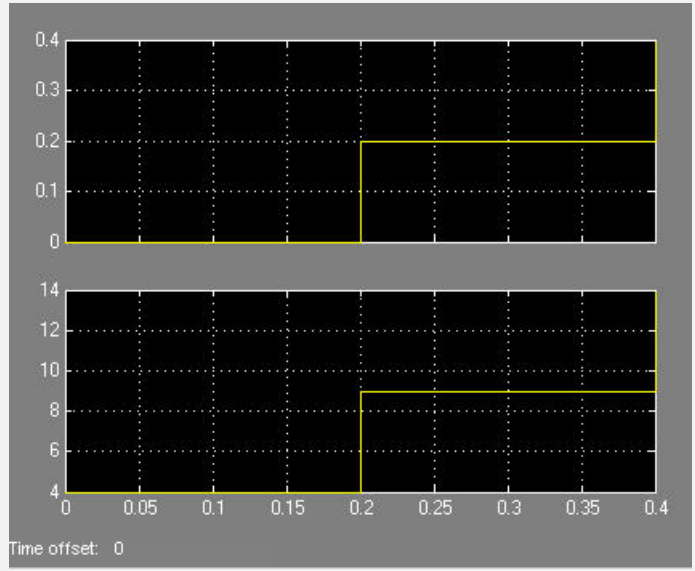
ID: Title	hisl_0006: Usage of While Iterator blocks
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.1e 'Review and Analyses of the High-Level Requirements: Conformance to standards' DO-178B, Section 6.3.2e 'Review and Analyses of the Low-Level Requirements: Conformance to standards' • MISRA-C:2004, Rule 21.1
Last Changed	R2011a

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 all 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	• **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 DO-178B > "Check usage of Ports and Subsystems blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.1e 'Review and Analyses of the High-Level Requirements: Conformance to standards' DO-178B, Section 6.3.2e 'Review and Analyses of the Low-Level Requirements: Conformance to standards' • MISRA-C:2004, Rule 21.1	

ID: Title	hisl_0007: Usage of While Iterator subsystems
Last Changed	R2011a
Examples	For iterative subsystems, the value `delta T` is nonzero for the first iteration only. For subsequent iterations, the value is zero. In the following example, in the output of the Sum block calculation that uses the unit delay, the Sum block calculation does not require `delta T`. The output of the Discrete-Time Integrator block displays the effect of the zero `delta T` value. 

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 generated code that is statistically deterministic, 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 hat can lead to execution-time overruns.	
Rationale	A, B, C, D	Support generation of statistically deterministic code.
Model Advisor Checks	• **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 DO-178B > "Check usage of Ports and Subsystems blocks"**	

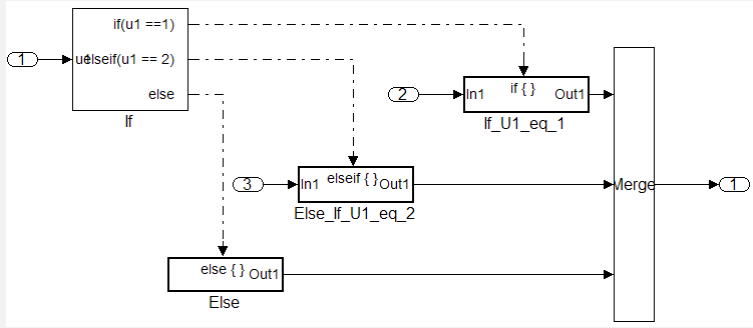
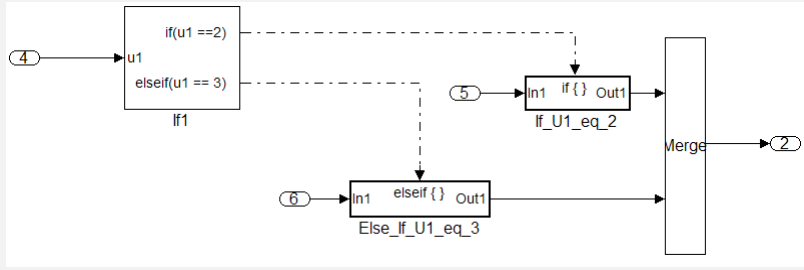
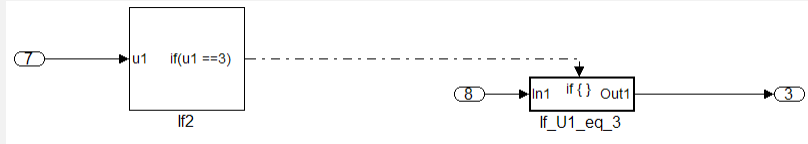
ID: Title	hisl_0008: Usage of For Iterator blocks
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.1e 'Review and Analyses of the High-Level Requirements: Conformance to standards' DO-178B, Section 6.3.2e 'Review and Analyses of the Low-Level Requirements: Conformance to standards' • MISRA-C:2004, Rule 13.6
Last Changed	R2011a

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	• **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 DO-178B > "Check usage of Ports and Subsystems blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset'; IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.4.2.2d 'Robustness Test Cases: (For Loops)' • MISRA-C:2004, Rule 13.6	
Last Changed	R2011a	
Examples	See "hisl_0007: Usage of While Iterator subsystems" on page 2-17.	

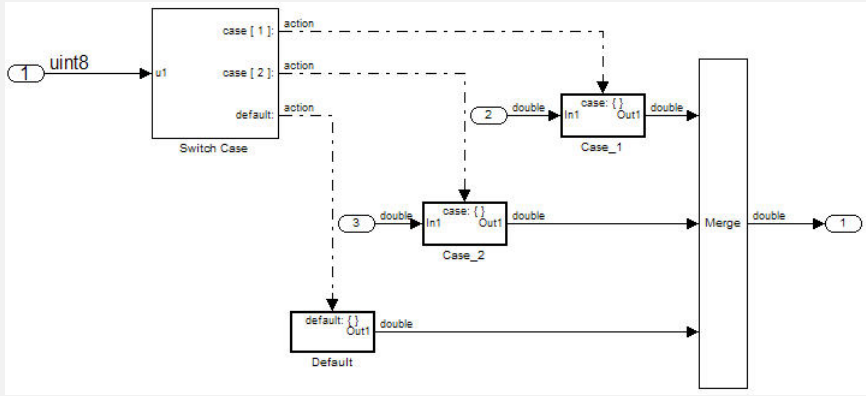
hisl_0010: Usage of If blocks and If Action Subsystem blocks

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 `Elseif` 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 2-39	
Notes	The combination of If and If Action Subsystem blocks enable conditional execution based on input conditions. When there is only an `if` branch, you do not need to include an `else` branch.	
Rationale	A, B	Support generation of verifiable code.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262–6, Table 1(b) 'Use of language subsets' ISO/DIS 26262–6, Table 1(d) 'Use of defensive implementation techniques' • 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	R2011a	

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

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 an If Action Subsystem block.
	C	Use an integer data type for the inputs to Switch Case blocks.
Prerequisites	"hisl_0016: Usage of blocks that compute relational operators" on page 2-39	
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	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262–6, Table 1(b) 'Use of language subsets' ISO/DIS 26262–6, Table 1(d) 'Use of defensive implementation techniques' • MISRA-C:2004, Rule 14.10	
See Also	db_0115: Simulink patterns for case constructs in the Simulink documentation.	

ID: Title	hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks
Last Changed	R2011a
Examples	The following graphic displays an example of providing a default path of execution using a “Default” block. 

hisl_0012: Usage of triggered subsystems

ID: Title	hisl_0012: Usage of triggered subsystems	
Description	To support unambiguous behavior, when using triggered subsystems,	
	A	Specify inherited (`-1`) sample times for all blocks in the subsystem, except Constant. Constant blocks can use infinite (`inf`) sample time.
	B	Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystem.
Rationale	A, B	Support unambiguous behavior.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming • ISO/DIS 26262–6, Table 1(b) 'Use of language subsets' ISO/DIS 26262–6, Table 1(d) 'Use of defensive implementation techniques'	
Last Changed	R2011a	

hisl_0012_b: Usage of function-call subsystems

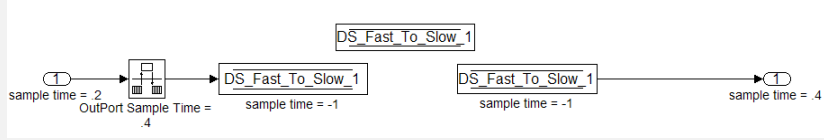
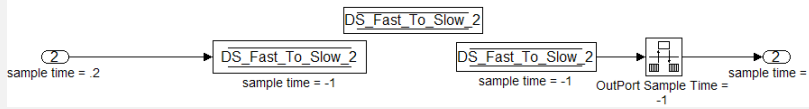
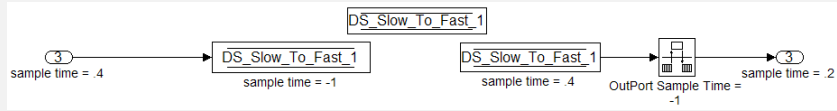
ID: Title	hisl_0012_b: Usage of function-call subsystems	
Description	To support unambiguous behavior, when using function-call subsystems,	
	A	Specify inherited (`-1`) sample times for all blocks in the subsystem, except Constant. Constant blocks can use infinite (`inf`) sample time.
	B	Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystem.
Rationale	A, B	Support unambiguous behavior.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262–6, Table 1(b) 'Use of language subsets' ISO/DIS 26262–6, Table 1(d) 'Use of defensive implementation techniques'	
Last Changed	R2011a	

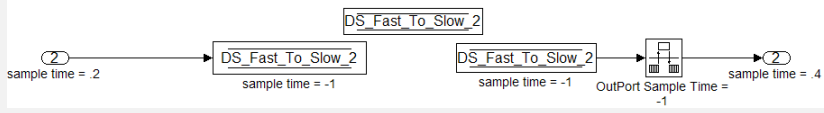
Signal Routing

In this section...
“hisl_0013: Usage of data store blocks” on page 2-30
“hisl_0015: Usage of Merge blocks” on page 2-33
“hisl_0021: Consistent vector indexing method” on page 2-35
“hisl_0022: Data type selection for index signals” on page 2-36
“hisl_0023: Verification of model and subsystem variants” on page 2-37

hisl_0013: Usage of data store blocks

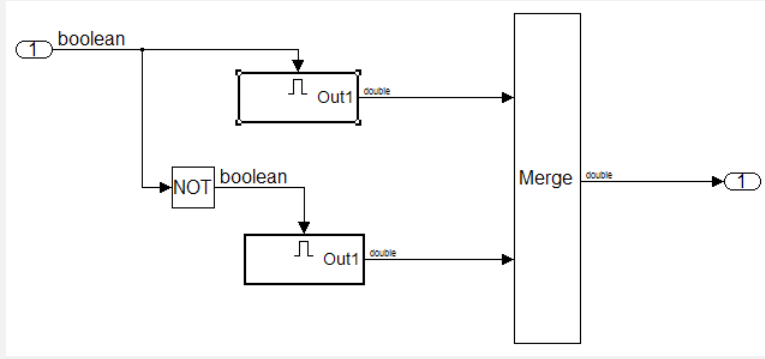
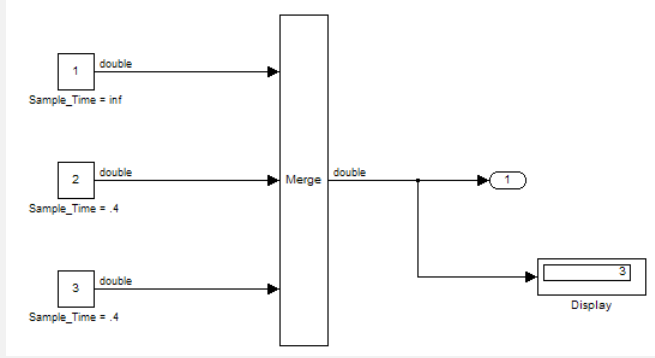
ID: Title	hisl_0013: Usage of data store blocks	
Description	To support statistically deterministic behavior across different sample times or models, when using data store blocks, including Data Store Memory, Data Store Read, and Data Store Write,	
	A	• In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, under **Data Store Memory Block**, set the following parameters to `error`: - **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.
	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.
Notes	The sorting algorithm in Simulink does not take into account data coupling between models and atomic subsystems. Using data store memory blocks can have significant effects on your software verification effort. Models and subsystems that use only inports and outports to pass data are clean, deterministic, and verifiable interfaces in the generated code.	
Rationale	A, B, C	Support statistically deterministic behavior across different sample times or models.
Model Advisor Checks	**By Task > Modeling Standards for DO-178B > "Check safety-related diagnostic settings for data store memory"**	

ID: Title	hisl_0013: Usage of data store blocks
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.3b 'Review and Analyses of the Software Architecture: Consistency'
Last Changed	R2011a
Examples	The following examples use Rate Transition blocks to provide deterministic data coupling in multirate systems • For fast-to-slow transitions: Set the rate of the slow sample time on either the Rate Transition block or the Data Store Write block.  Do not place the Rate Transition block after the Data Store Read block.  • For slow-to-fast transitions: If the Rate Transition block is after the Data Store Read block, specify the slow rate on the Data Store Read block. 

ID: Title	hisl_0013: Usage of data store blocks
	If the Rate Transition block is before the Data Store Write block, use the inherited sample time for all blocks. 

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 in all cases only one subsystem executes during a time step.
	C	Clear the Merge block parameter **Allow unequal port widths**.
Notes	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. 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.	
Rationale	A, B, C	Avoid unambiguous behavior.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1(b) 'Use of language subsets' ISO/DIS 26262-6, Table 1(d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.3b 'Reviews and Analyses of the Software Architecture: Consistency'	
Last Changed	R2011a	

ID: Title	hisl_0015: Usage of Merge blocks
Examples	 **Recommended**
	 **Not Recommended**

hisl_0021: Consistent vector indexing method

ID: Title	hisl_0021: Consistent vector indexing method	
Description	Within a model,	
	A	Use a consistent vector indexing method for all blocks. Blocks for which you should set the indexing method include: • Index Vector • Multiport Switch • Assignment • Selector • For Iterator
Rationale	A	Reduce the risk of introducing errors due to inconsistent indexing.
References	• DO-178B, Section 6.3.2b 'Accuracy and Consistency of Low-Level Requirements' • IEC 61508–3, Table A.3 (3) 'Language subset' IEC 61508–3, Table A.4 (5) 'Design and coding standards' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (f) 'Use of unambiguous graphical representation'	
See Also	"cgsl_0101: Zero-based indexing"	
Last Changed	R2011a	

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.
	Blocks that use a signal index include: Assignment Index Vector Multiport Switch Stateflow vector indexing Signal Routing Interp n-D Direct lookup n-D Selector / Matrix Selector Lookup Table n-D block (internal index type selection)	
Rationale	A	Prevent unexpected results that can occur with rounding operations for floating-point data types.
	B	Enable access to all data in a vector.
References	• IEC 61508–3, Table A.3 (2) 'Strongly typed programming language' IEC 61508–3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (c) 'Enforcement of strong typing' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.4f 'Accuracy and Consistency of Source Code'	
Last Changed	R2011a	

hisl_0023: Verification of model and subsystem variants

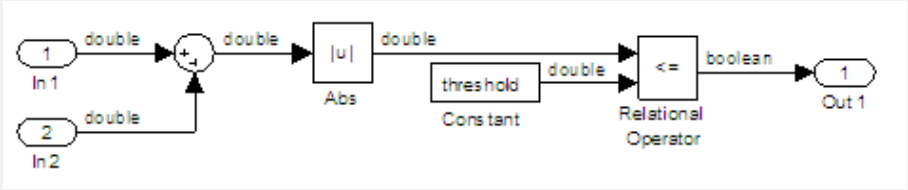
ID: Title	hisl_0023: Verification of model and subsystem variants	
Description	When verifying that a model is consistent with generated code, do one of the following:	
	A	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, disable variants in generated code by setting **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	• DO-178B, Section 6.4.4.2 'Structural Coverage Analysis and Section' DO-178B, Section 6.4.4.3 'Structural Coverage Analysis Resolution' • IEC 61508–3, Table A.4 (7) 'Use of trusted / verified software modules and components'	
Last Changed	R2010b	

Logic and Bit Operations

In this section...
“hisl_0016: Usage of blocks that compute relational operators” on page 2-39
“hisl_0017: Usage of blocks that compute relational operators (2)” on page 2-41
“hisl_0018: Usage of Logical Operator block” on page 2-42
“hisl_0019: Usage of Bitwise Operator block” on page 2-43

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	Due to floating-point precision issues, do not test floating-point expressions for equality (==) or inequality (~=). The software might not evaluate the comparison of floating-point expressions correctly. 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.	
Rationale	A	Improve model robustness.
Model Advisor Checks	• **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 DO-178B > "Check usage of Logic and Bit Operations blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2g 'Algorithms are accurate' • MISRA-C:2004, Rule 13.3	

ID: Title	hisl_0016: Usage of blocks that compute relational operators
See Also	"hisl_0017: Usage of blocks that compute relational operators (2)" on page 2-41
Last Changed	R2011a
Examples	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 (`eps`) and the magnitude of the numbers. The following pattern shows how to test two double-precision input signals, `In1` and `In2`, for equality. 

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	• **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 DO-178B > "Check usage of Logic and Bit Operations blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset'; IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (c) 'Enforcement of strong typing' • DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2g 'Algorithms are accurate' • MISRA-C:2004, Rule 12.6	
See Also	"hisl_0016: Usage of blocks that compute relational operators" on page 2-39	
Last Changed	R2011a	

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 3-14	
Rationale	A	Avoid ambiguous behavior of generated code.
Model Advisor Checks	• **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 DO-178B > "Check usage of Logic and Bit Operations blocks"** • **By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (c) 'Enforcement of strong typing' • DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2g 'Algorithms are accurate' • MISRA-C:2004, Rule 12.6	
Last Changed	R2011a	

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	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (c) 'Enforcement of strong typing' ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • MISRA-C:2004, Rule 12.7	
See Also	"hisf_0003: Usage of bitwise operations" on page 4-11in the Simulink documentation	
Last Changed	R2011a	

3

Configuration Parameter Considerations

- "Solver" on page 3-2
- "Diagnostics" on page 3-7
- "Optimizations" on page 3-13

Solver

In this section...

"hisl_0040: Configuration Parameters > Solver > Simulation time" on page 3-3

"hisl_0041: Configuration Parameters > Solver > Solver options" on page 3-4

"hisl_0042: Configuration Parameters > Solver > Tasking and sample time options" on page 3-5

hisl_0040: Configuration Parameters > Solver > Simulation time

ID: Title	hisl_0040: Configuration Parameters > Solver > Simulation time	
Description	For models in high-integrity systems, in the Configuration Parameters dialog box, on the Solver pane, set parameters for simulation time as follows:	
	A	Set **Start time** to `0.0`.
	B	Set **Stop time** to any positive value that is less than the value of **Application lifespan (days)**.
Note	Simulink allows nonzero start times for simulation. However, production code generation requires a zero start time. By default, **Application lifespan (days)** is `inf`. If you do not change this setting, any positive value for **Stop time** is valid and this setting has no effect on generated code. You specify **Stop time** in seconds and **Application lifespan (days)** is in days.	
Rationale	A	Generate code that is valid for production code generation.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets'	
See Also	• "hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)" on page 3-17 • Solver Pane section of the Simulink documentation	
Last Changed	R2011a	

hisl_0041: Configuration Parameters > Solver > Solver options

ID: Title	hisl_0041: Configuration Parameters > Solver > Solver options	
Description	For models in high-integrity systems, in the Configuration Parameters dialog box, on the Solver pane, set parameters for solvers as follows:	
	A	Set **Type** to `Fixed-step`.
	B	Set **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	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets'	
See Also	"Solver Pane" in the Simulink documentation	
Last Changed	R2011a	

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 in high-integrity systems, in the Configuration Parameters dialog box, on the Solver pane, set parameters for tasking and sample time as follows:	
	A	Set **Periodic sample time constraint** to `Specified` and assign appropriate values to **Sample time properties**. **Caution** If you use a referenced model as a reusable function, set **Periodic sample time constraint** to `Ensure sample time independent`.
	B	Set **Tasking mode for periodic sample times** to `SingleTasking` or `MultiTasking`.
	C	Clear the parameter **Automatically handle data transfers between tasks**.
Notes	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. 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.	
Rationale	A, B, C	Support fully specified models and unambiguous code.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' • DO-178B, Section 6.3.4e 'Source code is traceable to low-level requirements'	

ID: Title	hisl_0042: Configuration Parameters > Solver > Tasking and sample time options
See Also	“Solver Pane” in the Simulink documentation
Last Changed	R2011a

Diagnostics

In this section...
“hisl_0043: Configuration Parameters > Diagnostics > Solver” on page 3-8
“hisl_0044: Configuration Parameters > Diagnostics > Sample Time” on page 3-10

hisl_0043: Configuration Parameters > Diagnostics > Solver

ID: Title	hisl_0043: Configuration Parameters > Diagnostics > Solver	
Description	For models in high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set parameters for solver diagnostics as follows:	
	A	Set model solver diagnostics as follows: • Set **Algebraic loop** to `error`. • Set **Minimize algebraic loop** to `error`. • Set **Block priority violation** to `error` iff you are using block priorities. • Set **Unspecified inheritability of sample times** to `error`. • Set **Automatic solver parameter selection** to `error`. • Set **State name clash** to `warning`.
Note	Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.	

If Diagnostic Parameter...	Is Not Set Correctly,...
Algebraic loop	Automatic breakage of algebraic loops can go undetected and affect the predictability of the order of block execution.
Minimize algebraic loop	Automatic breakage of algebraic loops can go undetected and affect the predictability of the order of block execution.
Block priority violation	Block execution order can include undetected conflicts that might

ID: Title	hisl_0043: Configuration Parameters > Diagnostics > Solver

	affect the predictability of the order of block 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 can go undetected and affect the operation of generated code.
State name clash	A name being used for more than one state might go undetected.

You can set the following solver diagnostic parameters to any value:

Min step size violation
Sample hit time adjusting
Consecutive zero crossings violation
Solver data inconsistency
Extraneous discrete derivative signals

Rationale	A	Support generation of robust and unambiguous code.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178B > "Check safety-related model referencing settings"** • **By Task > Modeling Standards for DO-178B > "Check safety-related diagnostic settings for solvers"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' • DO-178B, 6.3.3e 'Software architecture conforms to standards'	
See Also	• "Diagnostics Pane: Solver" in the Simulink documentation • jc_0021: Model diagnostic settings in the Simulink documentation	
Last Changed	R2011a	

hisl_0044: Configuration Parameters > Diagnostics > Sample Time

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time	
Description	For models in high-integrity systems, in the Configuration Parameters dialog box, on the Diagnostics pane, set parameters for sample time diagnostics to `error`:	
	A	In the **Diagnostics** pane of the Configuration Parameters dialog box, set the following parameters to `error`: **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** If the target system does not allow preemption between tasks that have equal priority, set **Tasks with equal priority** to `none`.
Note	Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.	

If Diagnostic Parameter...	Is Not Set Correctly,...
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
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.

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time	
	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 nondeterministic behavior in a target system that allows preemption.
	Tasks with equal priority	Two asynchronous tasks with equal priority might go undetected and show nondeterministic 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-178B > "Check safety-related diagnostic settings for sample time"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' • DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.3b 'Software architecture is consistent'	

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time
See Also	“Diagnostics Pane: Sample Time” in the Simulink documentation
Last Changed	R2011a

Optimizations

In this section...

"hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)" on page 3-14

"hisl_0046: Configuration Parameters > Optimization > Block reduction" on page 3-15

"hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution" on page 3-16

"hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)" on page 3-17

"hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold" on page 3-18

"hisl_0052: Configuration Parameters > Optimization > Data initialization" on page 3-19

"hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values" on page 3-20

"hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions" on page 3-21

"hisl_0055: Prioritization of code generation objectives for high-integrity systems" on page 3-22

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-178B > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO/DIS 26262-6, Table 1 (c) 'Enforcement of strong typing' • DO-178B, 6.3.1e 'High-level requirements conform to standards' DO-178B, 6,3,2e 'Low-level requirements conform to standards' • MISRA-C:2004, Rule 12.6	
Last Changed	R2011a	

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 with no 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	**By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"**	
References	• 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-178B, Section 6.3.4e 'Source code is traceable to low-level requirements'	
See Also	"Block reduction" in the Simulink documentation	
Last Changed	R2010b	

hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution

ID: Title	hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution	
Description	To facilitate structural testing, in the Configuration Parameters dialog box, on the **Optimization** pane,	
	A	Consider clearing the **Conditional input branch execution** parameter.
Note	The Model Coverage tool in the Simulink® Verification and Validation™ product does not account for this optimization. This optimization can result in reporting 100% coverage, but for the same test cases, code coverage might be less than 100%.	
Rationale	A	Facilitate structural testing.
Model Advisor Checks	**By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.4 (6) 'Structure-based testing' • DO-178B, Section 6.4.4.2 'Structural Coverage Analysis: Test coverage of software structure is achieved'	
See Also	"Conditional input branch execution" in the Simulink documentation	
Last Changed	R2010b	

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

ID: Title	hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)	
Description	To support the robustness and behavior 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. Setting **Application lifespan (days)** to `inf` guarantees that the simulation time is always less than the application lifespan.	
Rationale	A	Support robustness of behavior of systems that run continuously.
Model Advisor Checks	**By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2g 'Algorithms are accurate'	
See Also	• "Application lifespan (days)" in the Simulink documentation • "hisl_0040: Configuration Parameters > Solver > Simulation time" on page 3-3	
Last Changed	R2011a	

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.
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	• IEC 61508-3, Table A.3 (3) 'Language Subset' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets'	
See Also	"Loop unrolling threshold" in the Simulink documentation	
Last Changed	R2011a	

hisl_0052: Configuration Parameters > Optimization > Data initialization

ID: Title	hisl_0052: Configuration Parameters > Optimization > Data initialization	
Description	To support complete definition of data and to ensure that all internal and external data is initialized 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 state 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	**By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • MISRA-C:2004, Rule 9.1	
See Also	Information about the following parameters in the Simulink documentation: • "Remove root level I/O zero initialization" • "Remove internal data zero initialization"	
Last Changed	R2011a	

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-178B > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO/DIS 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • MISRA-C:2004, Rule 14.1	
See Also	"Remove code from floating-point to integer conversions that wraps out-of-range values" in the Simulink documentation	
Last Changed	R2011a	

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-178B > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.3 (3) 'Language Subset' IEC 61508-3 Table A.4 (3) 'Defensive Programming' • ISO/DIS 26262-6, Table 1(b) 'Use of language subsets' ISO/DIS 26262-6, Table 1(d) 'Use of defensive implementation techniques' • MISRA-C:2004, Rule 21.1	
See Also	"Remove code that protects against division arithmetic exceptions" in the Simulink documentation	
Last Changed	R2011a	

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 safety precaution 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	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. Including the `ROM`, `RAM`, and `Execution efficiency` objectives with a lower priority in the list enables efficiency optimizations that do not conflict with `Safety precautions` and `Traceability` in the active configuration. The resulting parameter configuration should be reviewed to ensure that all safety requirements are met.	
Rationale	A, B	By using the Code Generation Advisor, you can ensure that the selection of configuration parameters conforms to desired objectives and are consistently enforced.
References	• DO-178B, Section 6.3.4e '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/DIS 26262–6, Table 1(b) 'Use of language subsets' ISO/DIS 26262–6, Table 1(d) 'Use of defensive implementation techniques'	

ID: Title	hisl_0055: Prioritized configuration objectives for high-integrity systems
See also	• "Set Objectives — Code Generation Advisor Dialog Box" • "Setting Up Configuration Sets" • "cgsl_0301: Prioritization of code generation objectives for code efficiency"
Last Changed	R2011a

4

Stateflow Chart Considerations

- "Chart Properties" on page 4-2
- "Chart Architecture" on page 4-10

Chart Properties

In this section...

"hisf_0001: Mealy and Moore semantics" on page 4-3

"hisf_0002: User-specified state/transition execution order" on page 4-5

"hisf_0009: Strong data typing (Simulink and Stateflow boundary)" on page 4-7

"hisf_0011: Stateflow debugging settings" on page 4-8

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`.
	B	Apply consistent settings to all Stateflow charts in a model.
Note	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. In Mealy charts, actions are associated with transitions. In the Moore charts, actions are associated with states. 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.	
Rationale	A, B	Promote a clear modeling style.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178B > "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"**	
References	• IEC 61508-3,Table A.7 (2) 'Simulation/modeling' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' • DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards' DO-178B, Section 6.3.3b 'Software architecture is consistent' DO-178B, Section 6.3.3e 'Software architecture conform to standards'	

ID: Title	hisf_0001: Mealy and Moore semantics
See Also	“Building Mealy and Moore Charts” in the Stateflow documentation
Last Changed	R2011a

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	Selecting **User specified state/transition execution order** restricts the dependency of a Stateflow chart semantics on the geometric position of parallel states and transitions. 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. Selecting **Show Transition Execution Order** displays the transition testing order.	
Rationale	A, B, C	Promote an unambiguous modeling style.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178B > "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"**	

ID: Title	hisf_0002: User-specified state/transition execution order
References	This guideline supports adhering to: • IEC 61508-3, Table A.3 (3) 'Language subset' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (f) 'Use of unambiguous graphical representation' • DO-178B, Section 6.3.3b 'Software architecture is consistent' DO-178B, Section 6.3.3e 'Software architecture conform to standards '
See Also	The following topics in the Stateflow documentation • "Transition Testing Order in Multilevel State Hierarchy" • "Execution Order for Parallel States"
Last Changed	R2011a

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	• **By Task > Modeling Standards for IEC 61508 > "Check usage of Stateflow constructs"** • **By Task > Modeling Standards for ISO 26262 > "Check usage of Stateflow constructs"**	
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO/DIS 26262-6, Table 1 (c) 'Enforcement of strong typing' • DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards' DO-178B, Section 6.3.2g 'Algorithms are accurate' • MISRA-C:2004, Rules 10.1, 10.2, 10.3 and 10.4	
Last Changed	R2011a	

hisf_0011: Stateflow debugging settings

ID: Title	hisf_0011: Stateflow debugging settings	
Description	To protect against unreachable code and indeterminate execution time,	
	A	Select the following run-time diagnostics: • In the Configuration Parameters dialog box, on the **Simulation Target** pane, select: **Enable debugging/animation** **Enable overflow detection (with debugging)** • In the Stateflow Debugging window, select **State Inconsistency** **Transition Conflict** **Detect Cycles** **Data Range**
	B	For each truth table in the model, in the **Settings** menu of the Truth Table Editor, set the following parameters to `Error`: **Underspecified** **Overspecified**
Notes	The truth table settings do not affect the generated code. If the error condition is not reached during simulation, the error message is not triggered for code generation.	
Rationale	A, B	Protect against unreachable code and unpredictable execution time.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178B > "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"**	

ID: Title	hisf_0011: Stateflow debugging settings
References	• IEC 61508-3, Table A.7 (2) 'Simulation/modeling' • ISO/DIS 26262 Table 1 (d) 'Use of defensive implementation techniques' • DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards'
Last Changed	R2011a

Chart Architecture

In this section...

"hisf_0003: Usage of bitwise operations" on page 4-11

"hisf_0004: Usage of recursive behavior" on page 4-12

"hisf_0007: Usage of junction conditions (maintaining mutual exclusion)" on page 4-15

"hisf_0010: Usage of transition paths (looping out of parent of source and destination objects)" on page 4-16

"hisf_0012: Chart comments" on page 4-18

"hisf_0013: Usage of transition paths (crossing parallel state boundaries)" on page 4-19

"hisf_0014: Usage of transition paths (passing through states)" on page 4-21

"hisf_0015: Strong data typing (casting variables and parameters in expressions)" on page 4-22

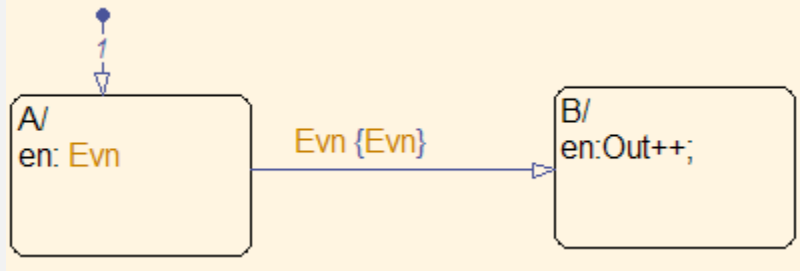
hisf_0003: Usage of bitwise operations

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	**By Task > Modeling Standards for MAAB > Stateflow > "Check for bitwise operations in Stateflow charts"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (c) 'Enforcement of strong typing' • DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards' DO-178B, Section 6.3.2g '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	R2011a	

hisf_0004: Usage of recursive behavior

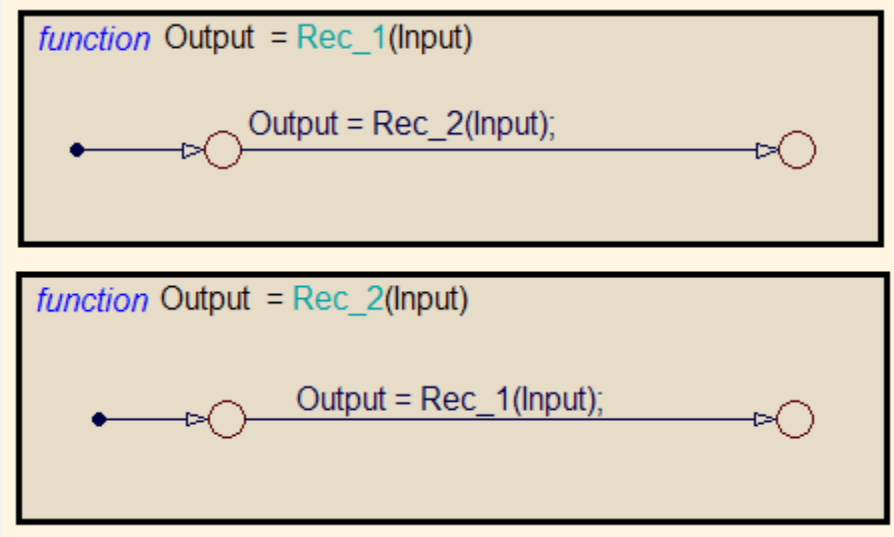
ID: Title	hisf_0004: Usage of recursive behavior	
Description	To ensure deterministic behavior, avoid using design patterns that include unbounded recursive behavior. Recursive behavior is bound if you do the following:	
	A	Use an explicit termination condition that is local to the recursive call.
	B	Make sure the termination condition is always reached.
Notes	This rule only applies if a chart is a classic Stateflow chart. If "hisf_0001: Mealy and Moore semantics" on page 4-3 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 deterministic behavior.
References	• IEC 61508-3, Table B.1 (6) 'Limited use of recursion' • ISO/DIS 26262-6, Table 9 (j) 'No recursions' • DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards' DO-178B, Section 6.3.2g 'Algorithms are accurate' • MISRA-C:2004, Rule 16.2	
Last Changed	R2011a	

ID: Title	hisf_0004: Usage of recursive behavior
Examples	There are multiple patterns in Stateflow that can result in unbounded recursion.



Recursive Function Calls

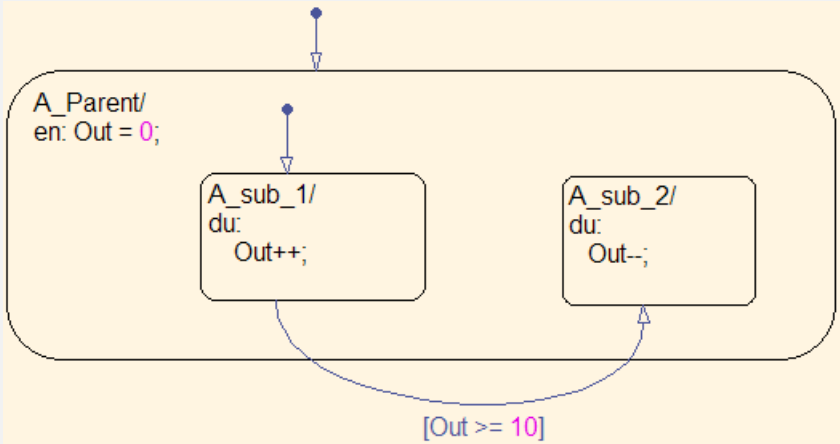
When the default state A is entered, event `Evn` is broadcast in the entry action of A. `Evn` results in a recursive call of the interpretation algorithm. Since A is active, the outgoing transition of A is tested. Since the current event `Evn` matches the transition event (and because of the absence 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.

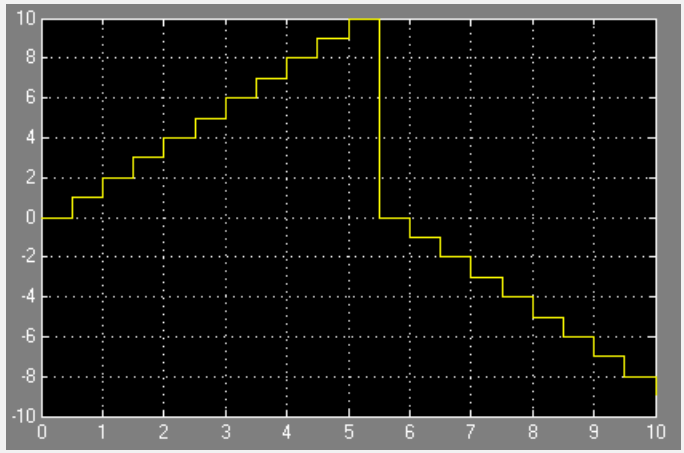
ID: Title	hisf_0004: Usage of recursive behavior
	 **Recursive Function Calls**

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	• DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1d 'High-level requirements are verifiable' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2d 'Low-level requirements are verifiable' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards'	
Last Changed	R2010b	

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	Transitions that loop out of the parent of the source and destination objects are typically unintentional and cause the parent to deactivate.	
	A	Avoid using these transitions.
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.	
Rationale	A	Promote a clear modeling style.
References	• DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards' DO-178B, Section 6.3.2g 'Algorithms are accurate'	
Last Changed	R2010b	
Examples		

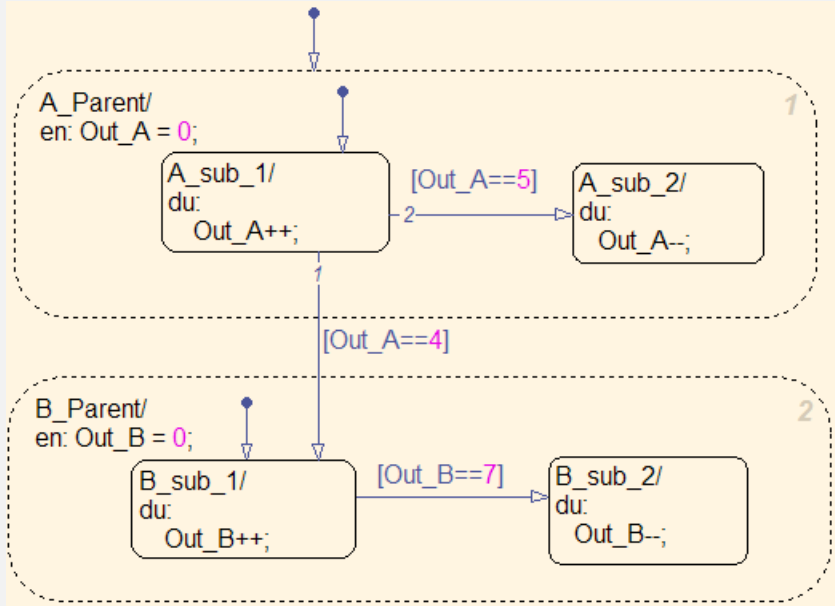
ID: Title	hisf_0010: Usage of transition paths (looping out of parent of source and destination objects
	

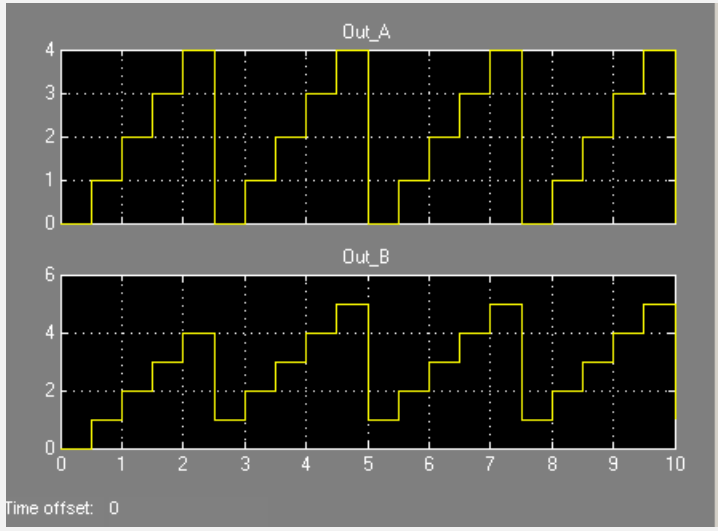
hisf_0012: Chart comments

ID: Title	hisf_0012: Chart comments	
Description	To enhance traceability between generated code and a model,	
	A	Add comments to the following Stateflow objects: In R2008b and higher: • Transitions In R2008a and lower: • Transitions • States
Notes	You can us this guideline to maintain a modeling language subset in high-integrity projects.	
Rationale	A	Enhance traceability between generated code and the corresponding model.
References	• DO-178B, Section 6.3.4e 'Source code is traceable to low-level requirements'	
Last Changed	R2010b	

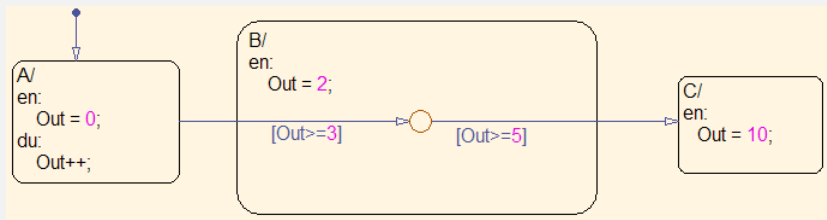
hisf_0013: Usage of transition paths (crossing parallel state boundaries)

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.
Last Changed	R2010b	
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.	



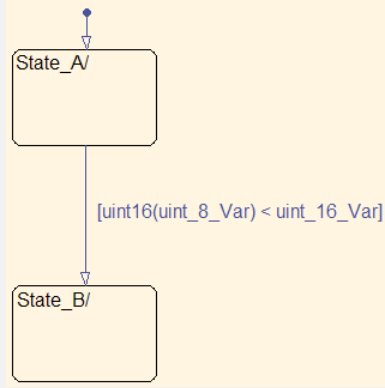
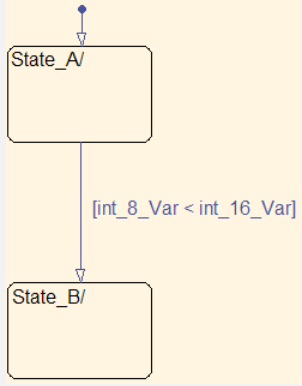
ID: Title	hisf_0013: Usage of transition paths (crossing parallel state boundaries)
	

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 that add no 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	• DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards'	
Last Changed	R2010b	
Examples		

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: • 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	• DO-178B, Section 6.3.1b 'High-level requirements are accurate and consistent' DO-178B, Section 6.3.1e 'High-level requirements conform to standards' DO-178B, Section 6.3.1g 'Algorithms are accurate' DO-178B, Section 6.3.2b 'Low-level requirements are accurate and consistent' DO-178B, Section 6.3.2e 'Low-level requirements conform to standards' DO-178B, Section 6.3.2g 'Algorithms are accurate'	
Last Changed	R2010b	

ID: Title	hisf_0015: Strong data typing (casting variables and parameters in expressions)
Examples	 **Recommended**  **Not Recommended**

5

MISRA-C:2004 Compliance Considerations

- "Modeling Style" on page 5-2
- "Block Usage" on page 5-11
- "Configuration Settings" on page 5-12
- "Stateflow Chart Considerations" on page 5-14

Modeling Style

In this section...

"hisl_0061: Unique identifiers for clarity" on page 5-3

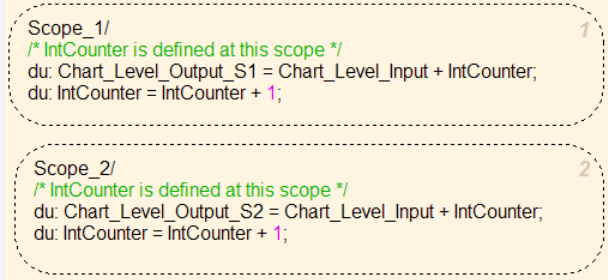
"hisl_0062: Global variables in graphical functions" on page 5-5

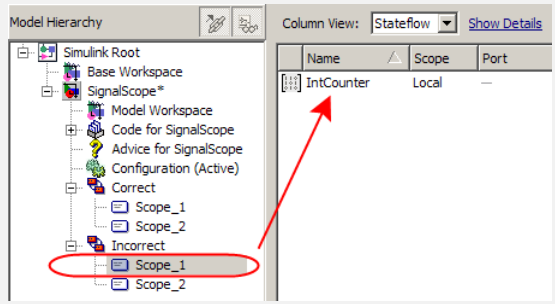
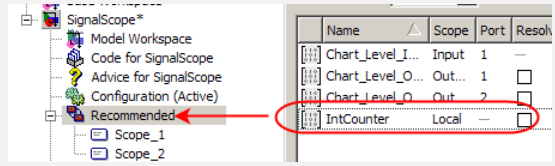
"hisl_0063: Length of user-defined function names to improve MISRA-C:2004 compliance" on page 5-8

"hisl_0064: Length of user-defined type object names to improve MISRA-C:2004 compliance" on page 5-9

"hisl_0065: Length of signal and parameter names to improve MISRA-C:2004 compliance" on page 5-10

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 automatically 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	• MISRA-C: 2004 5.6 • DO-178B, Section 6.3.2b 'Accuracy and Consistency of Low-Level Requirements' • IEC 61508–3, Table A.3 (3) 'Language subset' IEC 61508–3, Table A.4 (5) 'Design and coding standards' • ISO/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (e) 'Use of established design principles' ISO/DIS 26262-6, Table 1 (h) 'Use of naming conventions'	
See Also	"Construction of Symbols" in the Simulink® Coder™ documentation	
Last Changed	R2011a	
Examples	In the following example, two states use identifier `IntCounter`.  The identifier `IntCounter` is defined for two states, `Scope_1` and `Scope_2`.	

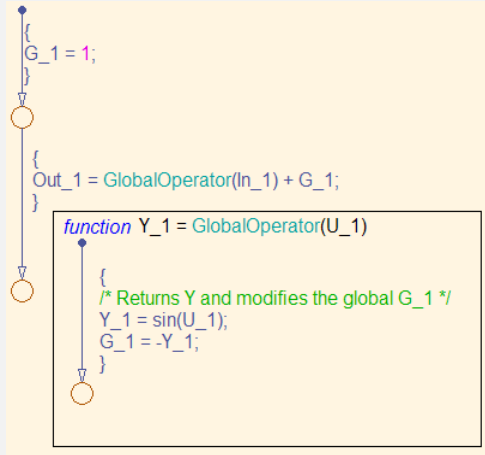
ID: Title	hisl_0061: Unique identifiers for clarity
	 **Not Recommended**
	To clarify the model, create unique identifiers—for example,*IntCounter_S1* and *IntCounter_S2*—or define *IntCounter* at the parent level.  **Recommended**

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	
	A	Do not use the data in the calling expression if a value is assigned to the data in that function.
Rationale	A	Enhance readability of a model by removing ambiguity in the values of global variables.
References	• 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/DIS 26262-6, Table 1 (b) 'Use of language subsets' ISO/DIS 26262-6, Table 1 (f) 'Use of unambiguous graphical representation' ISO/DIS 26262-6, Table 1 (h) 'Use of naming conventions' • DO-178B, Section 6.3.4f 'Accuracy and Consistency of Source Code' • MISRA-C: 2004 12.2 MISRA-C: 2004 12.4	
Last Changed	R2011a	
Examples	The basic expression is Y = f(U) + G where in the function `G` is assigned a value. This modeling pattern is realized:	

In a...	By Using...
Model	Data stores
Stateflow chart	Functions
MATLAB code	Subfunctions

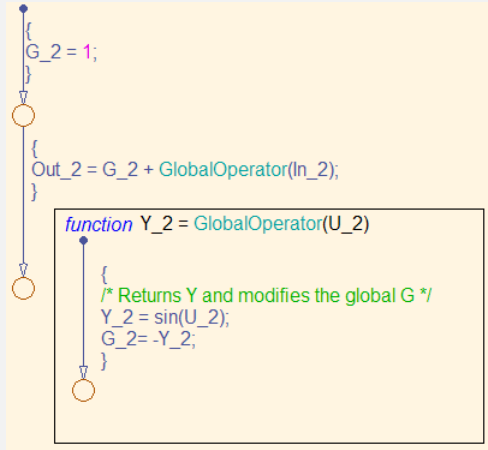
In the following example, the function `GlobalOperator` overwrites the initial value of *`G_1`*,




```
static real_T GlobalOperator_1(real_T U_1)
{
  real_T Y_1;

  /* Rreturns Y and modifies the global G_1 */
  Y_1 = sin(U_1);
  DWork.G_1 = -Y_1;
  return Y_1;
}
```

In the next example, the function uses the initial value of 1 for global variable *G_2* before the chart tries to assign the variable another value. The generated code omits the assignment of *G_2* to negative *Y_2*. (If the chart uses *G_2* at a later point, the chart uses the updated value of negative *Y_2*.)




```
static real_T GlobalOperator_2(real_T U_2)
{
  real_T Y_2;

  /* Returns Y and modifies the global G */
  Y_2 =sin(U_2);
  DWork.G_2 = -Y_2;
  return Y_2;
}
```

Code generator behavior is consistent and predictable.

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 generated code when working with Subsystem blocks with the block parameter **Function name options** set to `User specified`:	
	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	• 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	To improve MISRA-C:2004 compliance of the generated code, limit the length of data object names to 31 characters or fewer for: • Simulink.AliasType • Simulink.NumericType • Simulink.Variant • Simulink.Bus • Simulink.BusElement • Simulink.StructType • Simulink.StructElement • Simulink.EnumeratedType
Rationale	The length of the type definitions in the generated code name might result in a MISRA-C:2004 violation.
References	• 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	To improve compliance with MISRA-C:2004 in the generated code, limit the length of signal and parameter names to 31 characters or fewer when using any of the following storage classes: • 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	• MISRA-C:2004 Rule 5.1
Prerequisites	"hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance"
Last Changed	R2011a

Block Usage

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 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	Following this recommendation does not guarantee generation of MISRA-C:2004 compliant code. However, following this and other modeling guidelines increases the compliance of the generated code. Choose Simulink **Help > Block Support Table > Simulink** to view the block support table. Blocks with the footnote (4) in the Block Support Table are classified as "Not Recommended for production code."	
Rationale	A,B	Improve MISRA-C:2004 compliance of generated code.
Model Advisor Checks	**By Product > Embedded Coder > "Check for blocks not recommended for MISRA-C:2004 compliance"**	
References	MISRA-C:2004	
Last Changed	R2011a	

Configuration Settings

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

<table>
<tr><th>ID: Title</th><th colspan="2">hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance</th></tr>
<tr><td>Description</td><td colspan="2">To improve MISRA-C:2004 compliance of generated code,</td></tr>
<tr><td></td><td>A</td><td>Set the following model configuration parameters as specified:
<table>
<tr><th>Pane / Configuration Parameter</th><th>Value</th></tr>
<tr><td colspan="2">Diagnostics > Data Validity</td></tr>
<tr><td>Model Verification block enabling</td><td><code>Disable All</code></td></tr>
<tr><td colspan="2">Code Generation pane</td></tr>
<tr><td>System target file</td><td>ERT-based target</td></tr>
<tr><td colspan="2">Code Generation > Interface pane</td></tr>
<tr><td>Support: non-finite numbers</td><td>Cleared (<code>off</code>)</td></tr>
<tr><td>Support: continuous time</td><td>Cleared (<code>off</code>)</td></tr>
<tr><td>Support: non-inlined S-functions</td><td>Cleared (<code>off</code>)</td></tr>
<tr><td>MAT-file logging</td><td>Cleared (<code>off</code>)</td></tr>
<tr><td>Target function library</td><td><code>C89/C90 (ANSI)</code></td></tr>
<tr><td colspan="2">Code Generation > Code Style pane</td></tr>
<tr><td>Parenthesis level</td><td><code>Maximum (Specify precedence with parentheses)</code></td></tr>
</table></td></tr>
</table>

ID: Title	hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance	
		Code Generation > Symbols pane **Maximum identifier length** : 31
Note	Following this recommendation does not guarantee generation of MISRA-C:2004 compliant code. However, following this and other modeling guidelines increases the compliance of the generated code.	
Rationale	A	Improve MISRA-C:2004 compliance of generated code.
Model Advisor Checks	**By Product > Embedded Coder > "Check configuration parameters for MISRA-C:2004 compliance"**	
References	• MISRA-C:2004	
Last Changed	R2011a	

Stateflow Chart Considerations

In this section...

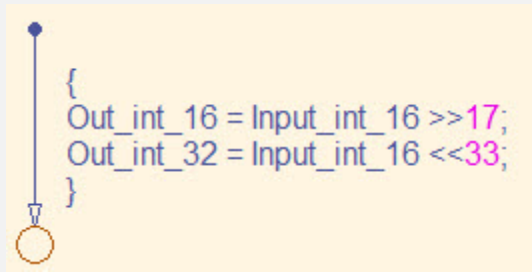
"hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance" on page 5-14

"hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance" on page 5-16

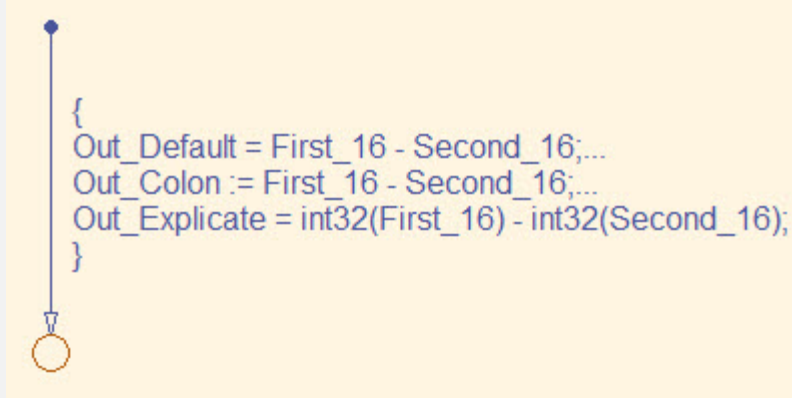
hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance

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
	B	Left-shift operations greater than the bit-width of the output type
Note	Following this recommendation does not guarantee generation of MISRA-C:2004 compliant code. However, following this and other modeling guidelines increases the likelihood of compliance.	
Rationale	A,B	To avoid shift operations in the generated code that might be a MISRA-C:2004 violation.
References	• MISRA-C:2004 Rule 12.7	
Prerequisites	"hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance"	

ID: Title	hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance
Last Changed	R2011a

Example	In the first equation, shifting 17 bits to the right pushes all 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.
	 
	
	`void stateflow_shift_passed_step(void)`
	`{`
	`  Out_int_16 = (int16_T)(Input_int_16 >> 17);`
	`  Out_int_32 = Input_int_16 << 33;`
	`}`

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	Following this recommendation does not guarantee generation of MISRA-C:2004 compliant code. However, following this and other modeling guidelines increases the likelihood of compliance.	
Rationale	A,B	To avoid shift operations in the generated code that might be a MISRA-C:2004 violation.
References	• MISRA-C:2004 Rule 10.1 • MISRA-C:2004 Rule 10.4	
Prerequisites	"hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance"	
Last Changed	R2011a	
Example	The example shows the default behavior and both methods of controlling the casting (explicitly type casting and using the colon operator). 	

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

```
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;
}
```