Simulink®

Modeling Guidelines for High-Integrity Systems

R2013**b**

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

Revision History

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

Contents

1 Introduction

2 Simulink Block Considerations

3 Stateflow Chart Considerations

4 MATLAB Function Block Considerations

5 Configuration Parameter Considerations

6 MISRA-C:2004 Compliance Considerations

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

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

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

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

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

Guideline Template

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

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

2

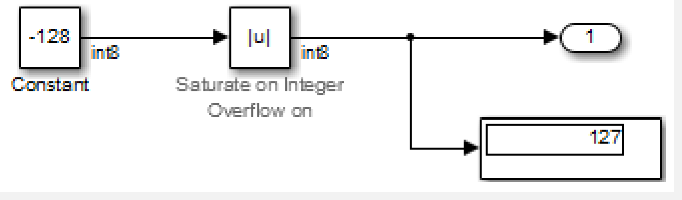
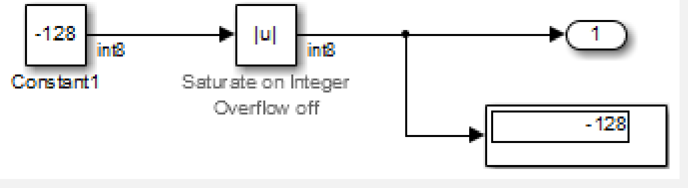
Simulink Block Considerations

Math Operations

In this section...

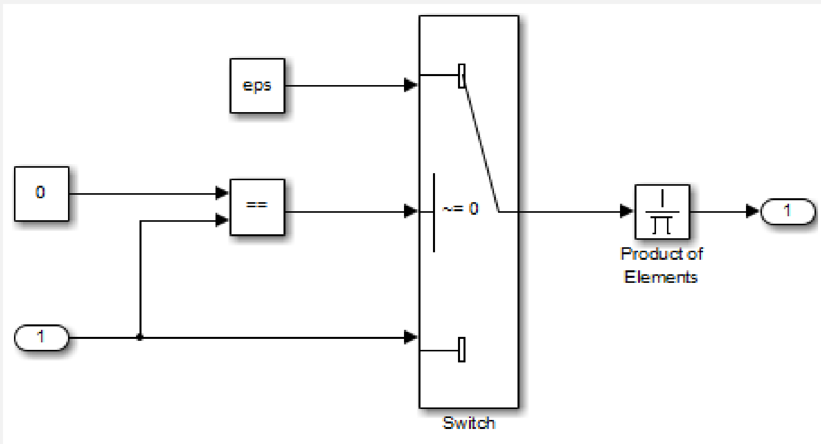
hisl_0001: Usage of Abs block

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**, absolute value calculations in the simulation and generated code might not be consistent or expected.	
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-178C/DO-331 > "Check usage of Math Operations blocks"** • **By Task > Modeling Standards for IEC-61508 > "Check usage of Math Operations blocks"** • **By Task > Modeling Standards for EN 50128 > "Check usage of Math Operations blocks"** • **By Task > Modeling Standards for ISO-26262 > "Check usage of Math Operations blocks"**	

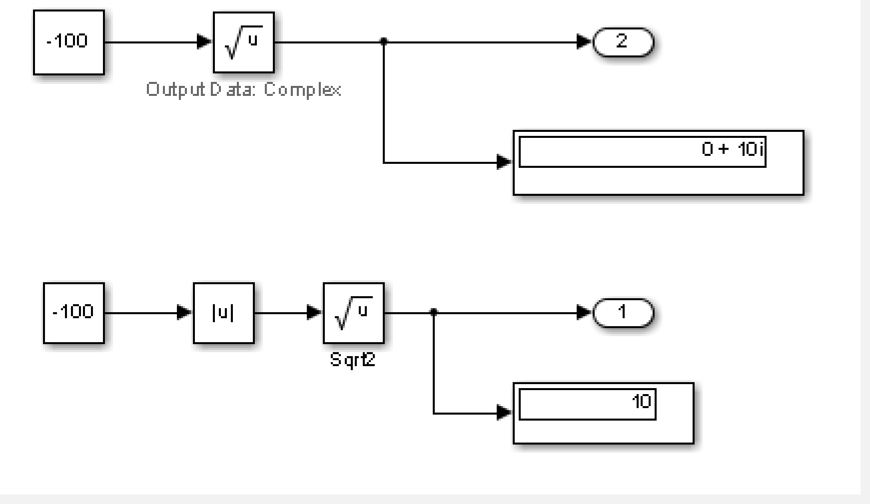
ID: Title	hisl_0001: Usage of Abs block
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' IEC 61508-3, Table B.8 (3) 'Control Flow Analysis' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' ISO 26262-6, Table 7 (f) 'Control flow analysis' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.19 (3) 'Control Flow Analysis' • DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' • MISRA-C:2004, Rule 14.1 MISRA-C:2004, Rule 21.1
Last Changed	R2013b
Examples	 **Recommended**  **Not Recommended**

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

<table>
<tr><th>ID: Title</th><th colspan="2">hisl_0002: Usage of Math Function blocks (rem and reciprocal)</th></tr>
<tr><td rowspan="3">Description</td><td colspan="2">To support robustness of generated code, when using the Math Function block with remainder-after-division (<code>rem</code>) or array-reciprocal (<code>reciprocal</code>) functions:</td></tr>
<tr><td>A</td><td>Protect the input of the <code>reciprocal</code> function from going to zero.</td></tr>
<tr><td>B</td><td>Protect the second input of the <code>rem</code> function from going to zero.</td></tr>
<tr><td>Note</td><td colspan="2">You can get a divide-by-zero operation, resulting in an infinite (<code>Inf</code>) output value for the <code>reciprocal</code> function, or a Not-a-Number (<code>NaN</code>) output value for the <code>rem</code> function. To avoid overflows or undefined values, protect the corresponding input from going to zero.</td></tr>
<tr><td>Rationale</td><td>A, B</td><td>Protect against overflows and undefined numerical results.</td></tr>
<tr><td>Model Advisor Checks</td><td colspan="2">By Task > Modeling Standards for DO-178C/DO-331 > “Check usage of Math blocks”</td></tr>
<tr><td>References</td><td colspan="2">IEC 61508-3, Table A.3 (3) ’Language subset’
IEC 61508-3, Table A.4 (3) ’Defensive programming’
ISO 26262-6, Table 1(b) ’Use of language subsets’
ISO 26262-6, Table 1(d) ’Use of defensive implementation techniques’
EN 50128, Table A.4 (11) ’Language Subset’
EN 50128, Table A.3 (1) ’Defensive Programming’
DO-331, Section MB.6.3.2.g ’Algorithms are accurate”
MISRA-C:2004, Rule 21.1</td></tr>
</table>

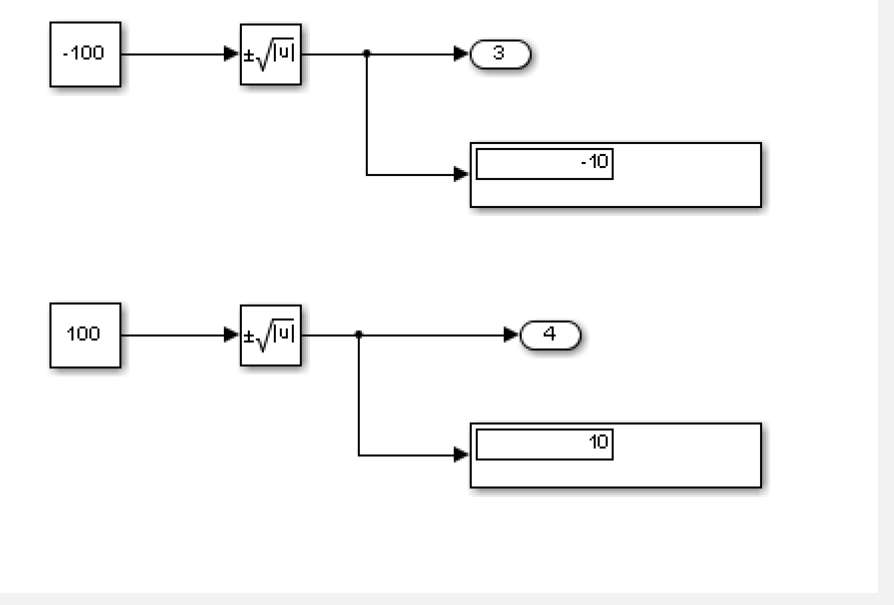
ID: Title	hisl_0002: Usage of Math Function blocks (rem and reciprocal)
Last Changed	R2013b
Examples	In the following example, when the input signal oscillates around zero, the output exhibits a large change in value. You need further protection against the large change in value. 

hisl_0003: Usage of Square Root blocks

ID: Title	hisl_0003: Usage of Square Root blocks	
Description	To support robustness of generated code, when using the Square Root block, do one of the following:	
	A	Account for complex numbers as the output.
	B	Protect the input from going negative.
Rationale	A, B	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 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate'	
Last Changed	R2013b	
Examples		

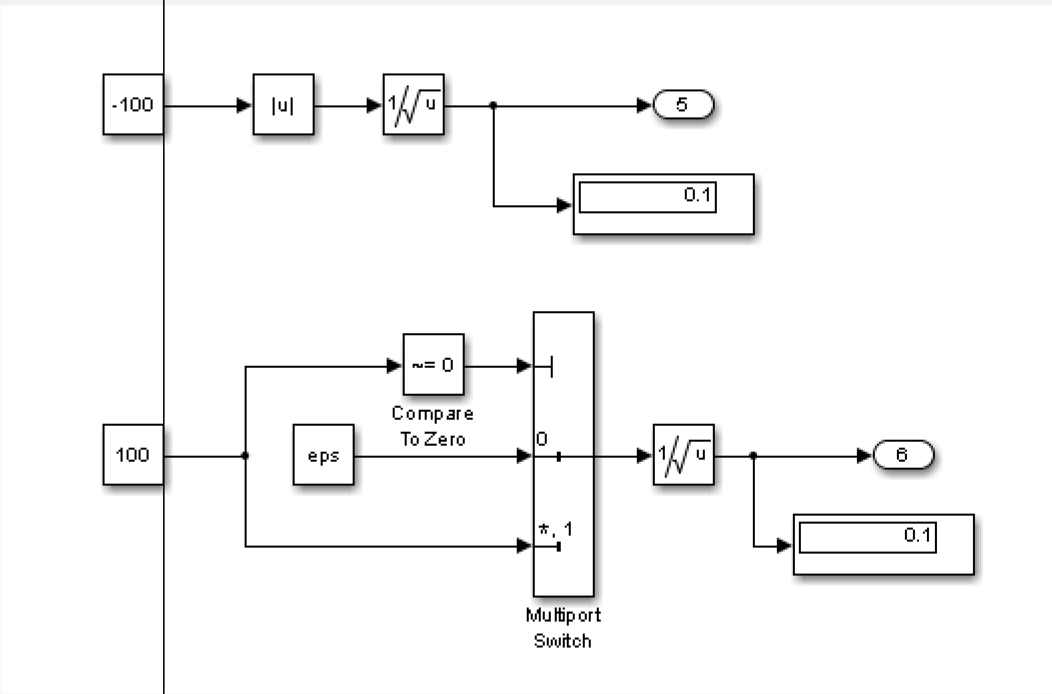
hisl_0027: Usage of Signed Square Root blocks

ID: Title	hisl_0027: Usage of Signed Square Root blocks
Description	To support robustness of generated code, when using the Signed Square Root block, account for negative block output values.
Notes	For negative input, the signed square root function takes the absolute value of the input and performs the square root operation. The signed square root function sets the sign of the output to negative, which might lead to undesirable results in the generated code.
Rationale	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 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate'

ID: Title	hisl_0027: Usage of Signed Square Root blocks
Last Changed	R2013b
Examples	

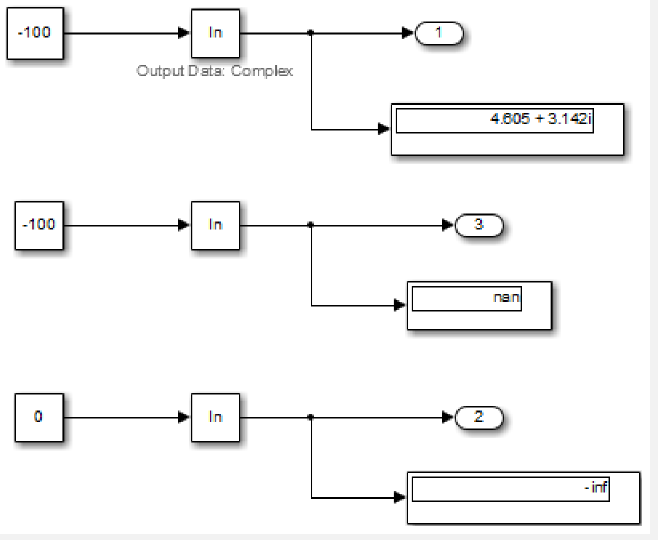
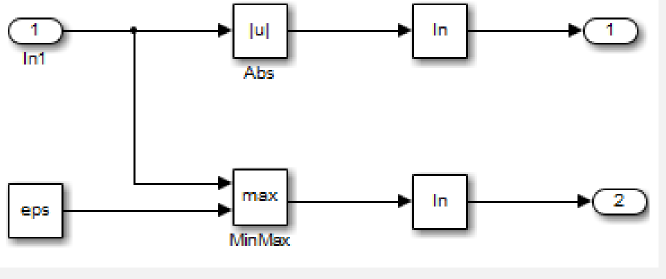
hisl_0028: Usage of Reciprocal Square Root blocks

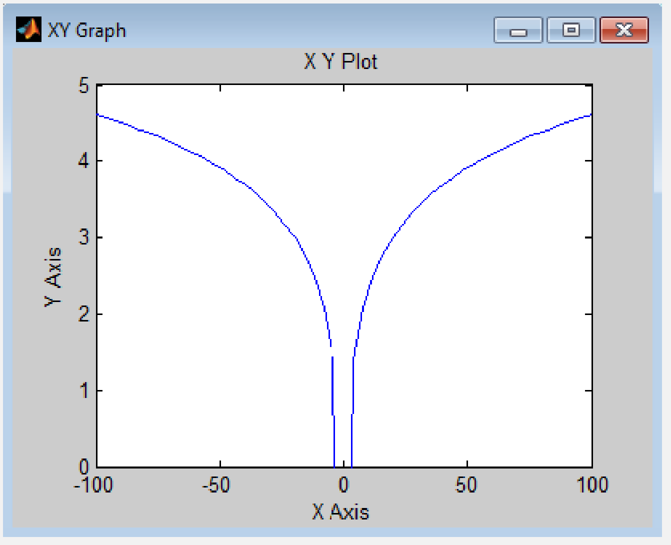
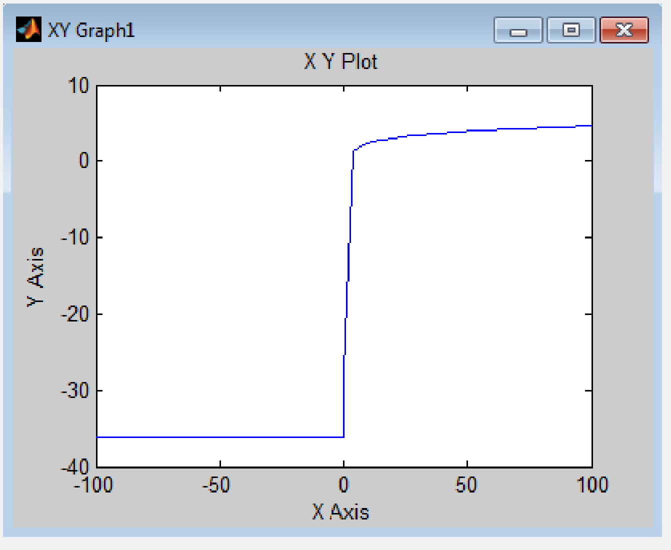
ID: Title	hisl_0028: Usage of Reciprocal Square Root blocks	
Description	To support robustness of generated code, when using the Reciprocal Square Root block, do one of the following:	
	A	Protect the input from going negative.
	B	Protect the input from going to zero.
Note	You can get a divide-by-zero operation, resulting in an `(Inf)` output value for the reciprocal function. To avoid overflows or undefined values, protect the corresponding input from going to zero.	
Rationale	A, B	Avoid undesirable results in generated code.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate'	

ID: Title	hisl_0028: Usage of Reciprocal Square Root blocks
Last Changed	R2013b
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-178C/DO-331 Checks > "Check usage of Math blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate"	
Last Changed	R2013b	

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 divisor inputs from going to zero.
	B	In `Matrix(*)` mode, protect divisor inputs from becoming singular input matrices.
	C	Set the model configuration parameter **Diagnostics > Data Validity > Signals > Division by singular matrix** to `error`.
Notes		When using Product blocks for element-wise divisions, you might get a divide by zero, resulting in a `NaN` output. To avoid overflows, protect divisor inputs from going to zero. When using Product blocks to compute the inverse of a matrix, or a matrix division, you might get a divide by a singular matrix. This division results in a `NaN` output. To avoid overflows, protect divisor inputs from becoming singular input matrices. 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-178C/DO-331 > "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 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.4.2.2 'Robustness Test Cases' DO-331, Section MB.6.4.3 'Requirements-Based Testing Methods' • MISRA-C:2004, Rule 21.1
Last Changed	R2013b

Ports & Subsystems

In this section...

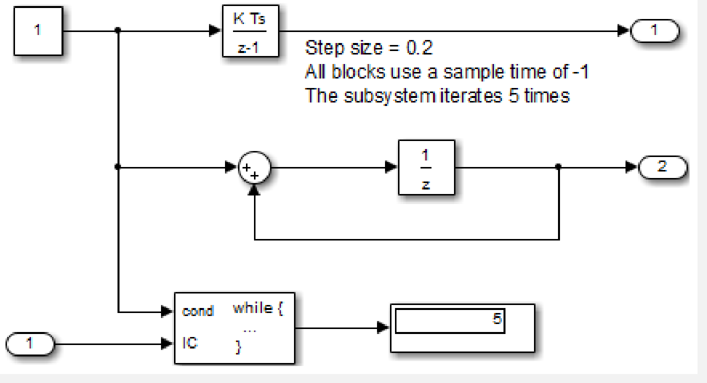
hisl_0006: Usage of While Iterator blocks

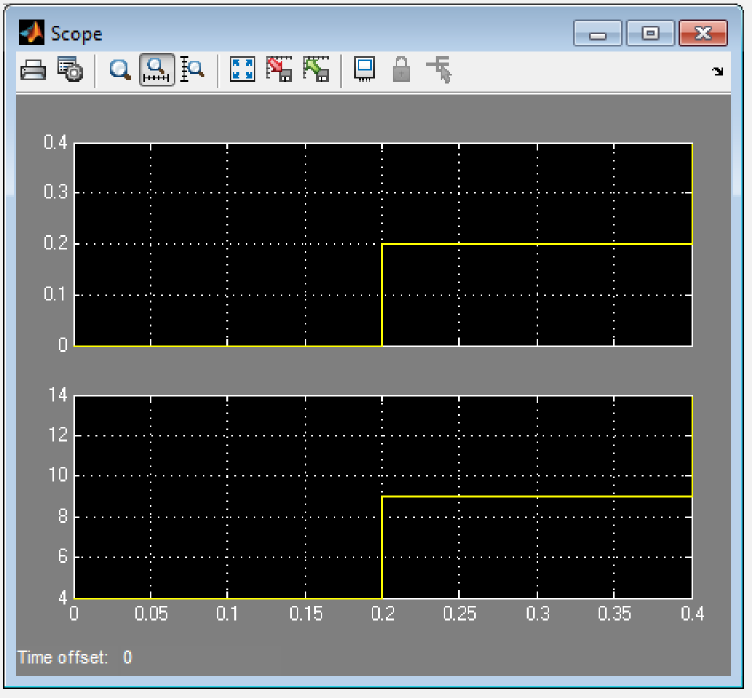
ID: Title	hisl_0006: Usage of While Iterator blocks	
Description	To support bounded iterative behavior in the generated code when using the While Iterator block, in the While Iterator block parameters dialog box:	
	A	Set **Maximum number of iterations** to a positive integer value; do not set value to —1 for unlimited.
	B	Consider selecting **Show iteration number port** to observe the iteration value during simulation.
Note	When you use While Iterator subsystems, set the maximum number of iterations. If you use an unlimited number of iterations, the generated code might include infinite loops, which lead to execution-time overruns. To observe the iteration value during simulation and determine whether the loop reaches the maximum number of iterations, select the While Iterator block parameter **Show iteration number port**. If the loop reaches the maximum number of iterations, verify the output values of the While Iterator block.	
Rationale	A, B	Support bounded iterative in the generated 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 EN 50128 > “Check usage of Ports and Subsystems blocks”** • **By Task > Modeling Standards for DO-178C/DO-331 > “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 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' • MISRA-C:2004, Rule 21.1
Last Changed	R2013b

hisl_0007: Usage of While Iterator subsystems

ID: Title	hisl_0007: Usage of While Iterator subsystems	
Description	To support unambiguous behavior, when using While Iterator subsystems,	
	A	Specify inherited (`-1`) or constant (`inf`) sample times for the blocks within the subsystems.
	B	Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems.
Rationale	A, B	Avoid ambiguous behavior from the subsystem.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > “Check usage of Ports and Subsystems blocks”** • **By Task > Modeling Standards for ISO 26262 > “Check usage of Ports and Subsystems blocks”** • **By Task > Modeling Standards for EN 50128 > “Check usage of Ports and Subsystems blocks”** • **By Task > Modeling Standards for DO-178C/DO-331> “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 26262-6, Table 1 (b) ’Use of language subsets’ ISO 26262-6, Table 1 (d) ’Use of defensive implementation techniques’ • EN 50128, Table A.4 (11) ’Language Subset’ EN 50128, Table A.3 (1) ’Defensive Programming’ • DO-331, Section MB.6.3.1.e ’High-level requirements conform to standards’ DO-331, Section MB.6.3.2.e ’Low-level requirements conform to standards’ • MISRA-C:2004, Rule 21.1	

ID: Title	hisl_0007: Usage of While Iterator subsystems
Last Changed	R2013b
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 result of having a 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 bounded iterative behavior in the generated code when using the For Iterator block, do one of the following:	
	A	In the For Iterator block parameters dialog box, set **Iteration limit source** to `internal`.
	B	If **Iteration limit source** must be `external`, use a block that has a constant value, such as a Width, Probe, or Constant.
	C	In the For Iterator block parameters dialog box, clear **Set next i (iteration variable) externally**.
	D	In the For Iterator block parameters dialog box, consider selecting **Show iteration variable** to observe the iteration value during simulation.
Notes	When you use the For Iterator block, feed the loop control variable with fixed (nonvariable) values to get a predictable number of loop iterations. Otherwise, a loop can result in unpredictable execution times and, in the case of external iteration variables, infinite loops that can lead to execution-time overruns.	
Rationale	A, B, C, D	Support bounded iterative behavior in generated code.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > "Check usage of Ports and Subsystems blocks"** • **By Task > Modeling Standards for ISO 26262 > "Check usage of Ports and Subsystems blocks"** • **By Task > Modeling Standards for EN 50128 > "Check usage of Ports and Subsystems blocks"** • **By Task > Modeling Standards for DO-178C/DO-331 > "Check usage of Ports and Subsystems blocks"**	

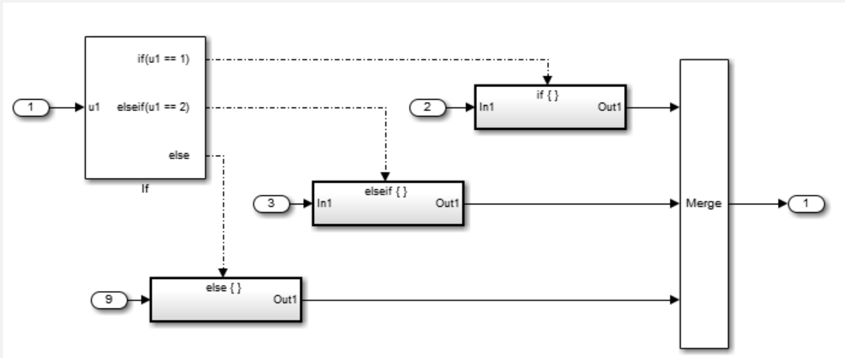
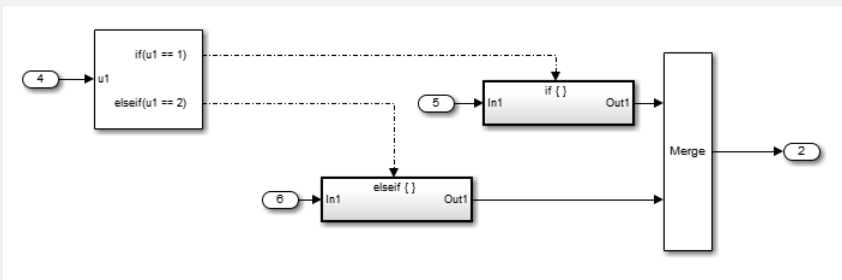
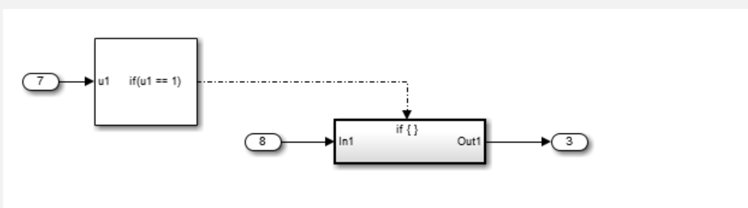
ID: Title	hisl_0008: Usage of For Iterator blocks
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, MB.Section 6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' • MISRA-C:2004, Rule 13.6
Last Changed	R2013b

hisl_0009: Usage of For Iterator Subsystem blocks

ID: Title	hisl_0009: Usage of For Iterator Subsystem blocks	
Description	To support unambiguous behavior, when using the For Iterator Subsystem block,	
	A	Specify inherited (`-1`) or constant (`inf`) sample times for blocks within the subsystem.
	B	Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystem.
Rationale	A, B	Avoid ambiguous behavior from the subsystem.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > “Check usage of Ports and Subsystems blocks”** • **By Task > Modeling Standards for ISO 26262 > “Check usage of Ports and Subsystems blocks”** • **By Task > Modeling Standards for EN 50128 > “Check usage of Ports and Subsystems blocks”** • **By Task > Modeling Standards for DO-178C/DO-331 > “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 26262-6, Table 1 (b) ’Use of language subsets’ ISO 26262-6, Table 1 (d) ’Use of defensive implementation techniques’ • EN 50128, Table A.4 (11) ’Language Subset’ EN 50128, Table A.3 (1) ’Defensive Programming’ • DO-331, Section MB.6.3.2.g ’Algorithms are accurate’ • MISRA-C:2004, Rule 13.6	
Last Changed	R2013b	
Examples	See “hisl_0007: Usage of While Iterator subsystems” on page 2-20.	

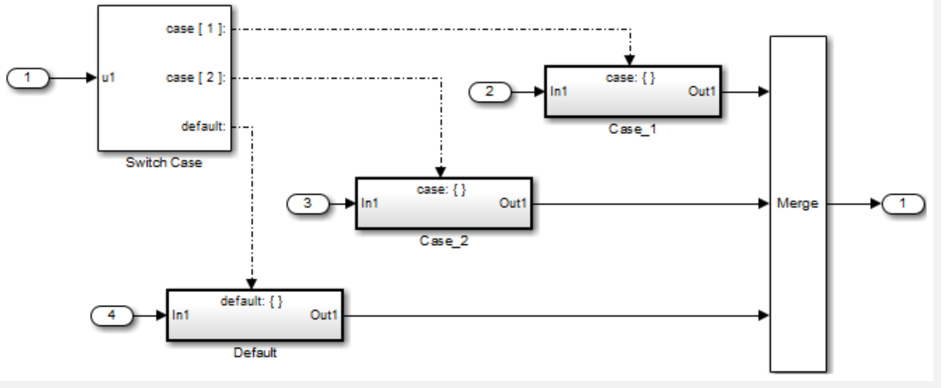
hisl_0010: Usage of If blocks and If Action Subsystem blocks

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 2-47	
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 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA-C:2004, Rule 14.10	
See Also	na_0012: Use of Switch vs. If-Then-Else Action Subsystem in the Simulink documentation	
Last Changed	R2013b	

ID: Title	hisl_0010: Usage of If blocks and If Action Subsystem blocks
Examples	 **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

<table>
<tr><th>ID: Title</th><th colspan="2">hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks</th></tr>
<tr><td rowspan="4">Description</td><td colspan="2">To support verifiable generated code, when using the Switch Case block:</td></tr>
<tr><td>A</td><td>In the Switch Case block parameter dialog box, select Show default case.</td></tr>
<tr><td>B</td><td>Connect the outports of the Switch Case block to a Switch Case Action Subsystem block.</td></tr>
<tr><td>C</td><td>Use an integer data type for the inputs to Switch Case blocks.</td></tr>
<tr><td>Prerequisites</td><td colspan="2">“hisl_0016: Usage of blocks that compute relational operators” on page 2-47</td></tr>
<tr><td>Notes</td><td colspan="2">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.</td></tr>
<tr><td>Rationale</td><td>A, B, C</td><td>Support generation of verifiable code.</td></tr>
<tr><td>References</td><td colspan="2">IEC 61508-3, Table A.3 (3) ’Language subset’
IEC 61508-3, Table A.4 (3) ’Defensive programming’
ISO 26262–6, Table 1(b) ’Use of language subsets’
ISO 26262–6, Table 1(d) ’Use of defensive implementation techniques’
EN 50128, Table A.4 (11) ’Language Subset’
EN 50128, Table A.3 (1) ’Defensive Programming’
MISRA-C:2004, Rule 14.10</td></tr>
<tr><td>See Also</td><td colspan="2">db_0115: Simulink patterns for case constructs in the Simulink documentation.</td></tr>
</table>

ID: Title	hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks
Last Changed	R2013b
Examples	The following graphic displays an example of providing a default path of execution using a “Default” block. 

hisl_0012: Usage of conditionally executed subsystems

<table>
<tr><th>ID: Title</th><th colspan="2">hisl_0012: Usage of conditionally executed subsystems</th></tr>
<tr><td rowspan="3">Description</td><td colspan="2">To support unambiguous behavior, when using conditionally executed subsystems:</td></tr>
<tr><td>A</td><td>Specify inherited (<code>-1</code>) sample times for all blocks in the subsystem, except Constant. Constant blocks can use infinite (<code>inf</code>) sample time.</td></tr>
<tr><td>B</td><td>If the subsystem is called asynchronously, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystem.</td></tr>
<tr><td>Notes</td><td colspan="2">Conditionally executed subsystems include:
• If Action
• Switch Case Action
• Function-Call
• Triggered
• Enabled

Sample time-dependent blocks include:
• Discrete State-Space
• Discrete-Time Integrator
• Discrete FIR Filter
• Discrete Filter
• Discrete Transfer Fcn
• Discrete Zero-Pole
• Transfer Fcn First Order
• Transfer Fnc Real Zero
• Transfer Fcn Lead or Lag</td></tr>
<tr><td>Rationale</td><td>A, B</td><td>Support unambiguous behavior.</td></tr>
</table>

ID: Title	hisl_0012: Usage of conditionally executed subsystems
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming'
Last Changed	R2013b
Examples	When using discrete blocks, the behavior depends on the operation across multiple contiguous time steps. When the blocks are called intermittently, the results may not conform to your expectations.

hisl_0024: Inport interface definition

ID: Title	hisl_0024: Inport interface definition
Description	To support strong data typing and unambiguous behavior of the model and the generated code, for each root-level Inport block, explicitly set the following block parameters: • **Data type** • **Port dimensions (-1 for inherited)** • **Sample time (-1 for inherited)**
Note	Using root-level Inport blocks without fully defined dimensions, sample times, or data type can lead to ambiguous simulation results. If you do not explicitly define these parameters, Simulink back-propagates dimensions, sample times, and data types from downstream blocks.
Rationale	• Avoid unambiguous behavior. • Support full specification of software interface.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > "Check for root Inports with missing properties"** • **By Task > Modeling Standards for ISO 26262 > "Check for root Inports with missing properties"** • **By Task > Modeling Standards for EN 50128 > "Check for root Inports with missing properties"**
References	• IEC 61508-3, Table B.9 (5) 'Fully defined interface' • ISO 26262-4, Table 2 (2) 'Precisely defined interfaces' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' • EN 50128, Table A.3 (19) 'Fully Defined Interface'
Last Changed	R2013b

hisl_0025: Design min/max specification of input interfaces

ID: Title	hisl_0025: Design min/max specification of input interfaces
Description	Provide design min/max information for root-level Inport blocks to specify the input interface ranges.
Notes	• Specifying the range of Inport blocks on the root level enables additional capabilities[1]. Examples include: - Detection of overflows through simulation range checking. - Code optimizations using Embedded Coder. - Design model verification using Simulink Design Verifier™. - Fixed-point autoscaling using Fixed-Point Designer™. • Specified design ranges can be used by Embedded Coder to optimize the generated code. If you want to use design ranges for optimization, in the Configuration Parameters dialog box, on the **Code Generation** pane, consider selecting **Optimize using the specified minimum and maximum values**. • Ranges for bus-type Inport blocks are specified with the bus elements of the defining bus object. Simulink ignores range specifications provided directly at Inport blocks that are bus-type.
Rationale	Support precise specification of the input interface.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > "Check for root Inports with missing range definitions"** • **By Task > Modeling Standards for ISO 26262 > "Check for root Inports with missing range definitions"** • **By Task > Modeling Standards for EN 50128 > "Check for root Inports with missing range definitions"**

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

ID: Title	hisl_0025: Design min/max specification of input interfaces
References	• IEC 61508-3, Table B.9 (5) 'Fully defined interface' • ISO 26262-4, Table 2 (2) 'Precisely defined interfaces' • EN 50128, Table A.1(11) – Software Interface Specifications, Table A.3 (19) 'Fully Defined Interface'
Last Changed	R2013b

hisl_0026: Design min/max specification of output interfaces

ID: Title	**hisl_0026: Design min/max specification of output interfaces**
Description	Provide design min/max information for root-level Outport blocks to specify the output interface ranges.

Notes	• Specifying the range of Outport blocks on the root level enables additional capabilities[2]. Examples include:
	  - Detection of overflows through simulation range checking.
	  - Code optimizations using Embedded Coder.
	  - Design model verification using Simulink Design Verifier.
	  - Fixed-point autoscaling using Fixed-Point Designer.
	• Specified design ranges can be used by Embedded Coder to optimize the generated code. If you want to use design ranges for optimization, in the Configuration Parameters dialog box, on the **Code Generation** pane, consider selecting **Optimize using the specified minimum and maximum values**.
	• Ranges for bus-type Outport blocks are specified with the bus elements of the defining bus object. Simulink ignores range specifications provided directly at Outport blocks that are bus-type.

Rationale	Support precise specification of the output interface.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > "Check for root Outports with missing range definitions"** • **By Task > Modeling Standards for ISO 26262 > "Check for root Outports with missing range definitions"** • **By Task > Modeling Standards for EN 50128 > "Check for root Outports with missing range definitions"**

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

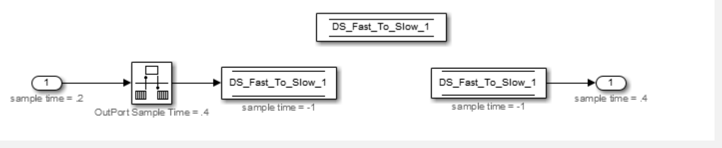
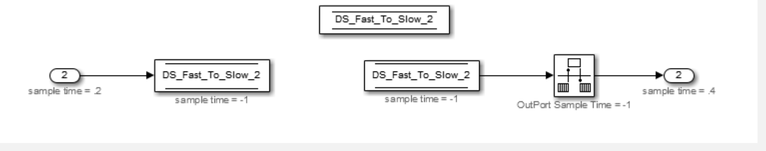
ID: Title	hisl_0026: Design min/max specification of output interfaces
References	• IEC 61508-3, Table B.9 (5) ‘Fully defined interface’ • ISO 26262-4, Table 2 (2) ‘Precisely defined interfaces‘ • EN 50128, Table A.1(11) – Software Interface Specifications, Table A.3 (19) ‘Fully Defined Interface‘
Last Changed	R2013b

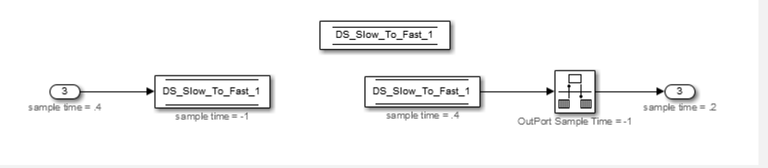
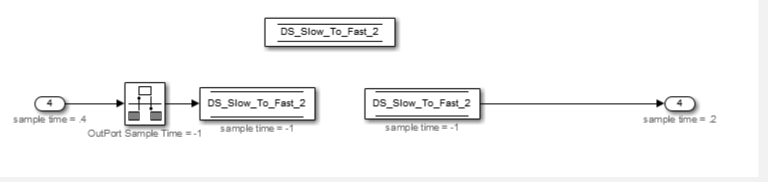
Signal Routing

In this section...

hisl_0013: Usage of data store blocks

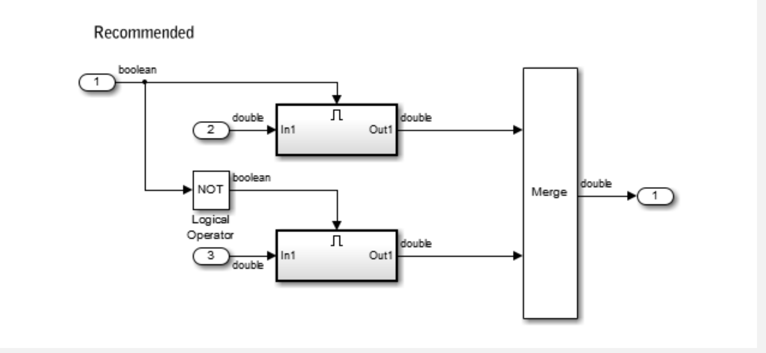
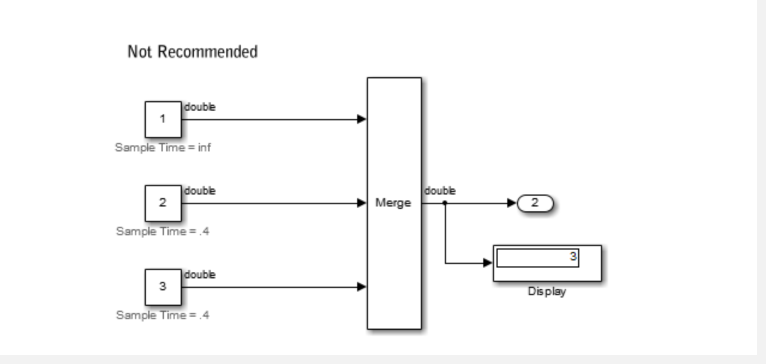
ID: Title	hisl_0013: Usage of data store blocks	
Description	To support deterministic behavior across different sample times or models when using data store blocks, including Data Store Memory, Data Store Read, and Data Store Write:	
	A	In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, under **Data Store Memory Block**, set the following parameters to `error`: • **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 impact on your software verification effort. Models and subsystems that use only inports and outports to pass data provide a directly traceable interface, simplifying the verification process.	
Rationale	A, B, C	Support consistent data values across different sample times or models.
Model Advisor Checks	**By Task > Modeling Standards for DO-178C/DO-331 > "Check safety-related diagnostic settings for data store memory"**	

ID: Title	hisl_0013: Usage of data store blocks
References	• IEC 61508-3, Table A.3 (3) ’Language subset’ IEC 61508-3, Table A.4 (3) ’Defensive programming’ • ISO 26262-6, Table 1 (b) ’Use of language subsets’ ISO 26262-6, Table 1 (d) ’Use of defensive implementation techniques’ • EN 50128, Table A.4 (11) ’Language Subset’ EN 50128, Table A.3 (1) ’Defensive Programming’ • DO-331, Section MB.6.3.3.b ’Software architecture is consistent’
Last Changed	R2013b
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 the 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 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 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.3.b 'Software architecture is consistent'	
Last Changed	R2013b	

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

hisl_0021: Consistent vector indexing method

ID: Title	hisl_0021: Consistent vector indexing method	
Description	Within a model, use:	
	A	A consistent vector indexing method for allblocks. 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.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > “Check for inconsistent vector indexing methods”** • **By Task > Modeling Standards for ISO 26262 > “Check for inconsistent vector indexing methods”** • **By Task > Modeling Standards for EN 50128 > “Check for inconsistent vector indexing methods”** • **By Task > Modeling Standards for DO-178C/DO-331 > “Check for inconsistent vector indexing methods”**	
References	• IEC 61508–3, Table A.3 (3) ’Language subset’ IEC 61508–3, Table A.4 (5) ’Design and coding standards’ • ISO 26262-6, Table 1 (b) ’Use of language subsets’ ISO 26262-6, Table 1 (f) ’Use of unambiguous graphical representation’ • EN 50128, Table A.4 (11) ’Language Subset’ EN 50128, Table A.12 (1) ’Coding Standard’ • DO-331, Section MB.6.3.2.b ’Low-level requirements are accurate and consistent’	
See Also	“cgsl_0101: Zero-based indexing”	
Last Changed	R2013b	

hisl_0022: Data type selection for index signals

ID: Title	hisl_0022: Data type selection for index signals	
Description	For index signals, use:	
	A	An integer or enumerated data type
	B	A data type that covers the range of indexed values.
	Blocks that use a signal index include: • Assignment • Direct Lookup Table (n-D) • Index Vector • Interpolation Using Prelookup • MATLAB® Function • Multiport Switch • n-D Lookup Table (internal type index selection) • Selector • Stateflow® Chart	
Rationale	A	Prevent unexpected results that can occur with rounding operations for floating-point data types.
	B	Enable access to 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 26262-6, Table 1 (b) ’Use of language subsets’ ISO 26262-6, Table 1 (c) ’Enforcement of strong typing’ ISO 26262-6, Table 1 (d) ’Use of defensive implementation techniques’ • EN 50128, Table A.4 (8) ’Strongly Typed Programming Language’ EN 50128, Table A.3 (1) ’Defensive Programming’ • DO-331, Section MB.6.3.4.f ’Accuracy and Consistency of Source Code’	
Last Changed	R2013b	

hisl_0023: Verification of model and subsystem variants

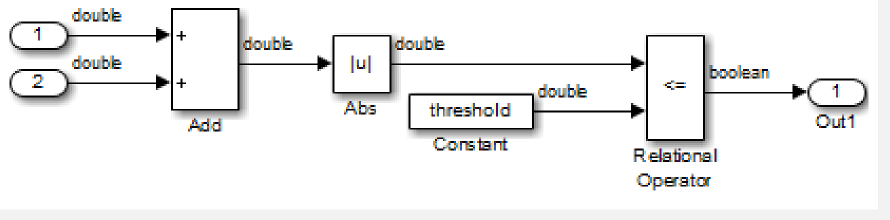
ID: Title	**hisl_0023: Verification of model and subsystem variants**	
Description	When verifying that a model is consistent with generated code, do one of the following:	
	A	In the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, disable variants in generated code by setting **Generate preprocessor conditionals** to `Disable all`.
	B	Verify all combinations of model variants that might be active in the generated code.
Rationale	A	Simplify consistency testing between the model and generated code by restricting the code base to a single variant.
	B	Make sure that consistency testing between the model and generated code is complete for all variants.
References	• DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508–3, Table A.4 (7) 'Use of trusted / verified software modules and components'	
Last Changed	R2012b	

Logic and Bit Operations

In this section...

hisl_0016: Usage of blocks that compute relational operators

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 (~=). 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 EN 50128 > "Check usage of Logic and Bit Operations blocks"** • **By Task > Modeling Standards for DO-178C/DO-331 > "Check usage of Logic and Bit Operations blocks"**	

ID: Title	hisl_0016: Usage of blocks that compute relational operators
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 13.3
See Also	"hisl_0017: Usage of blocks that compute relational operators (2)" on page 2-49
Last Changed	R2013b
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 EN 50128 > "Check usage of Logic and Bit Operations blocks"** • **By Task > Modeling Standards for DO-178C/DO-331 > "Check usage of Logic and Bit Operations blocks"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset'; IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 12.6	
See Also	"hisl_0016: Usage of blocks that compute relational operators" on page 2-47	
Last Changed	R2013b	

hisl_0018: Usage of Logical Operator block

ID: Title	hisl_0018: Usage of Logical Operator block	
Description	To support unambiguous behavior of generated code, when using the Logical Operator block,	
	A	Set the **Output data type** block parameter to `Boolean`.
Prerequisites	"hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)" on page 5-25	
Rationale	A	Avoid ambiguous behavior of generated code.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > "Check usage of Logic and Bit Operations blocks"** • **By Task > Modeling Standards for ISO 26262 > "Check usage of Logic and Bit Operations blocks"** **By Task > Modeling Standards for EN 50128 > "Check usage of Logic and Bit Operations blocks"** • **By Task > Modeling Standards for DO-178C/DO-331 > "Check usage of Logic and Bit Operations blocks"** • **By Task > Modeling Standards for DO-178C/DO-331 > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rule 12.6	
Last Changed	R2013b	

hisl_0019: Usage of Bitwise Operator block

ID: Title	hisl_0019: Usage of Bitwise Operator block	
Description	To support unambiguous behavior, when using the Bitwise Operator block,	
	A	Avoid signed integer data types as input to the block.
	B	Choose an output data type that represents zero exactly.
Notes	Bitwise operations on signed integers are not meaningful. If a shift operation moves a signed bit into a numeric bit, or a numeric bit into a signed bit, unpredictable and unwanted behavior can result.	
Rationale	A, B	Support unambiguous behavior of generated code.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • MISRA-C:2004, Rule 12.7	
See Also	"hisf_0003: Usage of bitwise operations" on page 3-12in the Simulink documentation	
Last Changed	R2013b	

3

Stateflow Chart Considerations

Chart Properties

In this section...

hisf_0001: Mealy and Moore semantics

ID: Title		hisf_0001: Mealy and Moore semantics
Description	To create Stateflow charts that implement a subset of Stateflow semantics,	
	A	In the Chart properties dialog box, set **State Machine Type** to `Mealy` or `Moore`.
	B	Apply consistent settings to the Stateflow charts in a model.
Note	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-178C/DO-331 > “Check state machine type of Stateflow charts”** • **By Task > Modeling Standards for IEC 61508 > “Check state machine type of Stateflow charts”** • **By Task > Modeling Standards for ISO 26262 > “Check state machine type of Stateflow charts”** • **By Task > Modeling Standards for EN 50128 > “Check state machine type of Stateflow charts”**	

ID: Title	hisf_0001: Mealy and Moore semantics
References	• IEC 61508-3,Table A.7 (2) 'Simulation/modeling' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.11 (3) 'Simulation' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' DO-331, Section MB.6.3.3.e 'Software architecture conform to standards'
See Also	"Create Mealy and Moore Charts" in the Stateflow documentation
Last Changed	R2013b

hisf_0002: User-specified state/transition execution order

ID: Title	**hisf_0002: User-specified state/transition execution order**	
Description	Do the following to explicitly set the execution order for active states and valid transitions in Stateflow charts:	
	A	In the Chart Properties dialog box, select **User specified state/transition execution order**.
	B	In the Stateflow Editor **View** menu, select **Show Transition Execution Order**.
	C	Set default transition to evaluate last.
Note	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-178C/DO-331 > "Check Stateflow charts for ordering of states and transitions"** • **By Task > Modeling Standards for IEC 61508 > "Check usage of Stateflow constructs"** • **By Task > Modeling Standards for ISO 26262 > "Check usage of Stateflow constructs"** • **By Task > Modeling Standards for EN 50128 > "Check usage of Stateflow constructs"**	

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 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (f) 'Use of unambiguous graphical representation' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.3.b 'Software architecture is consistent' DO-331, Section MB.6.3.3.e 'Software architecture conform to standards '
See Also	The following topics in the Stateflow documentation • "Transition Testing Order in Multilevel State Hierarchy" • "Execution Order for Parallel States"
Last Changed	R2013b

hisf_0009: Strong data typing (Simulink and Stateflow boundary)

ID: Title	hisf_0009: Strong data typing (Simulink and Stateflow boundary)	
Description	To support strong data typing between Simulink and Stateflow ,	
	A	Select **Use Strong Data Typing with Simulink I/O**.
Notes	By default, input to and output from Stateflow charts are of type `double`. To interface directly with Simulink signals of data types other than `double`, select **Use Strong Data Typing with Simulink I/O**. In this mode, data types between the Simulink and Stateflow boundary are strongly typed, and the Simulink software does not treat the data types as `double`. The Stateflow chart accepts input signals of any data type supported by the Simulink software, provided that the type of the input signal matches the type of the corresponding Stateflow input data object. Otherwise, the software reports a type mismatch error.	
Rationale	A	Support strongly typed code.
Model Advisor Checks	• **By Task > Modeling Standards for IEC 61508 > "Check usage of Stateflow constructs"** • **By Task > Modeling Standards for ISO 26262 > "Check usage of Stateflow constructs"** • **By Task > Modeling Standards for EN 50128 > "Check usage of Stateflow constructs"**	
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to	

ID: Title	hisf_0009: Strong data typing (Simulink and Stateflow boundary)
	standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C:2004, Rules 10.1, 10.2, 10.3 and 10.4
Last Changed	R2013b

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	Run-time diagnostics are only triggered during simulation. If the error condition is not reached during simulation, the error message is not triggered for code generation.	
Rationale	A, B	Protect against unreachable code and unpredictable execution time.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178C/DO-331 > “Check Stateflow debugging settings”** • **By Task > Modeling Standards for IEC 61508 > “Check usage of Stateflow constructs”** • **By Task > Modeling Standards for ISO 26262 > “Check usage of Stateflow constructs”** • **By Task > Modeling Standards for EN 50128 > “Check usage of Stateflow constructs”**	

ID: Title	hisf_0011: Stateflow debugging settings
References	• IEC 61508-3, Table A.7 (2) 'Simulation/modeling' • ISO 26262 Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.11 (3) 'Simulation' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'
Last Changed	R2013b

Chart Architecture

In this section...

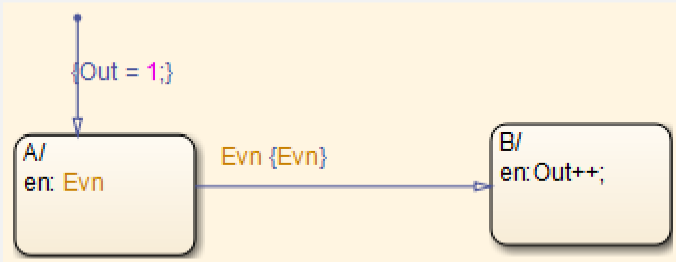
hisf_0003: Usage of bitwise operations

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 26262-6, Table 1 (b) ’Use of language subsets’ ISO 26262-6, Table 1 (c) ’Enforcement of strong typing’ • EN 50128, Table A.4 (11) ’Language Subset’ EN 50128, Table A.3 (1) ’Defensive Programming’ • DO-331, Section MB.6.3.1.b ’High-level requirements are accurate and consistent’ DO-331, Section MB.6.3.1.e ’High-level requirements conform to standards’ DO-331, Section 6.3.1.g ’Algorithms are accurate’ DO-331, Section MB.6.3.2.b ’Low-level requirements are accurate and consistent’ DO-331, Section MB.6.3.2.e ’Low-level requirements conform to standards’ DO-331, Section MB.6.3.2.g ’Algorithms are accurate’ • MISRA-C:2004, Rule 12.7 ’Bitwise operators shall not be applied to operands whose underlying type is signed’	
See Also	“hisl_0019: Usage of Bitwise Operator block”	
Last Changed	R2013b	

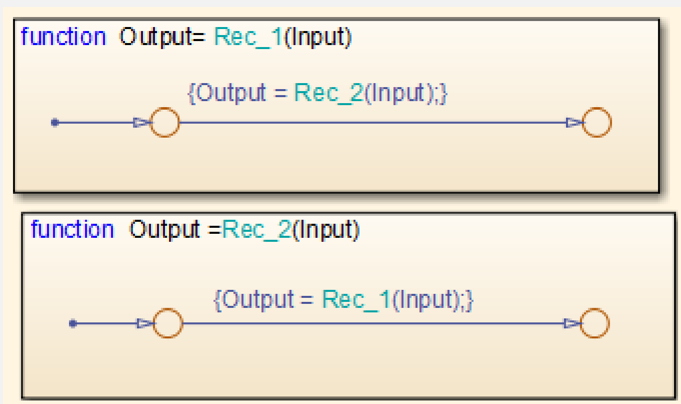
hisf_0004: Usage of recursive behavior

<table>
<tr><th>ID: Title</th><th colspan="2">hisf_0004: Usage of recursive behavior</th></tr>
<tr><td rowspan="3">Description</td><td colspan="2">To support bounded function call behavior, avoid using design patterns that include unbounded recursive behavior. Recursive behavior is bound if you do the following:</td></tr>
<tr><td>A</td><td>Use an explicit termination condition that is local to the recursive call.</td></tr>
<tr><td>B</td><td>Make sure the termination condition is reached.</td></tr>
<tr><td>Notes</td><td colspan="2">This rule only applies if a chart is a classic Stateflow chart. If “hisf_0001: Mealy and Moore semantics” on page 3-3 is followed, recursive behavior is prevented due to restrictions in the chart semantics. Additionally, you can detect the error during simulation by enabling the Stateflow diagnostic Detect Cycles.</td></tr>
<tr><td>Rationale</td><td>A, B</td><td>Promote bounded function call behavior.</td></tr>
<tr><td>References</td><td colspan="2">• IEC 61508-3, Table B.1 (6) ’Limited use of recursion’
• ISO 26262-6, Table 9 (j) ’No recursions’
• EN 50128, Table A.12 (6) ’Limited Use of Recursion’
• DO-331, Section MB.6.3.1.b ’High-level requirements are accurate and consistent’
DO-331, Section MB.6.3.1.e ’High-level requirements conform to standards’
DO-331, Section MB.6.3.1.g ’Algorithms are accurate’
DO-331, Section MB.6.3.2.b ’Low-level requirements are accurate and consistent’
DO-331, Section MB.6.3.2.e ’Low-level requirements conform to standards’
DO-331, Section MB.6.3.2.g ’Algorithms are accurate’
• MISRA-C:2004, Rule 16.2</td></tr>
<tr><td>Last Changed</td><td colspan="2">R2013b</td></tr>
</table>

<table>
<tr><th>ID: Title</th><th>hisf_0004: Usage of recursive behavior</th></tr>
<tr><td>Examples</td><td>There are multiple patterns in Stateflow that can result in unbounded recursion.



Recursive Function Calls</td></tr>
<tr><td></td><td>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.

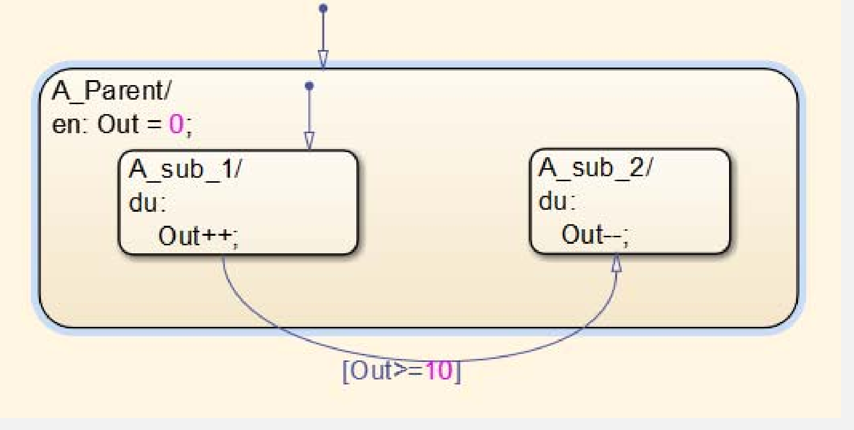


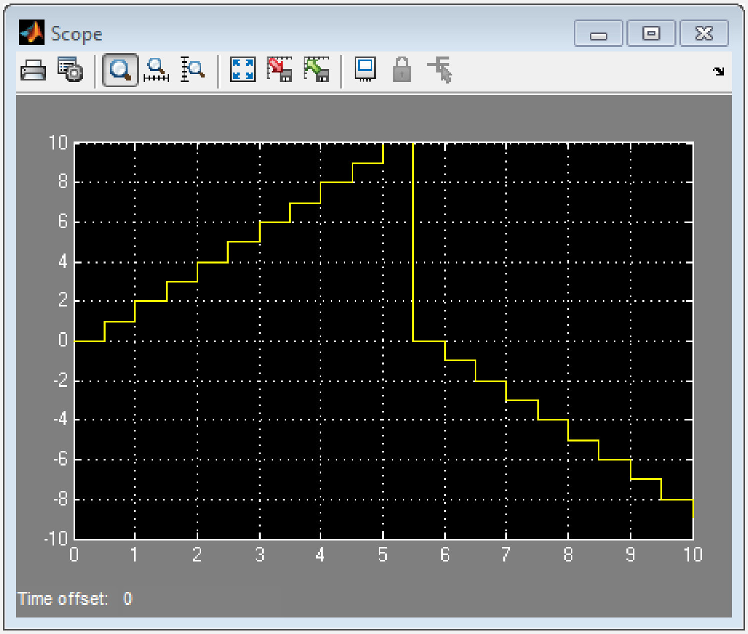
Recursive Function Calls</td></tr>
</table>

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-331, Section MB.6.3.1.b ’High-level requirements are accurate and consistent’ DO-331, Section MB.6.3.1.d ’High-level requirements are verifiable’ DO-331, Section MB.6.3.1.e ’High-level requirements conform to standards’ DO-331, Section MB.6.3.2.b ’Low-level requirements are accurate and consistent’ DO-331, Section MB.6.3.2.d ’Low-level requirements are verifiable’ DO-331, Section MB.6.3.2.e ’Low-level requirements conform to standards’	
Last Changed	R2012b	

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

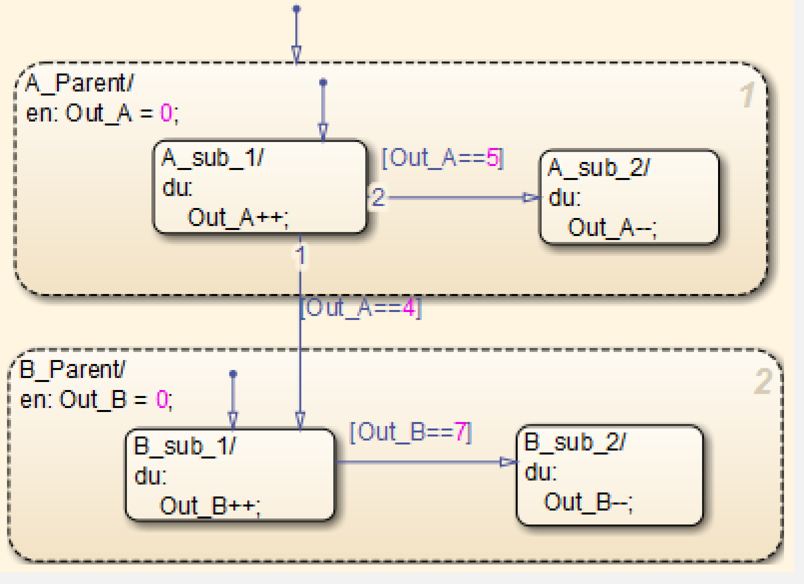
ID: Title	hisf_0010: Usage of transition paths (looping out of parent of source and destination objects	
Description	Transitions that loop out of the parent of the source and destination objects are typically unintentional and cause the parent to deactivate.	
	A	Avoid using these transitions.
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.	
Rationale	A	Promote a clear modeling style.
References	• DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate'	
Last Changed	R2012b	
Examples		

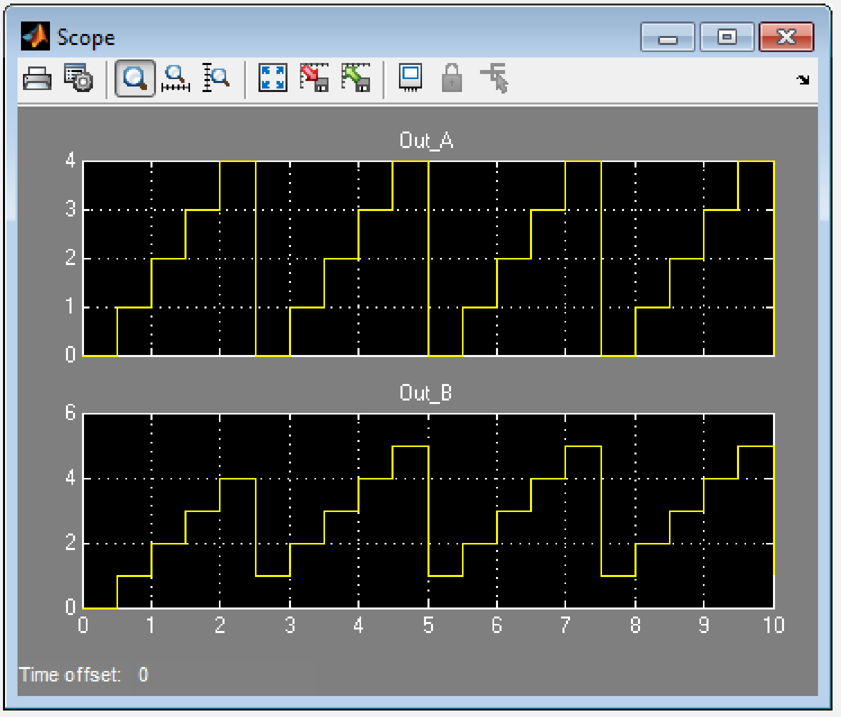
ID: Title	hisf_0010: Usage of transition paths (looping out of parent of source and destination objects
	

hisf_0012: Chart comments

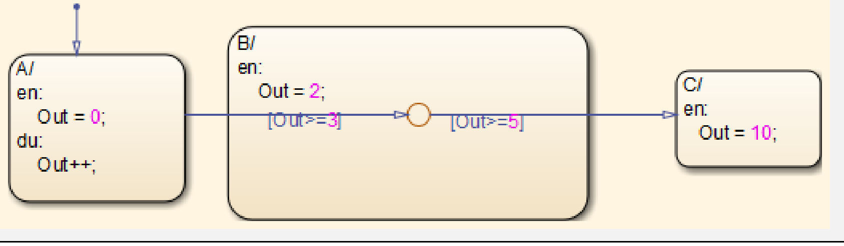
ID: Title	hisf_0012: Chart comments	
Description	To enhance traceability between generated code and a model,	
	A	Add comments to the following Stateflow objects: • Transitions
Rationale	A	Enhance traceability between generated code and the corresponding model.
References	• DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements'	
Last Changed	R2012b	

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

ID: Title	hisf_0013: Usage of transition paths (crossing parallel state boundaries)	
Description	To avoid creating diagrams that are hard to understand,	
	A	Avoid creating transitions that cross from one parallel state to another.
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.	
Rationale	A	Enhance model readability.
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. 	

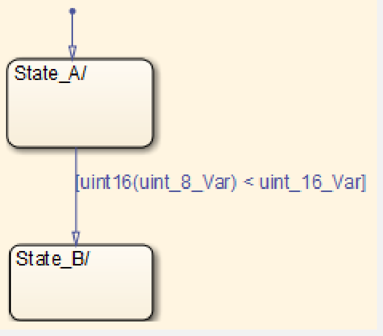
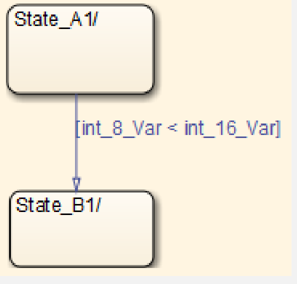
ID: Title	hisf_0013: Usage of transition paths (crossing parallel state boundaries)
	

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 without benefit,	
	A	Avoid transition paths that go into and out of a state without ending on a substate.
Notes	You can use this guideline to maintain a modeling language subset in high-integrity projects.	
Rationale	A	Enhance model readability.
References	• DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards'	
Last Changed	R2012b	
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-331, Section MB.6.3.1.b ’High-level requirements are accurate and consistent’ DO-331, Section MB.6.3.1.e ’High-level requirements conform to standards’ DO-331, Section MB.6.3.1.g ’Algorithms are accurate’ DO-331, Section MB.6.3.2.b ’Low-level requirements are accurate and consistent’ DO-331, Section MB.6.3.2.e ’Low-level requirements conform to standards’ DO-331, Section MB.6.3.2.g ’Algorithms are accurate’	

ID: Title	hisf_0015: Strong data typing (casting variables and parameters in expressions)
Last Changed	R2012b
Examples	 **Recommended**  **Not Recommended**

4

MATLAB Function Block Considerations

Modeling Style

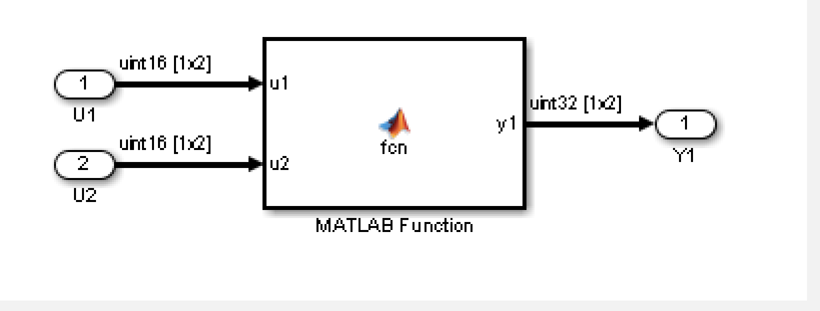
In this section...

himl_0001: Usage of standardized function headers

ID: Title	himl_0001: Usage of standardized function headers
Description	When using MATLAB Function blocks, use a standardized header to provide information about the purpose and use of the function.
Note	This guideline applies to MATLAB functions within a MATLAB function block and externally called MATLAB functions.
Rationale	A standardized header improves the readability and documentation of MATLAB functions. The header should provide a function description and usage information.
See Also	• MathWorks Automotive Advisory Board (MAAB) guideline na_0025: MATLAB Function Header • Orion GN&C: MATLAB and Simulink Standards, jh_0073: eML Header • “MATLAB Function Block Editor”
Last Changed	R2013a
Examples	A typical standardized function header includes: • Function name • Description • Inputs and outputs (if possible, include size and type) • Assumptions and limitations • Revision history

himl_0002: Strong data typing (MATLAB Function block boundary)

ID: Title	himl_0002: Strong data typing (MATLAB Function block boundary)
Description	To support strong data typing at the interfaces of MATLAB Function blocks, explicitly define the interface for input signals, output signals, and parameters, by setting: • Complexity • Type
Rationale	Defined interfaces: • Allow consistency checking of interfaces. • Prevent unintended generation of different functions for different input and output types. • Simplify testing of functions by limiting the number of test cases.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178C/DO-331 > "Check for MATLAB Function block interfaces with inherited properties"** • **By Task > Modeling Standards for ISO 26262 > "Check for MATLAB Function block interfaces with inherited properties"** • **By Task > Modeling Standards for EN 50128 > "Check for MATLAB Function block interfaces with inherited properties"** • **By Task > Modeling Standards for IEC 61508> "Check for MATLAB Function block interfaces with inherited properties"**
References	• IEC 61508-3, Table B.9 (5) - Fully defined interface • ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation • EN 50128, Table A.1 (11) - Software Interface Specifications • DO-331, Section MB.6.3.2.b - Low-level requirements are accurateand consistent

ID: Title	himl_0002: Strong data typing (MATLAB Function block boundary)
See Also	• MathWorks Automotive Advisory Board (MAAB) guideline na_0034: MATLAB Function block input/output settings • Orion GN&C: MATLAB and Simulink Standards, jh_0063: eML block input / output settings • "MATLAB Function Block Editor"
Last Changed	R2013b
Examples	**Recommended:** In the "Ports and Data Manager", specify the complexity and type of input `u1` as follows: • **Complexity** to `Off` • **Type** to `uint16`  **Not Recommended:** In the "Ports and Data Manager", do *not* specify the complexity and type of input `u1` as follows: • **Complexity** to `Inherited` • **Type** to `Inherit: Same as Simulink`. **Note** To access the "Ports and Data Manager", from the toolbar of the "MATLAB Function Block Editor", select **Edit Data**.

himl_0003: Limitation of MATLAB Function complexity

ID: Title	himl_0003: Limitation of MATLAB Function complexity
Description	When using MATLAB Function blocks, limit the size and complexity of MATLAB code. The size and complexity of MATLAB functions is characterized by: • Lines of code • Nested function levels • Cyclomatic complexity • Density of comments (ratio of comment lines to lines of code)
Note	Size and complexity limits can vary across projects. Typical limits might be as described in this table:
Rationale	• Readability • Comprehension • Traceability • Maintainability • Testability

Metric	Limit
Lines of code	60 per MATLAB function
Nested function levels	3[1,2]
Cyclomatic complexity	15
Density of comments	0.2 comment lines per line of code

[1]Pure Wrappers to external functions are not counted as separate levels.

[2]Standard MATLAB library functions do not count as separate levels.

ID: Title	himl_0003: Limitation of MATLAB Function complexity
Model Advisor Checks	• **By Task > Modeling Standards for DO-178C/DO-331 > "Check MATLAB Function block metrics"** • **By Task > Modeling Standards for ISO 26262 > "Check MATLAB Function block metrics"** • **By Task > Modeling Standards for EN 50128 > "Check MATLAB Function block metrics"** • **By Task > Modeling Standards for IEC 61508> "Check MATLAB Function block metrics"**
References	• IEC 61508-3, Table B.9 (5) - Fully defined interface • ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation • EN 50128, Table A.1(11) - Software Interface Specifications • DO-331, Sections MB.6.3.1.e - High-level requirements conform to standards DO-331, Sections MB.6.3.2.e - Low-level requirements conform to standards
See Also	• MathWorks Automotive Advisory Board (MAAB) guideline na_0016: Source lines of MATLAB Functions • MathWorks Automotive Advisory Board (MAAB) guideline na_0017: Number of called function levels • MathWorks Automotive Advisory Board (MAAB) guideline na_0018: Number of nested if/else and case statement • Orion GN&C: MATLAB and Simulink Standards, jh_0084: eML Comments • "MATLAB Function Block Editor"
Last Changed	R2013a

5

Configuration Parameter Considerations

Solver

In this section...

hisl_0040: Configuration Parameters > Solver > Simulation time

ID: Title	hisl_0040: Configuration Parameters > Solver > Simulation time	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Solver** pane, set parameters for simulation time as follows:	
	A	**Start time** to `0.0`.
	B	**Stop time** to a positive value that is less than the value of **Application lifespan (days)**.
Note	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. 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 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'	
See Also	• "hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)" on page 5-27 • Solver Pane section of the Simulink documentation	
Last Changed	R2013b	

hisl_0041: Configuration Parameters > Solver > Solver options

ID: Title	hisl_0041: Configuration Parameters > Solver > Solver options	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Solver** pane, set parameters for solvers as follows:	
	A	**Type** to `Fixed-step`.
	B	**Solver** to `discrete (no continuous states)`.
Note	Generating code for production requires a fixed-step, discrete solver.	
Rationale	A, B	Generate code that is valid for production code generation.
References	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'	
See Also	"Solver Pane" in the Simulink documentation	
Last Changed	R2013b	

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

ID: Title	hisl_0042: Configuration Parameters > Solver > Tasking and sample time options	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Solver** pane, set parameters for tasking and sample time as follows:	
	A	**Periodic sample time constraint** to `Specified` and assign 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	**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.

ID: Title	hisl_0042: Configuration Parameters > Solver > Tasking and sample time options
References	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements'
See Also	"Solver Pane" in the Simulink documentation
Last Changed	R2013b

Diagnostics

In this section...

hisl_0043: Configuration Parameters > Diagnostics > Solver

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

If Diagnostic Parameter...	Is Not Set As Indicated, Then ...
Algebraic loop	Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.
Minimize algebraic loop	Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.
Block priority violation	Block execution order can include undetected conflicts that might

<table>
<tr><th>ID: Title</th><th colspan="2">hisl_0043: Configuration Parameters > Diagnostics > Solver</th></tr>
<tr><td></td><td colspan="2">
<table>
<tr><td></td><td>result in unpredictable block order execution.</td></tr>
<tr><td>Unspecified inheritability of sample times</td><td>An S-function that is not explicitly set to inherit sample time can go undetected and result in unpredictable behavior.</td></tr>
<tr><td>Automatic solver parameter selection</td><td>An automatic change to the solver, step size, or simulation stop time can go undetected and might the operation of generated code.</td></tr>
<tr><td>State name clash</td><td>A name being used for more than one state might go undetected.</td></tr>
</table>
You can set the following solver diagnostic parameters to anyvalue:
Min step size violation
Sample hit time adjusting
Consecutive zero crossings violation
Solver data inconsistency
Extraneous discrete derivative signals
</td></tr>
<tr><td>Rationale</td><td>A</td><td>Support generation of robust and unambiguous code.</td></tr>
<tr><td>Model Advisor Checks</td><td colspan="2">By Task > Modeling Standards for DO-178C/DO-331 > “Check safety-related diagnostic settings for solvers”</td></tr>
<tr><td>References</td><td colspan="2">• IEC 61508-3, Table A.3 (3) ’Language subset’
• ISO 26262-6, Table 1 (b) ’Use of language subsets’
• EN 50128, Table A.4 (11) ’Language Subset’
• DO-331, MB.6.3.3.e ’Software architecture conforms to standards’</td></tr>
<tr><td>See Also</td><td colspan="2">• “Diagnostics Pane: Solver” in the Simulink documentation
• jc_0021: Model diagnostic settings in the Simulink documentation</td></tr>
<tr><td>Last Changed</td><td colspan="2">R2013b</td></tr>
</table>

hisl_0044: Configuration Parameters > Diagnostics > Sample Time

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** pane, set the parameters of the **Sample Time** section to `error`:	
	Compile-Time	• **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`.
	Run-Time	Not applicable
Note	Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.	

If Diagnostic Parameter...	Is Not Set As Indicated, Then ...
Source block specifies -1 sample time	Use of inherited sample times for a source block, such as Sine Wave, can go undetected and result in unpredictable execution rates for source and downstream blocks.
Discrete used as continuous	Input signals with continuous sample times for a discrete block, such as Unit Delay, can go undetected. You cannot use signals

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

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time
References	• IEC 61508-3, Table A.3 (3) 'Language subset' • ISO 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.3.b 'Software architecture is consistent'
See Also	"Diagnostics Pane: Sample Time" in the Simulink documentation
Last Changed	R2013b

hisl_0301: Configuration Parameters > Diagnostics > Compatibility

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

hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters

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

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

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

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

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

hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging

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

hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals

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

hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses

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

hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls

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

hisl_0309: Configuration Parameters > Diagnostics > Type Conversion

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

hisl_0310: Configuration Parameters > Diagnostics > Model Referencing

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

hisl_0311: Configuration Parameters > Diagnostics > Stateflow

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

Optimizations

In this section...

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

ID: Title	**hisl_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)**	
Description	To support unambiguous behavior when using logical operators, relational operators, and the Combinatorial Logic block,	
	A	Select **Implement logic signals as Boolean data (vs. double)** in the **Optimization** pane of the Configuration Parameters dialog box.
Notes	Selecting the **Implement logic signals as Boolean data (vs. double)** parameter, enables Boolean type checking, which produces an error when blocks that prefer Boolean inputs connect to double signals. This checking results in generating code that requires less memory.	
Rationale	A	Avoid ambiguous model behavior and optimize memory for generated code.
Model Advisor Checks	**By Task > Modeling Standards for DO-178C/DO-331 > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • ISO 26262-6, Table 1 (c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, MB.6.3.1.e 'High-level requirements conform to standards' DO-331, MB.6.3,2.e 'Low-level requirements conform to standards' • MISRA-C:2004, Rule 12.6	
Last Changed	R2013b	

hisl_0046: Configuration Parameters > Optimization > Block reduction

ID: Title	hisl_0046: Configuration Parameters > Optimization > Block reduction	
Description	To support unambiguous presentation of the generated code and support traceability between a model and generated code,	
	A	Clear the **Block reduction** parameter on the **Optimization** pane of the Configuration Parameters dialog box.
Notes	Selecting **Block reduction** might optimize blocks out of the code generated for a model. This results in requirements without associated code and violates traceability objectives.	
Rationale	A	Support unambiguous presentation of generated code.
	A	Support traceability between a model and generated code.
Model Advisor Checks	**By Task > Modeling Standards for DO-178C/DO-331 > "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-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements'	
See Also	"Block reduction" in the Simulink documentation	
Last Changed	R2012b	

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

ID: Title	hisl_0048: Configuration Parameters > Optimization > Application lifespan (days)	
Description	To support the robustness of systems that run continuously, in the Configuration Parameters dialog box, on the **Optimization** pane:	
	A	Set **Application lifespan (days)** to `inf`.
Notes	Embedded applications might run continuously. Do not assume a limited lifespan for timers and counters. . When you set **Application lifespan (days)** to `inf`, the simulation time is less than the application lifespan.	
Rationale	A	Support robustness of systems that run continuously.
Model Advisor Checks	**By Task > Modeling Standards for DO-178C/DO-331 > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate'	
See Also	• "Application lifespan (days)" in the Simulink documentation • "hisl_0040: Configuration Parameters > Solver > Simulation time" on page 5-3	
Last Changed	R2013b	

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

ID: Title	**hisl_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold**	
Description	To support unambiguous code, set the minimum signal or parameter width for generating a `for` loop. In the Configuration Parameters dialog box, on the **Optimization > Signals and Parameters** pane,	
	A	Set **Loop unrolling threshold** to `2` or greater.
	B	If **Pack Boolean data into bitfields** is selected, set **Bitfield declarator type specifier** to `uint_T`.
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 26262-6, Table 1 (b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • MISRA-C:2004, Rule 6.4	
See Also	"Loop unrolling threshold" in the Simulink documentation	
Last Changed	R2013b	

hisl_0052: Configuration Parameters > Optimization > Data initialization

ID: Title	**hisl_0052: Configuration Parameters > Optimization > Data initialization**	
Description	To support complete definition of data and initialize internal and external data to zero, in the Configuration Parameters dialog box, on the **Optimization** pane,	
	A	Clear **Remove root level I/O zero initialization**.
	B	Clear **Remove internal data zero initialization**.
Note	Explicitly initialize all variables. If the run-time environment of the target system provides mechanisms to initialize all I/O and state variables, consider using the initialization of the target as an alternative to the suggested settings.	
Rationale	A, B	Support fully defined data in generated code.
Model Advisor Checks	**By Task > Modeling Standards for DO-178C/DO-331 > “Check safety-related optimization settings”**	
References	• IEC 61508-3, Table A.4 (3) ’Defensive Programming’ • ISO 26262-6, Table 1 (d) ’Use of defensive implementation techniques’ • EN 50128, Table A.3 (1) ’Defensive Programming’ • MISRA-C:2004, Rule 9.1 • DO-331, Section MB.6.3.3.b ’Software architecture is consistent’	
See Also	Information about the following parameters in the Simulink documentation: • “Remove root level I/O zero initialization” • “Remove internal data zero initialization”	
Last Changed	R2013b	

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

ID: Title	**hisl_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values**	
Description	To support verifiable code, In the Configuration Parameters dialog box, on the **Optimization** pane,	
	A	Consider selecting **Remove code from floating-point to integer conversions that wraps out-of-range values**.
Notes	Avoid overflows as opposed to handling them with wrapper code. For blocks that have the parameter **Saturate on overflow** cleared, clearing **Remove code from floating-point to integer conversions that wraps out-of-range values** might add code that wraps out of range values, resulting in unreachable code that cannot be tested.	
Rationale	A	Support generation of code that can be verified.
Model Advisor Checks	**By Task > Modeling Standards for DO-178C/DO-331 > "Check safety-related optimization settings"**	
References	• IEC 61508-3, Table A.4 (3) 'Defensive Programming' • ISO 26262-6, Table 1 (d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA-C:2004, Rule 14.1 • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate'	
See Also	"Remove code from floating-point to integer conversions that wraps out-of-range values" in the Simulink documentation	
Last Changed	R2013b	

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

ID: Title	**hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions**
Description	To support the robustness of the operations, in the Configuration Parameters dialog box, on the **Optimization** pane, A — Clear **Remove code that protects against division arithmetic exceptions**.
Note	Avoid division-by-zero exceptions. If you clear **Remove code that protects against division arithmetic exceptions**, the code generator produces code that guards against division by zero for fixed-point data.
Rationale	A — Protect against divide-by-zero exceptions for fixed-point code.
Model Advisor Checks	**By Task > Modeling Standards for DO-178C/DO-331 > "Check safety-related optimization settings"**
References	• IEC 61508-3, Table A.3 (3) 'Language Subset' IEC 61508-3 Table A.4 (3) 'Defensive Programming' • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA-C:2004, Rule 21.1 • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate'
See Also	"Remove code that protects against division arithmetic exceptions" in the Simulink documentation
Last Changed	R2013b

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

ID: Title	hisl_0055: Prioritized configuration objectives for high-integrity systems	
Description	Prioritize objectives for high-integrity systems using the Code Generation Advisor by:	
	A	Assigning the highest priority to the 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. Review the resulting parameter configurations to verify that safety requirements are met.	
Rationale	A, B	When you use the Code Generation Advisor, configuration parameters conform to the objectives that you want and they are consistently enforced.
References	• DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' • IEC61508–3, Table A.3 (3) 'Language Subset' IEC 61508–3, Table A.4 (3) 'Defensive Programing' • ISO 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming'	

ID: Title	hisl_0055: Prioritized configuration objectives for high-integrity systems
See also	• "Set Objectives — Code Generation Advisor Dialog Box" • "Manage a Configuration Set" • "cgsl_0301: Prioritization of code generation objectives for code efficiency"
Last Changed	R2013b

MISRA-C:2004 Compliance Considerations

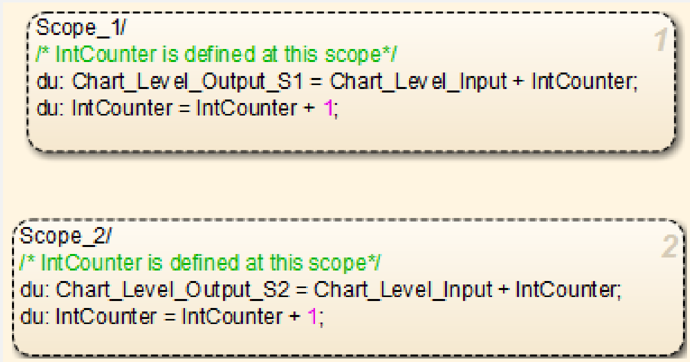
Modeling Style

In this section...

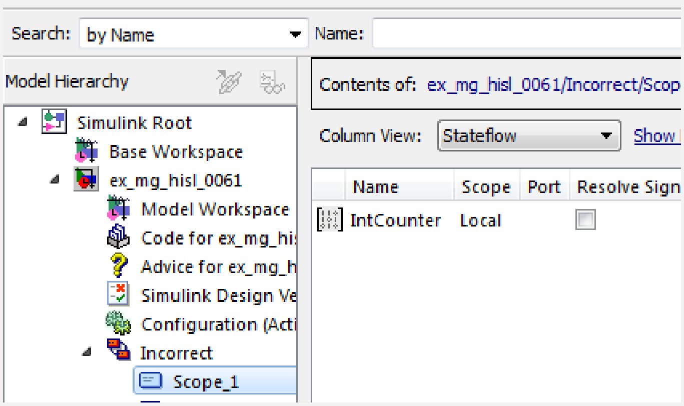
hisl_0061: Unique identifiers for clarity

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-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508–3, Table A.3 (3) 'Language subset' IEC 61508–3, Table A.4 (5) 'Design and coding standards' • ISO 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (e) 'Use of established design principles' ISO 26262-6, Table 1 (h) 'Use of naming conventions' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.12 (1) 'Coding Standard'	
See Also	"Code Appearance" in the Simulink Coder™ documentation	
Last Changed	R2013b	

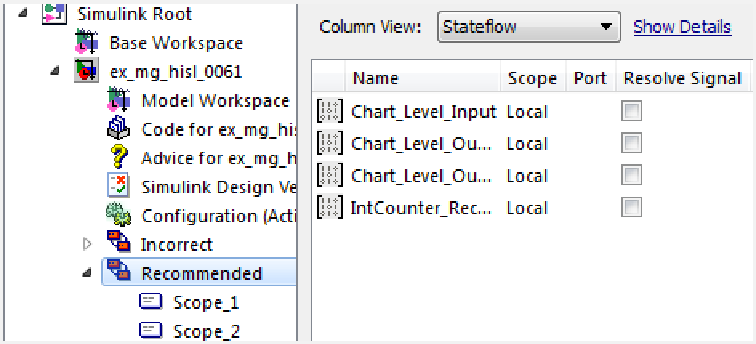
<table>
<tr><th>ID: Title</th><th>hisl_0061: Unique identifiers for clarity</th></tr>
<tr><td>Examples</td><td>In the following example, two states use identifier IntCounter.



The identifier IntCounter is defined for two states, Scope_1 and Scope_2.



Not Recommended</td></tr>
<tr><td></td><td>To clarify the model, create unique identifiers—for example, IntCounter_S1 and IntCounter_S2—or define IntCounter at the parent level.</td></tr>
</table>

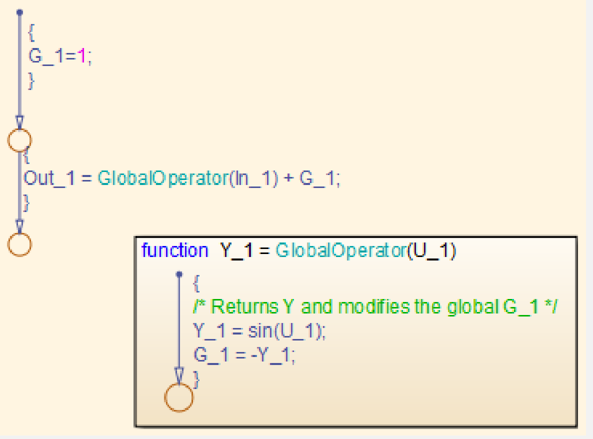
ID: Title	hisl_0061: Unique identifiers for clarity
	 **Recommended**

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 26262-6, Table 1 (b) 'Use of language subsets' ISO 26262-6, Table 1 (f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (h) 'Use of naming conventions' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.12 (1) 'Coding Standard' EN 50128, Table A.12 (2) 'Coding Style Guide' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA-C: 2004 12.2 MISRA-C: 2004 12.4	
Last Changed	R2013b	
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

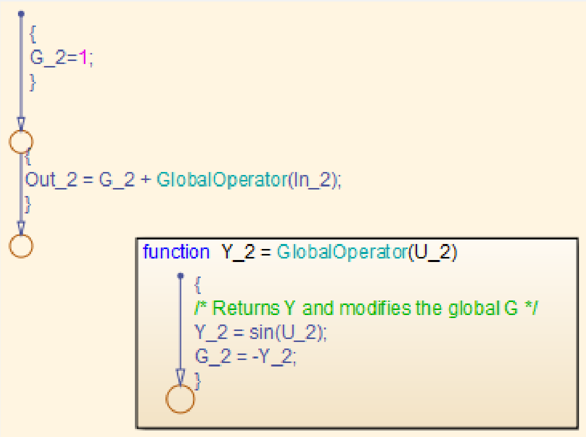
In the following example, the function `GlobalOperator` overwrites the initial value of *`G_1`*,




```
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 the 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.IntEnumType
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 MISRA-C:2004 compliance of the generated code, limit the length of signal and parameter names to 31 characters or fewer when using any of the following storage classes: • 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

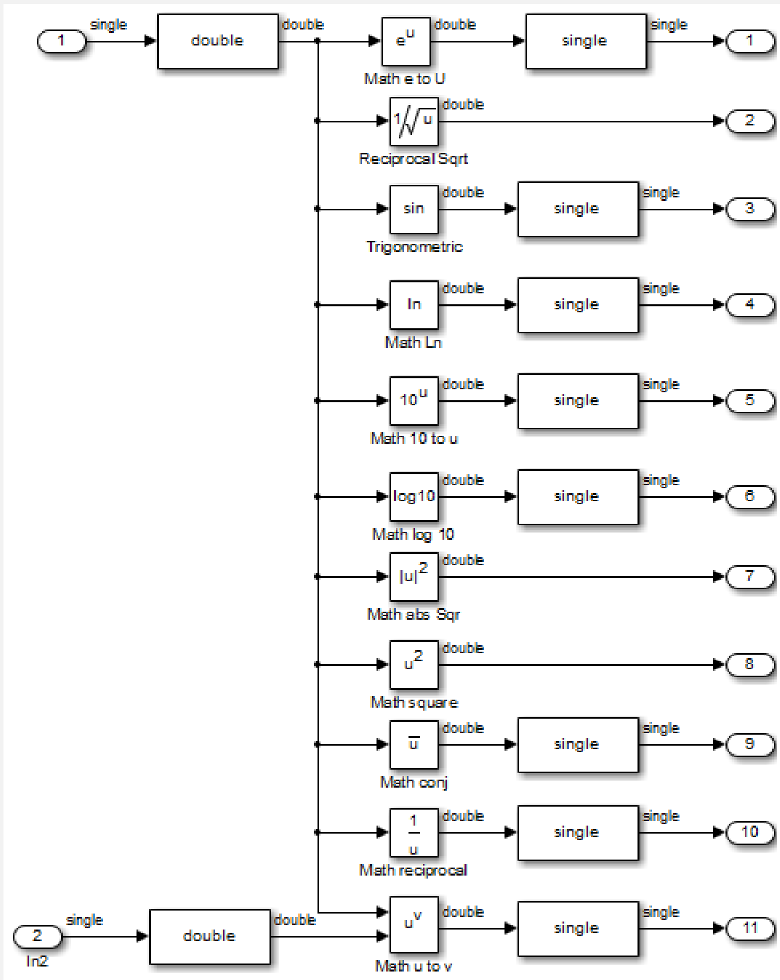
hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance

ID: Title	**hisl_0201: Define reserved keywords to improve MISRA-C:2004 compliance**	
Description	To improve MISRA-C: 2004 compliance of the generated code, define reserved keywords to prevent identifier clashes within the project namespace.	
	A	In the Configuration Parameters dialog box, on the **Simulation Target > Symbols > Reserved names** pane, define reserved identifiers.
	B	Use a consistent set of reserved identifiers for all models.
Notes	Simulink Coder checks models for standard C language key words. Expand the list of reserved identifiers to include project specific identifiers. Examples include target-specific clashes, standard and custom library clashes, and other identified clashes.	
Rationale	Improve MISRA-C:2004 compliance of the generated code.	
See Also	• "Simulation Target Pane: Symbols" in the Simulink documentation • "Reserved Keywords" in the Simulink Coder documentation • "Reserved names" in the Simulink Coder documentation	
References	MISRA-C:2004, Rule 20.2	
Last Changed	R2011b	

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

ID: Title	hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance
Description	To improve MISRA-C:2004 compliance of generated code, insert a data type conversion block when using signals of type single (`real32_T`) as inputs to the following blocks: • Math • Trigonometry • Sqrt The data type conversion block to changes the data type to double (`real_T`)
Rationale	Improve MISRA-C:2004 compliance of the generated code.
Notes	The function prototypes for many math functions require an input of type double. To accommodate the function prototype, you can add a data type conversion block. As an alternative to the data type conversion block, you could define a new function interface using the Target Function Library (TFL).

ID: Title	hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance
References	• MISRA-C: 2004 Rule 10.2

ID: Title	hisl_0202: Use of data conversion blocks to improve MISRA-C:2004 compliance
Last Changed	R2012a
Example	 **Recommended**

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

Block Usage

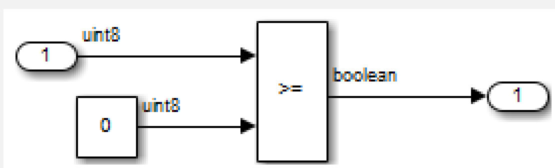
In this section...

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

ID: Title	hisl_0020: Blocks not recommended for MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code,	
	A	Use only blocks that support code generation, as documented in the Simulink Block Support Table
	B	Do not use blocks that are listed as "Not recommended for production code" in the Simulink Block Support Table
Notes	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard. 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 the 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	

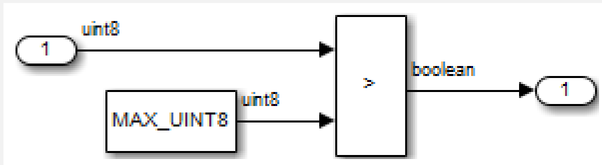
hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance

<table>
<tr><th>ID: Title</th><th>hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance</th></tr>
<tr><td>Description</td><td>To improve MISRA-C:2004 compliance of generated code, avoid comparison operations with invariant results. Comparison operations are performed by the following blocks:

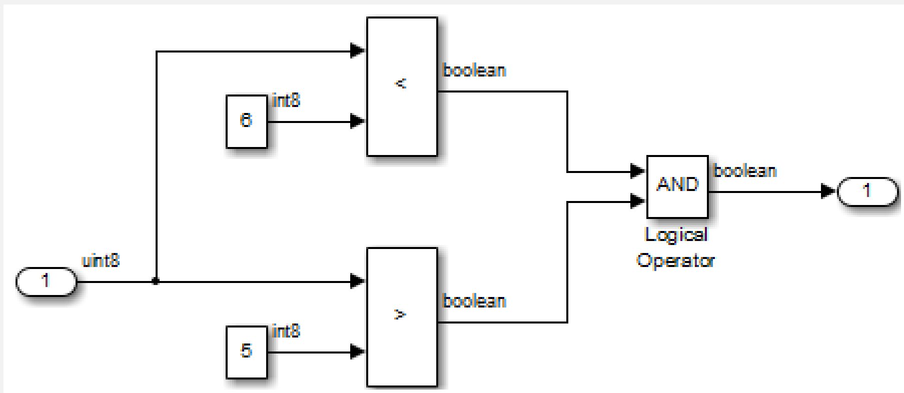
• If
• Logic
• Relational Operator
• Switch
• Switch Case
• Compare to Constant</td></tr>
<tr><td>Rationale</td><td>Improve MISRA-C:2004 compliance of the generated code.</td></tr>
<tr><td>References</td><td>• MISRA-C: 2004 Rule 13.7
• MISRA-C: 2004 Rule 14.1</td></tr>
<tr><td>Last Changed</td><td>R2012a</td></tr>
<tr><td>Example</td><td>Invariant comparisons can occur in simple or compound comparison operations. In compound comparison operations, the individual components can be variable when the full calculation is invariant.
Simple: A <code>uint8</code> is always greater then or equal to 0.


Simple: A <code>uint8</code> cannot have a value greater then 256</td></tr>
</table>

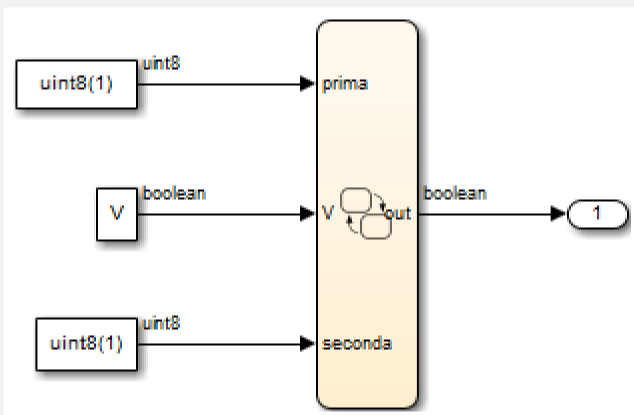
ID: Title	hisl_0101: Avoid invariant comparison operations to improve MISRA-C:2004 compliance

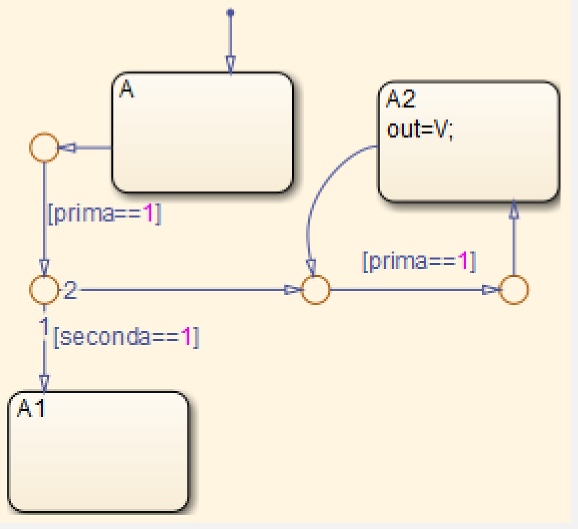


Compound: The comparison operations are mutually exclusive



Stateflow:



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

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

ID: Title	hisl_0102: Data type of loop control variables to improve MISRA-C:2004 compliance
Description	To improve MISRA-C:2004 compliance of generated code, use integer data type for variables that are used as loop control counter variables in: • For and while loops constructed in Stateflow and MATLAB. • While Iterator and For Iterator blocks.
Rationale	Improve MISRA-C:2004 compliance of the generated code.
References	• MISRA-C: 2004 Rule 13.7
Last Changed	R2012a

Configuration Settings

In this section...

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 rowspan="2">Description</td><td colspan="2">To improve MISRA-C:2004 compliance of the generated code,</td></tr>
<tr><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>
</table>
</td></tr>
</table>

ID: Title	hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance		
		MAT-file logging	Cleared (`off`)
		Code replacement library	`C89/C90 (ANSI)`
		Code Generation > Code Style pane	
		Parenthesis level	`Maximum (Specify precedence with parentheses)`
		Code Generation > Symbols pane	
		Maximum identifier length	31
Note	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard.		
Rationale	A	Improve MISRA-C:2004 compliance of the generated code.	
Model Advisor Checks	**By Product > Embedded Coder > "Check configuration parameters for MISRA-C:2004 compliance"**		
References	• MISRA-C:2004		
Last Changed	R2011a		

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

ID: Title	hisl_0312: Specify target specific configuration parameters to improve MISRA-C:2004 compliance	
Description	To improve MISRA-C:2004 compliance of generated code, use a consistent set of model parameters. The parameters include, but are not limited to:	
	A	Explicitly setting model character encoding using the `slCharacterEncoding(encoding)` function.
	B	In the Configuration Parameters dialog box, explicitly selecting a **Hardware Implementation > Production hardware > Signed integer division rounds to:** parameter.
	C	If complex numbers are not required, deselecting the **Code Generation > Interface > Software Environment > complex numbers** parameter.
Notes	Base the selection of the integer division method on the target hardware and compiler. When available, in the Configuration Parameters dialog box, specify both of these parameters: • **Hardware Implementation > Production hardware > Device vendor** • **Hardware Implementation > Production hardware > Device type**	
Rationale	Improve MISRA-C:2004 compliance of the generated code.	
See Also	• "Configure Test and Production Target Hardware" in the Simulink Coder documentation. • slCharacterEncoding in the Simulink documentation. • "hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance"	
References	• MISRA-C: 2004 Rule 3.2 • MISRA-C: 2004 Rule 3.3 • MISRA-C: 2004 Rule 5.7	
Last Changed	R2012a	

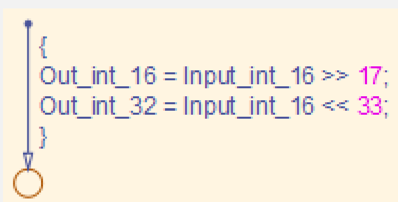
hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance

ID: Title	hisl_0313: Selection of bitfield data types to improve MISRA-C:2004 compliance
Description	To improve MISRA-C:2004 compliance of generated code when bitfields are used, in the Configuration Parameters dialog box, set **Optimization > Signals and Parameters > Code generation > Bitfield declarator type specifier** to `uint_T`.
Rationale	Improve MISRA-C:2004 compliance of the generated code.
Notes	Set **Bitfield declarator type specifier** to `uint_T` if any of the following Optimization parameters are enabled: • **Optimization > Signals and Parameters > Code generation > Pack Boolean data into bitfields** • **Optimization > Stateflow > Code generation > Use bitsets for storing state configuration** • **Optimization > Stateflow > Code generation > Use bitsets for storing Boolean data**
See Also	• "Optimization Pane: Signals and Parameters" in the Simulink documentation.
References	• MISRA-C: 2004 Rule 6.4
Last Changed	R2012a

Stateflow Chart Considerations

In this section...

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

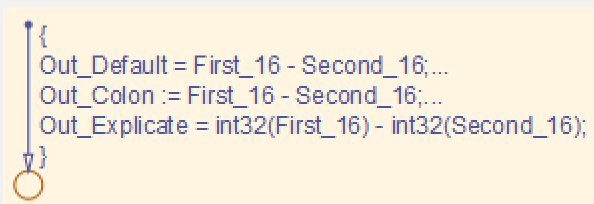
ID: Title	**hisf_0064: Shift operations for Stateflow data to improve MISRA-C:2004 compliance**	
Description	To improve MISRA-C:2004 compliance of the generated code with Stateflow bit-shifting operations, do not perform:	
	A	Right-shift operations greater than the bit-width of the input type, or by a negative value.
	B	Left-shift operations greater than the bit-width of the output type, or by a negative value.
Note	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard.	
Rationale	A,B	To avoid shift operations in the generated code that might be a MISRA-C:2004 violation.
References	• MISRA-C:2004 Rule 12.8	
Prerequisites	"hisl_0060: Configuration parameters that improve MISRA-C:2004 compliance"	
Last Changed	R2011a	
Example	In the first equation, shifting 17 bits to the right pushes data stored in a 16–bit word out of range. The resulting output is zero. In the second equation, shifting the data 33 bits pushes data beyond the range of storage for a 32–bit word. Again, the resulting output is zero. 	

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

```
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	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the MISRA-C:2004 standard.
Rationale	A,B — To avoid shift operations in the generated code that might be a MISRA-C:2004 violation.
References	• 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).



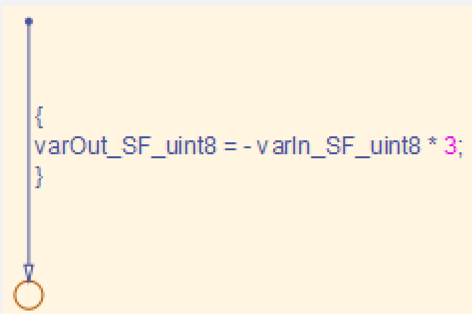
ID: Title	hisf_0065: Type cast operations in Stateflow to improve MISRA-C:2004 compliance

	`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;`
	`}`

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

<table>
<tr><th>ID: Title</th><th colspan="2">hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance</th></tr>
<tr><td rowspan="2">Description</td><td colspan="2">To improve MISRA-C:2004 compliance of the generated code:</td></tr>
<tr><td>A</td><td>Do not use unary minus operators on unsigned data types</td></tr>
<tr><td>Note</td><td colspan="2">The Stateflow action language does not restrict the use of unary minus operators on unsigned expressions.</td></tr>
<tr><td>Rationale</td><td>A</td><td>Improve MISRA-C:2004 compliance of the generated code.</td></tr>
<tr><td>References</td><td colspan="2">• MISRA-C:2004 Rule 12.9</td></tr>
<tr><td>Last Changed</td><td colspan="2">R2011b</td></tr>
<tr><td>Example</td><td colspan="2">

Not Recommended:




```
/* Gateway: Chart */
/* During: Chart */
/* Transition: '<S1>:1' */
varOut_SF_uint8 = (uint8_T)(-varIn_SF_uint8 * 3);
```

Applying the unary minus operator to the unsigned integer results in a MISRA-C:2004 Rule 12.9 violation. The resulting output wraps around the maximum value of 256 (uint8). In this example, if the input variable

</td></tr>
</table>

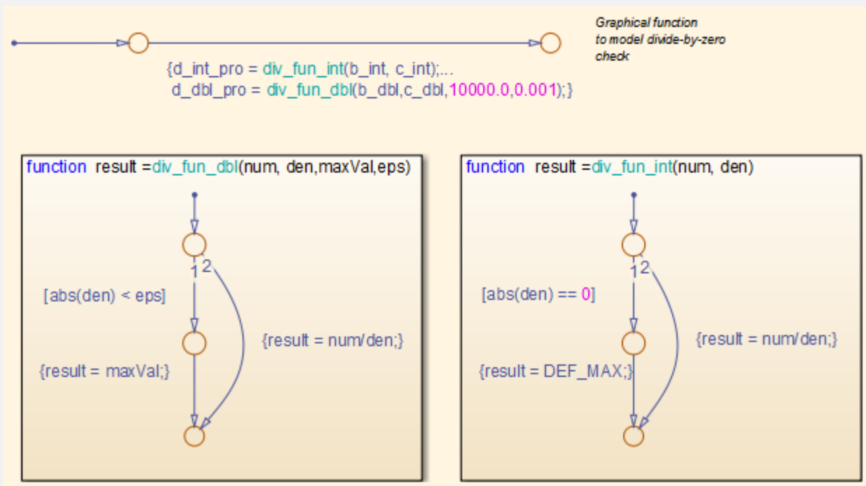
ID: Title	**hisf_0211: Protect against use of unary operators in Stateflow Charts to improve MISRA-C:2004 compliance**
	`In_SF_uint8` equals 7, then the output variable `varOut_uint8` equals 256 – (7 * 3), or 235. The simulation and code generation values are in agreement.

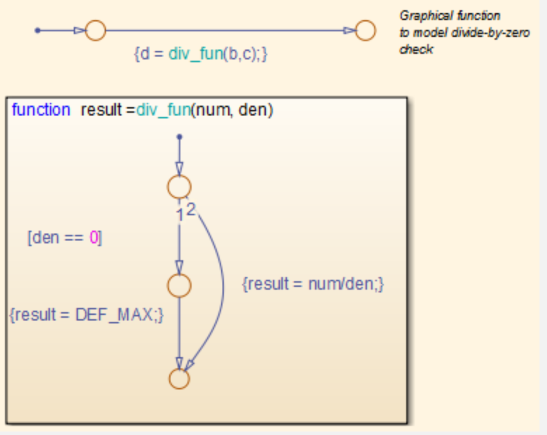
hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance

ID: Title	hisf_0212: Data type of Stateflow for loop control variables to improve MISRA-C: 2004 compliance	
Description	To improve MISRA-C:2004 compliance of the generated code:	
	A	Explicitly select an integer data type as the control variable in a Stateflow for loop
Note	The default data type in Simulink and Stateflow is double. Explicitly select an integer data type.	
Rationale	A	Improve MISRA-C:2004 compliance of the generated code
References	• MISRA-C:2004 Rule 13.4	
Last Changed	R2011b	

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

ID: Title	**hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance**	
Description	To improve MISRA-C:2004 compliance of the generated code for floating point and integer-based operations, do one of the following:	
	A	Perform static analysis of the model to prove that division by zero is not possible
	B	Provide run-time error checking in the generated C code by explicitly modeling the error checking in Stateflow
	C	Modify the code generation process using Code Replacement Libraries (CRLs) to protect against division by zero
	D	For integer-based operations, in the Configuration Parameters dialog box, on the **Optimization** pane, clear **Remove code that protects against division arithmetic exceptions**
Note	Using run-time error checking introduces additional computational and memory overhead in the generated code. It is preferable to use static analysis tools to limit errors in the generated code. You can use Simulink Design Verifier or Polyspace® Code Prover™ to perform the static analysis. If static analysis determines that sections of the code can have a division by zero, then add run-time protection into that section of the model (see example). Using a modified CRL or selecting the parameter **Remove code that protects against division arithmetic exceptions** protects division operations against divide-by-zero operations. However, this action does introduce additional computational and memory overhead. Use only one of the run-time protections (B, C or D) in a model. Using more than one option can result in redundant protection operations.	
Rationale	A,B, C,D	Improve MISRA-C:2004 compliance of the generated code
References	• MISRA-C:2004 Rule 21.1	

ID: Title	**hisf_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA-C: 2004 compliance**
See Also	• "Introduction to Code Replacement Libraries" • "hisl_0002: Usage of Math Function blocks (rem and reciprocal)" • "hisl_0005: Usage of Product blocks" • "hisl_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions"
Last Changed	R2011b
Example	Run-time divide by zero protection can be realized using a graphical function. Unique functions should be provided for each data type. 

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

System Level

In this section...

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

ID: Title	hisl_0401: Encapsulation of code to improve MISRA-C:2004 compliance
Description	To improve the MISRA-C:2004 compliance of the generated code, encapsulate manually inserted code. This code includes, but is not limited to, C, Fortran, and assembly code.
Rationale	Improve MISRA-C:2004 compliance of the generated code
See Also	• "External Code Integration" in the Embedded Coder documentation. • "External Code Integration" in the Simulink Coder documentation.
Notes	Simulink provides multiple methods for integrating existing code. The user is responsible for encapsulating the generated code. Encapsulation can be defined as "the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation" [a]
References	• MISRA-C: 2004 Rule 2.1
Last Changed	R2012a

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

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

ID: Title	hisl_0402: Use of custom #pragma to improve MISRA-C:2004 compliance	
Description	To improve the MISRA-C:2004 compliance of the generated code, document user defined pragma. In the documentation, include:	
	A	Memory range (start and stop address)
	B	Intended use
	C	Justification for using a pragma
Rationale	Improve MISRA-C:2004 compliance of the generated code	
See Also	• "About Memory Sections" in the Embedded Coder documentation. • "Document Generated Code with Simulink Report Generator™" in the Simulink Coder documentation.	
Notes	The Simulink Report Generator documents pragmas.	
References	• MISRA-C: 2004 Rule 3.4	
Last Changed	R2012a	

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

ID: Title	hisl_0403: Use of char data type to improve MISRA-C:2004 compliance	
Description	To improve the MISRA-C:2004 compliance of the generated code with custom storage classes that use the Char data type, only use:	
	A	Plain char type for character values.
	B	Signed and unsigned char type for numeric values.
Rationale	Improve MISRA-C:2004 compliance of the generated code.	
See Also	• "Custom Storage Classes" in the Embedded Coder documentation. • "About Memory Sections" in the Embedded Coder documentation. • "Document Generated Code with Simulink Report Generator" in the Simulink Coder documentation.	
References	• MISRA-C: 2004 Rule 6.1 • MISRA-C: 2004 Rule 6.2	
Last Changed	R2012a	