MBD Platform

Software Modeling Guidelines

How to Contact MathWorks

Latest news: [www.mathworks.com/](https://www.mathworks.com/)

Sales and services: [www.mathworks.com/sales\_and\_services](https://www.mathworks.com/sales_and_services)

User community: [www.mathworks.com/matlabcentral](https://www.mathworks.com/matlabcentral)

Technical support: [www.mathworks.com/support/contact\_us](https://www.mathworks.com/support/contact_us)

Phone: 508-647-7000

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

Software Modeling Guidelines

© COPYRIGHT 2021 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](https://www.mathworks.com/patents) for more information.

Document Title: **Software Modeling Guidelines for** *<Project>*

**Document Number:** *<Doc No>*

Revision: *<Revision>*

Project: *<Project>*

|  |  |  |
| --- | --- | --- |
| Approvals: |  |  |
|  |  |  |
|  |  |  |
| *<Name>*, Author |  | Date |
|  |  |  |
| *<Name>*, Project Management |  | Date |
|  |  |  |
| *<Name>*, Engineering |  | Date |
|  |  |  |
| *<Name>*, Quality Engineering |  | Date |

| Change History | | | | |
| --- | --- | --- | --- | --- |
| Revision | Modification/Description | Date | Author | Checked |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[1 Introduction 1-1](#_Toc88563953)

[1.1 Purpose and Scope 1-1](#_Toc88563954)

[1.2 Applicable Documents 1-1](#_Toc88563955)

[1.3 Referenced Documents 1-2](#_Toc88563956)

[2 Methods, Tools, and Modeling Languages 2-1](#_Toc88563957)

[2.1 Simulink 2-1](#_Toc88563958)

[2.2 Stateflow 2-1](#_Toc88563959)

[2.3 MATLAB 2-2](#_Toc88563960)

[3 Style Guidelines and Complexity Restrictions 3-1](#_Toc88563961)

[3.1 Modeling Guidelines for High-Integrity Systems 3-1](#_Toc88563962)

[3.1.1 Simulink Block Considerations 3-1](#_Toc88563963)

[3.1.2 Stateflow Chart Considerations 3-28](#_Toc88563964)

[3.1.3 MATLAB Function and MATLAB Code Considerations 3-37](#_Toc88563965)

[3.1.4 Requirements Considerations 3-47](#_Toc88563966)

[3.1.5 MISRA C:2012 Compliance Considerations 3-49](#_Toc88563967)

[3.2 MathWorks Advisory Board Guidelines 3-57](#_Toc88563968)

[3.2.1 Simulink 3-57](#_Toc88563969)

[3.2.2 Stateflow 3-64](#_Toc88563970)

[3.3 Company-Specific Guidelines 3-70](#_Toc88563971)

[4 Constraints on Modeling Tools 4-1](#_Toc88563972)

[4.1 Modeling Guidelines for High-Integrity Systems 4-1](#_Toc88563973)

[4.1.1 Configuration Parameter Considerations 4-1](#_Toc88563974)

[4.1.2 MISRA C:2012 Compliance Considerations 4-19](#_Toc88563975)

[4.2 Company-Specific Constraints 4-21](#_Toc88563976)

[5 Model Requirements and Traceability 5-1](#_Toc88563977)

[6 Derived Requirements 6-1](#_Toc88563978)

[7 Additional Model Elements 7-1](#_Toc88563979)

[8 Suitability of Technique 8-1](#_Toc88563980)

[9 Appendix A: Mapping of Modeling Guideline to ISO 26262-6 9-1](#_Toc88563981)

[9.1 Topics to be Covered by Modeling and Coding Guidelines 9-1](#_Toc88563982)

[9.2 Design Principles for Software Unit Design and Implementation 9-4](#_Toc88563983)

# Introduction

## Purpose and Scope

This document comprises the Software Modeling Guidelines as referenced by the Documentation of the Software Development Environment for the project *<Project>*. The Software Modeling Guidelines define the modeling techniques for each type of model.

This document defines all the following:

* Methods and tools for developing the models.
* Modeling languages.
* Style guidelines and complexity restrictions for the use of the modeling languages and tools.
* Constraints on the use of the modeling tool(s).
* Method to identify and delimit the requirements contained in the model.
* Means to establish traceability between requirements and other life cycle data.
* Method to identify and delimit the derived requirements contained in the model.
* Method to provide derived requirements to the system processes.
* Means to identify each model element that does not contribute to the representation of a software requirement or of the software architecture and is not an input to a subsequent software development process or activity.
* Rationale for the suitability of the technique for the type of information expressed by a Specification Model or Design Model.

Within the project *<Project>*, this document is applicable to design models.

You can use this template as a resource when creating a Software Modeling Guidelines document. If you are updating an existing Software Modeling Guidelines document to support Model-Based Design (MBD), you can use this template as a reference.

|  |  |
| --- | --- |
| **Note** | Except for MATLAB®, Simulink®, and Stateflow®, this document does not include definitions or information for modeling environments. |

## Applicable Documents

Table . Regulations and Standards

| **ID** | **Document Title** |
| --- | --- |
| ISO 26262-6 | Road Vehicles – Functional Safety – Part 6: Product Development at the Software Level. ISO, 2018 |
| ISO 26262-8 | Road Vehicles – Functional Safety – Part 8: Supporting Processes. ISO, 2018 |
| ISO 26262-9 | Road Vehicles – Functional Safety – Part 9: Automotive Safety Integrity Level (ASIL)-Oriented and Safety-Oriented Analyses. ISO, 2018 |
|  | *<List additional documents here, e.g. Advisory Circulars, EASA Certification Memos, etc.>* |

Table . Organizational & Project Plans, Standards, and Documents

| ID | Document Title |
| --- | --- |
| SDE | **Documentation of the Software Development Environment for** *<Project>* |
| SVP | **Software Verification Plan for** *<Project>* |
| SCMP | **Software Configuration Management Plan for** *<Project>* |
| SCCP | **Software Change Management (Control) Plan for** *<Project>* |
| SMG | **Software Modeling Guidelines for** *<Project>* |
| SCG | **Software Coding Guidelines for** *<Project>* |
|  | *<List additional documents here.>* |

This initial release will identify the versions of completed documents, versions of the tools used, and the initial software configuration.

If any of the plans are revised during the project, the reasons for the changes must be captured and documented.

## Referenced Documents

Table 3. Reference Materials

| ID | Document Title |
| --- | --- |
| HISM | Modeling Guidelines for High-Integrity Systems |
| MAB | MathWorks Advisory Board Guidelines (Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow) |
|  | *<List additional documents here.>* |

# Methods, Tools, and Modeling Languages

This section specifies the methods, tools, and modeling languages for the development of design models.

Simulink® products from MathWorks® are an accepted standard for Model-Based Design (MBD). Simulink, Fixed-Point Designer™, and Stateflow® software support graphical modeling with time-based block diagrams and event-based state machines. Embedded Coder software supports code generation for embedded systems.

## Simulink

Simulink is a software package that enables you to model, simulate, and analyze dynamic systems. Embedded Coder is a software package that enables you to generate C code for embedded platforms. Simulink uses block diagrams as a model-based programming language (Simulink language). Block diagrams graphically consist of blocks and lines (signals).

* Simulink block diagrams define time-based relationships between signals and state variables. The solution of a block diagram is obtained by evaluating these relationships over time. Time starts at a user specified start time and ends at a user specified stop time. Each evaluation of the relationships is referred to as a time step.
* Signals represent quantities that change over time and are defined for all points in time between the block diagram’s start and stop time.
* A set of equations represented by blocks defines the relationships between signals and state variables. Each block consists of a set of equations (block methods). These equations define a relationship between the input signals, output signals, and the state variables. Inherent in the definition of an equation is the notion of parameters, which are the coefficients found in the equation.
* Simulink contains both virtual blocks and nonvirtual blocks, as described in the Simulink User’s Guide and Simulink Reference documents. In general, implementation is defined by nonvirtual blocks, not virtual blocks. Examples of virtual blocks include DOC blocks, subsystems used to group blocks together in a model, and some connections inside virtual subsystems, such as inports or outports. The modeling standards for the project should define these types of virtual blocks as not contributing to the implementation.
* The Simulink User’s Guide and Simulink Reference documents provide detailed descriptions of Simulink features.
* The Embedded Coder User’s Guide document provides detailed descriptions of Simulink features that apply to code generation for embedded systems, including a list of supported blocks.
* The Simulink Code Inspector Reference document provides detailed descriptions of Simulink features that apply to code inspection, including a list of supported blocks.
* The Fixed-Point Designer User’s Guide and Fixed-Point Designer Reference documents provide detailed description of Simulink’s fixed-point features.

## Stateflow

Stateflow® is an environment for modeling and simulating combinatorial and sequential decision logic in the form of state transition diagrams, flow charts, state transition tables, and truth tables. A state transition diagram is a graphical representation of a finite state machine. States and transitions form the basic building blocks of a sequential logic system. Another way to represent sequential logic is a state transition table, which allows you to enter the state logic in tabular form. Combinatorial logic can also be represented in a chart with flow charts and truth tables.

Stateflow charts can be blocks in a Simulink® model. The collection of these blocks in a Simulink model is the Stateflow machine.

A Stateflow chart enables the representation of hierarchy, parallelism, and history. You can organize complex systems by defining a parent and offspring object structure. A system with parallelism can have two or more orthogonal states active at the same time. You can also specify the destination state of a transition based on historical information.

* The Stateflow User’s Guide and Stateflow Reference documents provide a description of Stateflow features.
* The Simulink Code Inspector Reference document provides a description of Stateflow features that apply to code inspection.

## MATLAB

MATLAB® is a high-level language and interactive environment for numerical computation, visualization, and programming.

* The MATLAB Coder User’s Guide provides a detailed description of MATLAB features that apply to code generation, including a list of supported functions.

# Style Guidelines and Complexity Restrictions

This section describes the style guidelines and complexity restrictions for the use of MATLAB, Simulink, and Stateflow.

*<*The following guidelines provide a starting point for defining constraints on the use of modeling tools within your project. The guidelines are available *at the* [*MathWorks Documentation Center*](https://www.mathworks.com/help)*.* Tailor the following set of guidelines based on your planned use of the tools and applicability to your project.>

## Modeling Guidelines for High-Integrity Systems

The Simulink Modeling Guidelines for High-Integrity Systems contain guidelines for developing models and generating code for high-integrity systems using Model-Based Design with MathWorks® products. The guidelines provide model setting, block usage, and block parameter considerations for creating models that are complete, unambiguous, statically deterministic, robust, and verifiable.

*<Use the Simulink Modeling Guidelines for High-Integrity Systems when defining your own subset of guidelines. The guidelines are available at the* [*MathWorks Documentation Center*](https://www.mathworks.com/help)*. For this chapter, consider listing the following types of High-Integrity Guidelines:*

* *Simulink Block Considerations*
* *Stateflow Chart Considerations*
* *MATLAB Function and MATLAB Code Considerations*
* *Requirements Considerations*
* *MISRA C:2012 Compliance Considerations>*

### Simulink Block Considerations

Naming Conventions

* hisl\_0031: Model file names
* hisl\_0032: Model object names

hisl\_0031: Model file names

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0031: Model file names** |
| Description | For model file names:   * Use these characters: a-z, A-Z, 0-9, and the underscore (\_). * Use strings that are more than 2 and less than 64 characters (not including the dot and file extension).   Do not:   * Start the name with a number. * Use underscores at the beginning or end of a string. * Use more than one consecutive underscore. * Use underscores in file extensions. * Use reserved identifiers. |
| Rationale | * Readability * Compiler limitations * Model-to-generated code traceability |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Naming > Check model file name**  For check details, see Check model file name. |
| References | * ISO 26262-6, Table 1 (1h) 'Use of naming conventions' |
| See Also | * ar\_0001: Usable characters for file names (Simulink) * ar\_0002: Usable characters for folder names (Simulink) * Reserved Keywords (Embedded Coder) |
| Last Changed | R2021b |
| Examples | **Recommended**   * My\_model.slx   **Not Recommended**   * \_My\_\_model.slx * 2018\_01\_11\_model.slx * New.slx |

hisl\_0032: Model object names

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0032: Model object names** |
| Description | For the following model object names:   * Signals * Parameters * Blocks * Named Stateflow objects (States, Boxes, Simulink Functions, Graphical Functions, Truth Tables)   Use:   * These characters: a-z, A-Z, 0-9, and the underscore (\_). * Strings that are fewer than 32 characters.   Do not:   * Start the name with a number. * Use underscores at the beginning or end of a string. * Use more than one consecutive underscore. * Use reserved identifiers. |
| Notes | Reserved names:   * MATLAB keywords * Reserved keywords for C, C++, and code generation. For complete list, see Reserved Keywords (Simulink Coder). * int8, uint8 * int16, uint16 * int32, uint32 * inf, Inf * NaN, nan * eps * intmin, intmax * realmin, realmax * pi * infinity * Nil |
| Rationale | * Readability * Compiler limitations * Model-to generated code traceability |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Naming > Check model object names**  For check details, see Check model object names. |
| References | * ISO 26262-6, Table 1 (1h) 'Use of naming conventions' * MISRA C:2012, Rule 21.2 |
| See Also | * jc\_0201: Usable characters for Subsystem names (Simulink) * jc\_0211: Usable characters for Inport blocks and Outport blocks (Simulink) * jc\_0231: Usable characters for block names (Simulink) * na\_0019: Restricted variable names (Simulink) |
| Last Changed | R2021b |
| Examples | **Recommended**   * Block name: My\_Controller * Signal name: a\_b   **Not Recommended**   * Block name: My Controller * Signal name: 12a\_\_b |

Math Operations

* hisl\_0001: Usage of Abs block
* hisl\_0002: Usage of remainder and reciprocal operations
* hisl\_0003: Usage of square root operations
* hisl\_0028: Usage of Reciprocal Square Root blocks
* hisl\_0004: Usage of natural logarithm and base 10 logarithm operations
* hisl\_0005: Usage of Product blocks
* hisl\_0029: Usage of Assignment block
* hisl\_0066: Usage of Gain blocks
* hisl\_0067: Protect against divide-by-zero calculations

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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Abs blocks**  For check details, see Check usage of Abs blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2021b | |
| Examples | **Recommended**    **Not Recommended** | |

hisl\_0002: Usage of remainder and reciprocal operations

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0002: Usage of remainder and reciprocal operations** | |
| Description | To support robustness of generated code, when using the Math Function block with remainder-after-division (rem) or reciprocal (reciprocal) functions: | |
| A | Protect the input of the reciprocal function from going to zero. |
| B | Protect the second input of the rem function from going to zero. |
| Notes | You can get a divide-by-zero operation, resulting in an infinite (Inf) output value for the reciprocal function, or a Not-a-Number (NaN) output value for the rem function. To avoid overflows or undefined values, protect the corresponding input from going to zero. | |
| Rationale | Protect against overflows and undefined numerical results. | |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of remainder and reciprocal operations**  For check details, see Check usage of remainder and reciprocal operations. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2021b | |
| 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 operations

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0003: Usage of square root operations** | |
| 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 | Avoid undesirable results in generated code. | |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of square root operations**  For check details, see Check usage of square root operations. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2021b | |
| 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. |
| Notes | 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. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Reciprocal Sqrt blocks**  For check details, see Check usage of Reciprocal Sqrt blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2021b | |
| Examples |  | |

hisl\_0004: Usage of natural logarithm and base 10 logarithm operations

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0004: Usage of natural logarithm and base 10 logarithm operations** | |
| 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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of log and log10 operations**  For check details, see Check usage of log and log10 operations. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2021b | |
| 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. | |

hisl\_0005: Usage of Product blocks

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0005: Usage of Product blocks** |
| Description | When the Product block parameter **Multiplication** is set to Matrix(\*), protect divisor inputs from becoming singular input matrices. |
| Notes | 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. |
| Rationale | Protect against overflows and support robustness of generated code. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 |
| Prerequisites | hisl\_0314: Configuration Parameters > Diagnostics > Data Validity > Signals |
| Last Changed | R2021a |

hisl\_0029: Usage of Assignment blocks

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0029: Usage of Assignment blocks** |
| Description | To support robustness of generated code, when using the Assignment block, initialize array fields before their first use. |
| Notes | If the output vector of the Assignment block is not initialized with an input to the block, elements of the vector might not be initialized in the generated code.  When the Assignment block is used iteratively and all array field are assigned during one simulation time step, you do not need initialization input to the block.  Accessing uninitialized elements of block output can result in unexpected behavior. |
| Rationale | Avoid undesirable results in generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Assignment blocks**  For check details, see Check usage of Assignment blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 9.1 |
| Last Changed | R2021b |
| Examples | **Not Recommended: No initialization input Y0 when block is not used iteratively**      **Recommended: Initialization input Y0 when block is not used iteratively**    **Recommended: Initialize array fields when block is used iteratively** |

hisl\_0066: Usage of Gain blocks

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0066: Usage of Gain blocks** |
| Description | To support traceability of generated code, the value of the Gain block must not resolve to 1. |
| Notes | The code generation process can remove Gain values equal to 1 during optimization, resulting in model elements with no traceable code.  An exception to this rule is setting the Gain value to a named parameter data object with a nonauto storage class. |
| Rationale | Support the generation of traceable code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Gain blocks**  For check details, see Check usage of Gain blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| Last Changed | R2018a |

hisl\_0067: Protect against divide-by-zero calculations

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0067: Protect against divide-by-zero calculations** |
| Description | To support robustness of generated code, when performing divide operations, protect the divisor from going to zero. |
| Notes | To prove that division-by-zero is not possible, perform a static analysis of the model.  If division-by-zero is possible, implement one of the following. Using more than one option can result in redundant protection operations:   * Execute the divide-by-zero Model Advisor check * Modify the code generation process to use Code Replacement Libraries (CRLs) * For integer-based operations, clear configuration parameter **Remove code that protects against division arithmetic exceptions**   Using CRLs or clearing configuration 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, as well as the potential to introduce unreachable code. |
| Rationale | Improve code compliance of generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check for divide-by-zero calculations**  For check details, see Check for divide-by-zero calculations. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 |
| See Also | * What is Code Replacement (Embedded Coder) * Code Replacement Libraries (Embedded Coder) * hisl\_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions |
| Last Changed | R2021a |
| Example | **Incorrect**  Division operation can result in a divide-by-zero scenario.  A picture containing clock, table  Description automatically generated  **Correct**  Graphical function to model divide-by-zero check.  A picture containing screenshot  Description automatically generated |

Ports & Subsystems

* hisl\_0006: Usage of While Iterator blocks
* hisl\_0007: Usage of For Iterator or While Iterator subsystems
* hisl\_0008: Usage of For Iterator Blocks
* hisl\_0010: Usage of If blocks and If Action Subsystem blocks
* hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks
* hisl\_0012: Usage of conditionally executed subsystems
* hisl\_0024: Inport interface definition
* hisl\_0025: Design min/max specification of input interfaces
* hisl\_0026: Design min/max specification of output interfaces
* hisl\_0072: Usage of tunable parameters for referenced models

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, set the While Iterator block parameter **Maximum number of iterations** to a positive integer value. |
| Notes | 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 | Support bounded iterative in the generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of While Iterator blocks**  For check details, see Check usage of While Iterator blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1 |
| Last Changed | R2021b |

hisl\_0007: Usage of For Iterator or While Iterator subsystems

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0007: Usage of For Iterator or While Iterator subsystems** |
| Description | To support unambiguous behavior, when using For Iterator Subsystem or While Iterator Subsystem, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems. |
| Rationale | Avoid ambiguous behavior from the subsystem. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of For and While Iterator subsystems**  For check details, see Check usage of For and While Iterator subsystems. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1 |
| Last Changed | R2018b |
| Examples | The following example causes a warning: the Discrete FIR Filter block is time-dependent and is in a For or While Iterator subsystem. |

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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of For Iterator blocks**  For check details, see Check usage of For Iterator blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1 | |
| Last Changed | R2016a | |

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. |
| 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. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of If blocks and If Action Subsystem blocks**  For check details, see Check usage of If blocks and If Action Subsystem blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1 | |
| Prerequisites | hisl\_0016: Usage of blocks that compute relational operators | |
| Last Changed | R2016b | |
| Examples | **Recommended: Elseif with Else**    **Not Recommended: No Else Path**    **Recommended: Only an If, no Else required** | |

hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks** | |
| Description | To support verifiable generated code, when using the Switch Case block: | |
| A | In the Switch Case block parameter dialog box, select **Show default case**. |
| B | Connect the outports of the Switch Case block to a Switch Case Action Subsystem block. |
| C | Use an integer data type or an enumeration value for the inputs to Switch Case blocks. |
| Notes | The combination of Switch Case and If Action Subsystem blocks enable conditional execution based on input conditions. Provide a default path of execution in the form of a "Default" block. | |
| Rationale | A, B, C | Support generation of verifiable code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Switch Case blocks and Switch Case Action Subsystem blocks**  For check details, see Check usage of Switch Case blocks and Switch Case Action Subsystem blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1 | |
| Prerequisites | hisl\_0016: Usage of blocks that compute relational operators | |
| Last Changed | R2016b | |
| Examples | The following graphic displays an example of providing a default path of execution using a "Default" block. | |

hisl\_0012: Usage of conditionally executed subsystems

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0012: Usage of conditionally executed subsystems** | |
| Description | To support unambiguous behavior, when using conditionally executed subsystems: | |
| A | Specify inherited (-1) sample times for all blocks in the subsystem, except Constant. Constant blocks can use infinite (Inf) sample time. |
| B | If the subsystem is called asynchronously, avoid using sample time- dependent blocks, such as integrators, filters, and transfer functions, within the subsystem. |
| Rationale | A, B | Support unambiguous behavior. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of conditionally executed subsystems**  For check details, see Check usage of conditionally executed subsystems. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' | |
| Last Changed | R2018b | |
| 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 or Simulink signal object that explicitly resolves to the connected signal line, set the following block parameters:   * **Data type** * **Port dimensions** * **Sample time** |
| Notes | 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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check for root Inports with missing properties**  For check details, see Check for root Inports with missing properties. |
| References | * ISO 26262-6, Table 1 (1a) 'Enforcement of low complexity' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' |
| Last Changed | R2017b |

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 capabilities1. 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. |
| *1 These capabilities leverage design range information for different purposes. For more information, refer to the documentation for the tools you intend to use.* |
| Rationale | Support precise specification of the input interface. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check for root Inports with missing range definitions**  For check details, see Check for root Inports with missing range definitions. |
| References | * ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' |
| Last Changed | R2017b |

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 capabilities1. 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. |
| *1 These capabilities leverage design range information for different purposes. For more information, refer to the documentation for the tools you intend to use.* |
| Rationale | Support precise specification of the output interface. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check for root Outports with missing range definitions**  For check details, see Check for root Outports with missing range definitions. |
| References | * ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' |
| Last Changed | R2017b |

hisl\_0072: Usage of tunable parameters for referenced models

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0072: Usage of tunable parameters for referenced models** |
| Description | Use the Simulink.Parameter object to define tunable parameters. This applies to all tunable parameters that are meant to be shared via either the base workspace or Simulink data dictionaries. It does not apply to model arguments. |
| Notes | Simulink ignores the storage class settings of parameters that are configured by using the **Model Parameter Configuration** dialog box for referenced models.  This guideline is applicable only when configuration parameter **Default parameter behavior** is set to Inlined. |
| Rationale | Prevent unintended loss of parameter tunability. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check for parameter tunability ignored for referenced models**  For check details, see Check for parameter tunability ignored for referenced models. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| See Also | Create Tunable Calibration Parameter in the Generated Code (Simulink Coder) |
| Last Changed | R2021b |

Signal Routing

* hisl\_0013: Usage of data store blocks
* hisl\_0015: Usage of Merge blocks
* hisl\_0021: Consistent vector indexing method
* hisl\_0022: Data type selection for index signals
* hisl\_0023: Verification of variant blocks
* hisl\_0034: Usage of Signal Routing blocks

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: |
| 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** |
| Notes | 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.  To provide deterministic data transfer between different rates and tasks, use Rate Transition blocks before Data Store Write blocks or after Data Store Read blocks.  In addition to the diagnostics, you can more accurately detect data store memory access violations in your model using Simulink Design Verifier. To do this, on the **Design Verifier** tab, select **Settings**. In the Configuration Parameters dialog box, on the **Design Verifier > Design Error Detection** pane, select **Data store access violations**. For more information, see Detect Data Store Access Violations in a Model. Requires a Simulink Design Verifier license. |
| Rationale | Support consistent data values across different sample times or models. Prevent unintended data corruption. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for data store memory**  For check details, see Check safety-related diagnostic settings for data store memory. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1i) 'Concurrency aspects' |
| Last Changed | R2020b |
| Examples | The following examples use Rate Transition blocks to provide deterministic data transfer between different rates and tasks.   * 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.    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**. |
| D | Set the Outport block parameter **Output when disabled** to held for each conditionally executed subsystem being merged. |
| 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.  Merge block parameter **Allow unequal port widths** is only available when configuration parameter **Underspecified initialization detection** is set to Classic. | |
| Rationale | A, B, C, D | Avoid unambiguous behavior. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Merge blocks**  For check details, see Check usage of Merge blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' | |
| See Also | Merge block (Simulink) | |
| Prerequisites | * hisl\_0303: Configuration Parameters > Diagnostics > Merge block * hisl\_0304: Configuration Parameters > Diagnostics > Model initialization | |
| Last Changed | R2018b | |
| Examples | **Recommended**    **Not Recommended** | |

hisl\_0021: Consistent vector indexing method

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0021: Consistent vector indexing method** | |
| Description | Within a model, use: | |
| A | Consistent vector indexing method.  Support configurable indexing:   * Assignment * For Iterator * Index Vector * Multiport Switch * Selector   Support only one-based indexing:   * Fcn (deprecated) * MATLAB Function * MATLAB System * State Transition Table * Test Sequence * Truth Table * Stateflow chart with MATLAB action language * Truth Table function with MATLAB action language   Support only zero-based indexing:   * Stateflow chart with C action language * Truth Table function with C action language |
| Rationale | A | Reduce the risk of introducing errors due to inconsistent indexing. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check for inconsistent vector indexing methods**  For check details, see Check for inconsistent vector indexing methods. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' | |
| See Also | cgsl\_0101: Zero-based indexing (Simulink) | |
| Last Changed | R2019a | |

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 * 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. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check data types for blocks with index signals**  For check details, see Check data types for blocks with index signals. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' | |
| Last Changed | R2021b | |

hisl\_0023: Verification of variant blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0023: Verification of variant blocks** | |
| Description | When verifying that a model is consistent with generated code, do the following: | |
| A | For each Variant block, set the **Variant activation time** to update diagram or update diagram analyze all choices. |
| Rationale | A | Simplify consistency testing between the model and generated code by restricting the code base to a single variant. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of variant blocks**  For check details, see Check usage of variant blocks. | |
| References | * ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' | |
| See Also | Variant Subsystem, Variant Model (Simulink) | |
| Last Changed | R2021b | |

hisl\_0034: Usage of Signal Routing blocks

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0034: Usage of Signal Routing blocks** |
| Description | When using Switch blocks, avoid comparisons using the ~= operator on floating-point data types. |
| Notes | Due to floating-point precision issues, do not test floating-point expressions for inequality (~=).  When the model contains a Switch block computing a relational operator with the ~= operator, 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 ~= operator within Switch blocks. |
| Rationale | Improve model robustness. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Signal Routing blocks**  For check details, see Check usage of Signal Routing blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 1.1 |
| Last Changed | R2021a |
| Examples | **Not Recommended**  **A screenshot of a cell phone  Description automatically generated**  **Recommended**  A screenshot of a cell phone  Description automatically generated |

Logic and Bit Operations

* hisl\_0016: Usage of blocks that compute relational operators
* hisl\_0017: Usage of blocks that compute relational operators (2)
* hisl\_0018: Usage of Logical Operator block
* hisl\_0019: Usage of bitwise operations
* hisl\_0073: Usage of bit-shift operations

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, avoid using the equality and inequality operators on floating-point data types. |
| Notes | Due to floating-point precision issues, do not test floating-point expressions for equality (==) or inequality (~=, !=). |
| Rationale | Improve model robustness and prevent unexpected results. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check relational comparisons on floating-point signals**  For check details, see Check relational comparisons on floating-point signals. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Dir 1.1 |
| See Also | Relational Operations |
| Last Changed | R2021b |
| Examples | **Ex: 1**  **Example — Correct**   * myDouble > 0.99 && myDouble < 1.01; % test range   **Example — Incorrect**   * myDouble == 1.0 * mySingle ~= 15.0   **Ex: 2**  **Example — Correct**  Equality comparison operators are not used in floating-point operands.    **Example — Incorrect**  Equality comparison operator == is used in floating-point operands.    **Example — Correct**  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.    **Example — Incorrect**  Equality comparison operator == is used in floating-point operands |

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. |
| B | For Relational Operator blocks, ensure that all input signals are of the same data type. |
| Rationale | A, B | Support generation of code that produces unambiguous behavior. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Relational Operator blocks**  For check details, see Check usage of Relational Operator blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Rule 10.1 | |
| See Also | hisl\_0016: Usage of blocks that compute relational operators | |
| Last Changed | R2018a | |

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. |
| B | Ensure all input signals are of type Boolean. |
| Rationale | A, B | Avoid ambiguous behavior of generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of Logical Operator blocks**  For check details, see Check usage of Logical Operator blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Rule 10.1 | |
| Prerequisites | hisl\_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double) | |
| Last Changed | R2017b | |

hisl\_0019: Usage of bitwise operations

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0019: Usage of bitwise operations** | |
| Description | To support unambiguous behavior, when using bitwise operations, | |
| A | Avoid bitwise operations on signed integer data types. |
| Notes | Bitwise operations are not meaningful on signed integers due to unpredictable behavior. For example, a shift operation might move the sign bit into the number, or a numeric bit into the sign bit. | |
| Rationale | A | Support unambiguous behavior of generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of bitwise operations**  For check details, see Check usage of bitwise operations. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 10.1 | |
| See Also | hisl\_0073: Usage of bit-shift operations | |
| Last Changed | R2021b | |

hisl\_0073: Usage of bit-shift operations

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0073: Usage of bit-shift operations** |
| Description | For bit-shifting operations (e.g. a >> b or a << b), do not perform:  Shift operations that are greater than or equal to the bit-width (b must not be equal or greater than the bit width of a). |
| Rationale | Generation of code with shift operations can result in violation of coding standards. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of bit-shift operations**  For check details, see Check usage of bit-shift operations. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 12.2 |
| Last Changed | R2021b |

Lookup Table Blocks

* hisl\_0033: Usage of Lookup Table blocks

hisl\_0033: Usage of Lookup Table blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0033: Usage of Lookup Table blocks** | |
| Description | To support robustness of generated code, when using the 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, Prelookup, and Interpolation Using Prelookup blocks: | |
| A | Clear block parameter **Remove protection against out-of-range input in generated code** in each 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, or Prelookup block. |
| B | Clear block parameter **Remove protection against out-of-range index in generated code** in each Interpolation Using Prelookup block. |
| Notes | If the lookup table inputs are not guaranteed to fall within the range of valid breakpoint values, exclusion of range-checking code may produce unexpected results. | |
| Rationale | A, B | Protect against out-of-range inputs or indices. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check usage of lookup table blocks**  For check details, see Check usage of lookup table blocks. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' | |
| See Also | * [1-D Lookup Table](https://www.mathworks.com/help/simulink/slref/1dlookuptable.html) (Simulink) * [2-D Lookup Table](https://www.mathworks.com/help/simulink/slref/2dlookuptable.html) (Simulink) * [n-D Lookup Table](https://www.mathworks.com/help/simulink/slref/ndlookuptable.html) (Simulink) * [Prelookup](https://www.mathworks.com/help/simulink/slref/prelookup.html) (Simulink) | |
| Last Changed | R2021a | |

### Stateflow Chart Considerations

Chart Properties

* hisf\_0001: State Machine Type
* hisf\_0002: User-specified state/transition execution order
* hisf\_0009: Strong data typing (Simulink and Stateflow boundary)
* hisf\_0011: Stateflow debugging settings

hisf\_0001: State Machine Type

|  |  |
| --- | --- |
| **ID: Title** | **hisf\_0001: State Machine Type** |
| Description | To create Stateflow charts that implement consistent Stateflow semantics, use the same State Machine Type (Classic, Mealy, or Moore) for all charts in the model. |
| Notes | In Mealy charts, actions are associated with transitions. In the Moore charts, actions are associated with states. In Classic charts, actions can be associated with both transition and states.  At compile time, Stateflow 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 | Promote a clear modeling style. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check state machine type of Stateflow charts**  For check details, see Check state machine type of Stateflow charts. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | * Specify Properties for Stateflow Charts (Stateflow) * Create Mealy and Moore Charts (Stateflow) |
| Last Changed | R2018b |

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**. |
| Notes | 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. | |
| Rationale | A | Promote an unambiguous modeling style. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check Stateflow charts for ordering of states and transitions**  For check details, see Check Stateflow charts for ordering of states and transitions. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' | |
| See Also | * Specify Properties for Stateflow Charts (Stateflow) * Evaluate Transitions (Stateflow) * Execution Order for Parallel States (Stateflow) | |
| Prerequisites | hisl\_0311: Configuration Parameters > Diagnostics > Stateflow | |
| Last Changed | R2018b | |

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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check for Strong Data Typing with Simulink I/O**  For check details, see Check for Strong Data Typing with Simulink I/O. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' ISO 26262-6, Table 1 (1h) 'Use of naming conventions' | |
| See Also | Specify Properties for Stateflow Charts (Stateflow) | |
| Last Changed | R2017b | |

hisf\_0011: Stateflow debugging settings

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0011: Stateflow debugging settings** | |
| Description | To protect against unreachable code and indeterminate execution time, | |
| A | Set configuration parameters **Wrap on overflow** and **Simulation range checking** to error.  In the model, open the **Debug** tab and select **Diagnostics > Detect Cyclical Behavior**. |
| 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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check Stateflow debugging options**  For check details, see Check Stateflow debugging options. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' | |
| See Also | Specify Properties of Truth Table Functions (Stateflow) | |
| Last Changed | R2017b | |

Chart Architecture

* hisf\_0003: Usage of bitwise operations
* hisf\_0004: Protect against recursive function calls to improve code compliance
* hisf\_0007: Usage of junction conditions (maintaining mutual exclusion)
* hisf\_0013: Usage of transition paths (crossing parallel state boundaries)
* hisf\_0014: Usage of transition paths (passing through states)
* hisf\_0015: Strong data typing (casting variables and parameters in expressions)
* hisf\_0016: Stateflow port names
* hisf\_0017: Stateflow data object scoping

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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check usage of bitwise operations in Stateflow charts**  For check details, see Check usage of bitwise operations in Stateflow charts. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Rule 10.1 | |
| See Also | hisl\_0019: Usage of Bitwise Operator block | |
| Last Changed | R2016a | |

hisf\_0004: Protect against recursive function calls to improve code compliance

|  |  |
| --- | --- |
| **ID: Title** | **hisf\_0004: Protect against recursive function calls to improve code compliance** |
| Description | To improve compliance of generated code, do not call functions recursively. This includes any combination of graphical functions, truth table functions, MATLAB functions, or Simulink functions. |
| Notes | A recursion exists if a function calls itself directly or indirectly through another function call. |
| Rationale | Promote bounded function call behavior. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check usage of recursions**  For check details, see Check usage of recursions. |
| References | * ISO 26262-6, Table 6 (1j) 'No recursions' * MISRA C:2012, Rule 17.2 |
| Prerequisites | * hisf\_0011: Stateflow debugging settings * hisl\_0311: Configuration Parameters > Diagnostics > Stateflow * hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance |
| Last Changed | R2021a |
| Examples | There are multiple patterns in Stateflow that can result in recursion.  **Recursive Function Calls**    When the default state A is entered, event Evn is broadcast in the entry action of A. Evn results in a recursive call of the interpretation algorithm. Since A is active, the outgoing transition of A is tested. Since the current event Evn matches the transition event (and because of the absence of condition) the condition action is executed, broadcasting Evn again. This results in a new call of the interpretation algorithm which repeats the same sequence of steps until stack overflow.  **Recursive Function Calls** |

hisf\_0007: Usage of junction conditions (maintaining mutual exclusion)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0007: Usage of junction conditions (maintaining mutual exclusion)** | |
| Description | To enhance clarity and prevent the generation of unreachable code, | |
| A | Make junction conditions mutually exclusive. |
| Notes | You can use the design error detection functionality in Simulink Design Verifier to perform the analysis. For more information, see Dead Logic Detection (Simulink Design Verifier). If you have a Simulink Design Verifier license, you can use Model Advisor check Detect Dead Logic. | |
| Rationale | A | Enhance clarity and prevent generation of unreachable code. |
| References | * ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' | |
| Last Changed | R2012b | |
| Examples |  | |

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. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check Stateflow charts for transition paths that cross parallel state boundaries**  For check details, see Check Stateflow charts for transition paths that cross parallel state boundaries. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' | |
| Last Changed | R2017b | |
| Examples | 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. | |

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. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check for inappropriate use of transition paths**  For check details, see Check for inappropriate use of transition paths. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' | |
| Last Changed | R2018b | |
| 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 conditions * Transition actions * State actions |
| 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. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check Stateflow charts for strong data typing**  For check details, see Check Stateflow charts for strong data typing. | |
| References | * ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Rule 10.1 MISRA C:2012, Rule 12.2 | |
| Last Changed | R2017b | |
| Examples | **Recommended**    **Not Recommended** | |

hisf\_0016: Stateflow port names

|  |  |
| --- | --- |
| **ID: Title** | **hisf\_0016: Stateflow port names** |
| Description | The name of a Stateflow input or output must be the same as the corresponding signal. An exception to the guideline is that reusable Stateflow blocks can have different port names. |
| Rationale | Support generation of traceable code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check naming of ports in Stateflow charts**  For check details, see Check naming of ports in Stateflow charts. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| Last Changed | R2018a |

hisf\_0017: Stateflow data object scoping

|  |  |
| --- | --- |
| **ID: Title** | **hisf\_0017: Stateflow data object scoping** |
| Description | Stateflow data objects with local scope must be defined at the chart level or below. |
| Rationale | Support generation of traceable code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check scoping of Stateflow data objects**  For check details, see Check scoping of Stateflow data objects. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| Last Changed | R2018a |
| Examples | **Recommended**    **Not Recommended** |

### MATLAB Function and MATLAB Code Considerations

MATLAB Functions

* himl\_0001: Usage of standardized MATLAB function headers
* himl\_0002: Strong data typing at MATLAB function boundaries
* himl\_0003: Complexity of user-defined MATLAB Functions

himl\_0001: Usage of standardized MATLAB function headers

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0001: Usage of standardized MATLAB function headers** |
| Description | When using MATLAB functions, use a standardized header to provide information about the purpose and use of the function. |
| Rationale | A standardized header improves the readability and documentation of MATLAB functions. The header should provide a function description and usage information. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check usage of standardized MATLAB function headers**  For check details, see Check usage of standardized MATLAB function headers. |
| References | * ISO 26262-6, Table 1 (1g) 'Use of style guides' |
| See Also | * na\_0025: MATLAB Function Header (Simulink) * [Orion GN&C: MATLAB and Simulink Standards](http://www.mathworks.com/aerospace-defense/standards/FltDyn-CEV-08-148_MATLAB_Standards_v9_20111202.pdf), jh\_0073: eML Header * MATLAB Function Block Editor (Simulink) |
| Last Changed | R2018b |
| Examples | A typical standardized function header includes:   * Function name * Description * Inputs and outputs (if possible, include size and type) * Assumptions and limitations * Revision history   Example:  % FUNCTION NAME: % avg % % DESCRIPTION: % Compute the average of three inputs % INPUT: % in1 - (double) Input one % in2 - (double) Input two % in3 - (double) Input three % % OUTPUT: % out - (double) Calculated average of the three inputs % % ASSUMPTIONS AND LIMITATIONS: % None % % REVISION HISTORY: % 05/02/2018 – mmyers % \* Initial implementation % |

himl\_0002: Strong data typing at MATLAB function boundaries

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0002: Strong data typing at MATLAB function boundaries** |
| Description | To support strong data typing at the interfaces of MATLAB functions, 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 | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check for MATLAB Function interfaces with inherited properties**  For check details, see Check for MATLAB Function interfaces with inherited properties. |
| References | * ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' |
| See Also | * na\_0034: MATLAB Function block input/output settings (Simulink) * [Orion GN&C: MATLAB and Simulink Standards](http://www.mathworks.com/aerospace-defense/standards/FltDyn-CEV-08-148_MATLAB_Standards_v9_20111202.pdf), jh\_0063: eML block input/output settings * MATLAB Function Block Editor (Simulink) |
| Last Changed | R2016a |
| 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: Complexity of user-defined MATLAB Functions

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **himl\_0003: Complexity of user-defined MATLAB Functions** | |
| Description | When using MATLAB functions, 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) | |
| Notes | Size and complexity limits can vary across projects. Typical limits might be as described in this table: | |
| **Metric** | **Limit** |
| Lines of code | 60 per MATLAB function |
| Nested function levels | 31,2 |
| 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.* | |
| Rationale | * Readability * Comprehension * Traceability * Maintainability * Testability | |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check MATLAB Function metrics**  For check details, see Check MATLAB Function metrics. | |
| References | * ISO 26262-6, Table 1 (1a) 'Enforcement of low complexity' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' | |
| See Also | * na\_0016: Source lines of MATLAB Functions (Simulink) * na\_0017: Number of called function levels (Simulink) * na\_0018: Number of nested if/else and case statement (Simulink) * [Orion GN&C: MATLAB and Simulink Standards](http://www.mathworks.com/aerospace-defense/standards/FltDyn-CEV-08-148_MATLAB_Standards_v9_20111202.pdf), jh\_0084: eML Comments * MATLAB Function Block Editor (Simulink) | |
| Last Changed | R2021b | |

MATLAB Code

* himl\_0004: MATLAB Code Analyzer recommendations for code generation
* himl\_0006: MATLAB code if / elseif / else patterns
* himl\_0007: MATLAB code switch / case / otherwise patterns
* himl\_0008: MATLAB code relational operator data types
* himl\_0010: MATLAB code with logical operators and functions
* himl\_0011: Data type and size of condition expressions
* himl\_0012: Usage of MATLAB functions for code generation
* himl\_0013: Limitation of built-in MATLAB Function complexity

himl\_0004: MATLAB Code Analyzer recommendations for code generation

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **himl\_0004: MATLAB Code Analyzer recommendations for code generation** | |
| Description | When using MATLAB code: | |
| A | To activate MATLAB Code Analyzer messages for code generations, use the %#codegen directive in external MATLAB functions. |
| B | Review the MATLAB Code Analyzer messages. Either:   * Implement the recommendations or * Justify not following the recommendations with %#ok<message- ID(S)> directives in the MATLAB function. Do not use %#ok without specific message IDs. |
| Notes | The MATLAB Code Analyzer messages provide identifies potential errors, problems, and opportunities for improvement in the code. | |
| Rationale | A | In external MATLAB functions, the %#codegen directive activates MATLAB Code Analyzer messages for code generation. |
| B | * Following MATLAB Code Analyzer recommendations helps to: * Generate efficient code. * Follow best code generation practices * Avoid using MATLAB features not supported for code generation. * Avoid code patterns which potentially influence safety. * Not following MATLAB Code Analyzer recommendations are justified with message id (e.g. %#ok<NOPRT>). * In the MATLAB function, using %#ok without a message id justifies the full line, potentially hiding issues. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check MATLAB Code Analyzer messages**  For check details, see Check MATLAB Code Analyzer messages. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' ISO 26262-6, Table 1 (1h) 'Use of naming conventions' | |
| See Also | Check Code for Errors and Warnings Using the Code Analyzer | |
| Last Changed | R2016a | |
| Examples | **Recommended**   * Activate MATLAB Code Analyzer messages for code generations:   %#codegen function y = function(u)  y = inc\_u(u); end function yy = inc\_u(uu)  yy = uu + 1; end   * Justify missing ; and value assigned might be unused:   y = 2\*u %#ok<NOPRT,NAGSU> output for debugging ... y = 3\*u;   * If output is not desired and assigned value is unused, remove the line y = 2\*u ...:   y = 3\*u;  Not Recommended   * External MATLAB file used in Simulink with missing %#codegen directive:   function y = function(u)  % nested functions can't be used for code generation  function yy = inc\_u(uu)  yy = uu + 1;  end  y = inc\_u(u); end   * All messages in line are justified by using %#ok without a message ID:   % missing ';' and the value might be unused y = 2\*u %#ok ... y = 3\*u;   * No justification:   % missing justification for missing ';' % and unnecessary '[..]' y = [2\*u] | |

himl\_0006: MATLAB code if / elseif / else patterns

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0006: MATLAB code if / elseif / else patterns** |
| Description | For MATLAB code with if / elseif / else constructs, terminate the constructs with an else statement that includes at least a meaningful comment. A final else statement is not required if there is no elseif. |
| Rationale | * Defensive programming * Readability * Traceability |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check if/elseif/else patterns in MATLAB Function blocks**  For check details, see Check if/elseif/else patterns in MATLAB Function blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| See Also | hisl\_0010: Usage of If blocks and If Action Subsystem blocks |
| Last Changed | R2018b |
| Examples | **Recommended**   * if u > 0  y = 1; end * if u > 0  y = 1; elseif u < 0  y = -1; else  y = 0; end * y = 0; if u > 0  y = 1; elseif u < 0  y = -1; else  % handled before if end   **Not Recommended**   * % empty else y = 0; if u > 0  y = 1; elseif u < 0  y = -1; else end * % missing else y = 0; if u > 0  y = 1; elseif u < 0  y = -1; end |

himl\_0007: MATLAB code switch / case / otherwise patterns

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0007: MATLAB code switch / case / otherwise patterns** |
| Description | For MATLAB code with switch statements, include:   * At least two case statements. * An otherwise statement that at least includes a meaningful comment. |
| Notes | If there is only one case and one otherwise statement, consider using an  if / else statement. |
| Rationale | * Defensive programming * Readability * Traceability |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check switch statements in MATLAB Function blocks**  For check details, see Check switch statements in MATLAB Function blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 16.4 |
| See Also | * na\_0022: Recommended patterns for Switch/Case statements (Simulink) * hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks |
| Last Changed | R2018b |
| Examples | **Recommended**   * switch u  case 1  y = 3;  case 3  y = 1;  otherwise  y = 1; end * y = 0; switch u  case 1  y = 3;  case 3  y = 1;  otherwise  % handled before switch end   **Not Recommended**   * % no case statements switch u  Otherwise  y = 1; end * % empty otherwise statement switch u  case 1  y = 3;  case 3  y = 1;  otherwise end * % no otherwise statement switch u  case 1  y = 3; end |

himl\_0008: MATLAB code relational operator data types

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0008: MATLAB code relational operator data types** |
| Description | For MATLAB code with relational operators, use the same data type for the left and right operands. |
| Notes | If the two operands have different data types, MATLAB will promote both operands to a common data type. This can lead to unexpected results. |
| Rationale | * Prevent implicit casts * Prevent unexpected results |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check usage of relational operators in MATLAB Function blocks**  For check details, see Check usage of relational operators in MATLAB Function blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 6 (1g) 'No implicit type conversions' |
| See Also | * hisl\_0016: Usage of blocks that compute relational operators * hisl\_0017: Usage of blocks that compute relational operators (2) |
| Last Changed | R2018b |
| Examples | **Recommended**   * myBool == true myInt8 == int8(1)   **Not Recommended**   * myBool == 1 myInt8 == true myInt8 == 1 myInt8 == int16(1) myEnum1.EnumVal == int32(1) |

himl\_0010: MATLAB code with logical operators and functions

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0010: MATLAB code with logical operators and functions** |
| Description | For logical operators and logical functions in MATLAB code, use logical data types |
| Notes | Logical operators: &&, ||, ~  Logical functions: and, or, not, xor |
| Rationale | Prevent unexpected results |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check usage of logical operators and functions in MATLAB Function blocks**  For check details, see Check usage of logical operators and functions in MATLAB Function blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' |
| Last Changed | R2018b |
| Examples | **Recommended**   * ~myLogical (myInt8 > int8(4)) && myLogical xor(myLogical1,myLogical2)   **Not Recommended**   * ~myInt8 myInt8 && myDouble |

himl\_0011: Data type and size of condition expressions

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0011: Data type and size of condition expressions** |
| Description | Logical scalars should be used for condition expressions. Condition expressions include:   * if expressions * elseif expressions * while expressions * Condition expressions of Stateflow transitions |
| Rationale | Prevent execution of unexpected code paths |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check type and size of condition expressions**  For check details, see Check type and size of condition expressions. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Rule 14.4 |
| Last Changed | R2019b |
| Examples | **Recommended**  Assume variable var is a scalar of type double with value -1.  MATLAB Code:  if var > 0 % expression is a logical scalar  … % will not be executed elseif var < 0 % expression is a logical scalar  … % will be executed else  … % will not be executed end while var < 5 % expression is a logical scalar  var = var + 1; % executed 5 times end  Stateflow Transition Condition:  [var > 0]{…} % condition action will not be executed  **Not Recommended**  Assume variable var is a scalar of type double with value -1.  MATLAB Code:  if var % expression is a double scalar  … % will be executed because var is non-zero elseif ~var  … % will not be executed else  … % will not be executed end while var % expression is a double scalar  var = var + 1; % executed 1 time end  Stateflow Transition Condition:  [var]{…} % condition action will be executed because var is non-zero |

himl\_0012: Usage of MATLAB functions for code generation

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0012: Usage of MATLAB functions for code generation** |
| Description | Use only MATLAB functions that support code generation. |
| Rationale | To detect and avoid the usage of MATLAB functions which are not supported by code generation at earliest possible stages of development. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Check MATLAB functions not supported for code generation**  For check details, see Check MATLAB functions not supported for code generation. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | coder.screener (Simulink) |
| Last Changed | R2021b |

himl\_0013: Limitation of built-in MATLAB Function complexity

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **himl\_0013: Limitation of built-in MATLAB Function complexity** | |
| Description | When authoring MATLAB code, limit the usage of built-in MATLAB functions that may result in generated code that exceeds complexity limits established for your project. | |
| Notes | Complexity limits can vary across projects. Typical limits might be as described in this table: | |
| **Metric** | **Limit** |
| Cyclomatic Complexity (Generated Code) | 10 |
| Rationale | Improve testability and maintainability. | |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > MATLAB > Metrics for generated code complexity**  For check details, see Metrics for generated code complexity. | |
| References | * ISO 26262-6, Table 1 (1a) 'Enforcement of low complexity' | |
| Last Changed | R2021b | |

### Requirements Considerations

Requirements

* hisl\_0070: Placement of requirement links in a model

hisl\_0070: Placement of requirement links in a model

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0070: Placement of requirement links in a model** | |
| Description | Establish bidirectional traceability between model requirements and the model elements that are used to implement the requirement. A single element or combination of elements can link to requirements.  When linking requirements, follow these guidelines. | |
| A | Apply requirement links to the lowest level component of model elements. Model elements that do not impact the model’s behavior or the generated code are exempt from requirement linking. See Notes for additional information |
| B | At the project level, define the maximum number of unique requirement links associated with each component. A minimum of one requirement link is required. |
| C | At the project level, define the maximum number of child model elements for each linked component. |
| Notes | Use Simulink Requirements to trace between the model and the requirements from which the model was developed. Apply user tags to define model elements as derived and/or safety requirements.  To reduce the number of requirements that are linked to a model, apply requirements at the component-level. A component contains a group of model elements, for example:   * In Simulink, a component is a top-level block diagram, subsystem, MATLAB function, or area annotation. * In Stateflow, a component is a chart, superstate, box, Simulink function, graphical function, Simulink State, MATLAB Function, and Truth Table. * In MATLAB, a component is a function. * n System Composer, a Component is an Adapter or a Component block.   Components that contain only these model elements are exempt from requirement linking:   * Model Info, DocBlock, or System Requirements blocks * Area annotations * Model element with requirement links * Commented out model elements   When a linked component contains a nonexempt child model element, the child implements the associated requirement either in part or whole. | |
| Rationale | A | Establishing requirement links at the component level captures the relationship of model elements. In addition, maintainability improves because the need to update requirement links for minor logic changes is reduced. |
| B, C | Support requirement change impact analysis. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Requirements > Check for model elements that do not link to requirements**  For check details, see Check for model elements that do not link to requirements. | |
| References |  | |
| See Also | Requirements Traceability (Simulink Requirements) | |
| Last Changed | R2021a | |
| Examples | **Recommended: Requirement links on parent component**  Requirement link placed at the top level model with no subsystems.    **Recommended: Requirement links placed on area annotation**  Requirement link placed on an area annotation. | |

### MISRA C:2012 Compliance Considerations

Modeling Style

* hisl\_0061: Unique identifiers for clarity
* hisl\_0062: Global variables in graphical functions
* hisl\_0063: Length of user-defined function names to improve MISRA C:2012 compliance

hisl\_0061: Unique identifiers for clarity

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0061: Unique identifiers for clarity** | |
| Description | When developing a model: | |
| A | Use unique identifiers for Simulink signals. |
| B | Define unique identifiers across multiple scopes within a chart. |
| Notes | The code generator resolves conflicts between identifiers so that symbols in the generated code are unique. The process is called name mangling. | |
| Rationale | A, B | Improve readability of a graphical model and mapping between identifiers in the model and generated code. |
| Model Advisor Check | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check Stateflow charts for uniquely defined data objects**  For check details, see Check Stateflow charts for uniquely defined data objects. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' ISO 26262-6, Table 1 (1h) 'Use of naming conventions' ISO 26262-6, Table 6 (1d) 'No multiple use of variable names' | |
| See Also | Code Appearance (Simulink Coder) | |
| Last Changed | R2017b | |
| Examples | **Not Recommended**  In the following example, two states Scope\_1 and Scope\_2 use local identifier IntCounter.    The identifier IntCounter is defined for two states, Scope\_1 and Scope\_2.      **Recommended**  To clarify the model, create unique identifiers. In the following example, state Scope\_1 uses local identifier IntCounter\_Scope\_1. State Scope\_2 uses local identifier IntCounter\_Scope\_2.    The identifier IntCounter\_Scope\_1 is defined for state Scope\_1. Identifier  IntCounter\_Scope\_2 is defined for Scope\_2. | |

hisl\_0062: Global variables in graphical functions

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0062: Global variables in graphical functions** |
| Description | For data with a global scope used in a function, do not use the data in the calling expression if a value is assigned to the data in that function. |
| Rationale | Enhance readability of a model by removing ambiguity in the values of global variables. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check global variables in graphical functions**  For check details, see Check global variables in graphical functions. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1h) 'Use of naming conventions' ISO 26262-6, Table 6 (1e) 'Avoid global variables or else justify their usage' * MISRA C:2012, Rule 13.2 MISRA C:2012, Rule 13.5 |
| Last Changed | R2018b |
| Examples | Consider a graphical function graphicalFunction that modifies the global data G.    **Recommended**    **Not Recommended** |

hisl\_0063: Length of user-defined object names to improve MISRA C:2012 compliance

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0063: Length of user-defined object names to improve MISRA C:2012 compliance** | |
| Description | To improve MISRA C:2012 compliance of generated code, use configuration parameter **Maximum identifier length** (MaxIdLength) to limit the length of user defined names. | |
| **Note**: The default of Maximum identifier length is 31. | |
| A | For Subsystem blocks with parameter **Function name options** set to User specified, limit the length of function names to be equal to or less than the value specified in **Maximum identifier length**. |
| B | Limit the length of data object names to **Maximum identifier length** (MaxIdLength) characters or fewer for:   * Simulink.AliasType * Simulink.NumericType * Simulink.Variant * Simulink.Bus * Simulink.BusElement * Simulink.IntEnumType |
| C | Limit the length of signal and parameter names to **Maximum identifier length** (MaxIdLength) characters or fewer when using the following storage classes:   * Exported Global * Imported Extern * Imported Extern Pointer * Custom storage class |
| **Note**: If specified, this includes the length of the **Identifier** name. |
| Rationale | Length in the generated code can result in a MISRAC:2012 violation. | |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check for length of user-defined object names**  For check details, see Check for length of user-defined object names. | |
| References | * ISO 26262-6, Table 6 (1d) 'No multiple use of variable names' * MISRA C:2012, Rule 5.1 MISRA C:2012, Rule 5.2 MISRA C:2012, Rule 5.3 MISRA C:2012, Rule 5.4 MISRA C:2012, Rule 5.5 | |
| Prerequisites | hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance | |
| Last Changed | R2021a | |

Block Usage

* hisl\_0020: Blocks not recommended for MISRA C:2012 compliance
* hisl\_0101: Avoid invariant comparison operations to improve MISRA C:2012 compliance
* hisl\_0102: Data type of loop control variables to improve MISRA C:2012 compliance

hisl\_0020: Blocks not recommended for MISRA C:2012 compliance

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0020: Blocks not recommended for MISRA C:2012 compliance** | |
| Description | To improve MISRA C:2012 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 |
| C | Do not use Lookup Table blocks using cubic spline interpolation or extrapolation methods. Specific blocks are:   * 1-D Lookup Table * 2-D Lookup Table * n-D Lookup Table |
| D | Do not use deprecated Lookup Table blocks. The deprecated Lookup Table blocks are Lookup and Lookup2D. |
| E | Do not use S-Function Builder blocks in the model or subsystem. |
| F | Do not use From Workspace blocks in the model or subsystem. |
| G | Do not use these String blocks in the model or subsystem:   * Compose String * Scan String * String to Single * String to Double * To String |
| Notes | If you follow this and other modeling guidelines, you can eliminate model constructs that are not suitable for C/C++ production code generation, at the same time, increase the likelihood of generating code that complies with the MISRA C:2012 standard.  Use the Block Support Table block 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, C, D, E, F, G | Improve quality and MISRA C:2012 compliance of the generated code. |
| Model Advisor Checks | To check model for conditions A, B, C, D, E, F, and G:  **Modeling Standards for ISO 26262 > High-Integrity Systems > Code > Check for blocks not recommended for MISRA C:2012**  For check details, see Check for blocks not recommended for MISRA C:2012. | |
| To check model for conditions A and B:  **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check for blocks not recommended for C/C++ production code deployment**  For check details, see Check for blocks not recommended for C/C++ production code deployment. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' | |
| Last Changed | R2018b | |

hisl\_0101: Avoid invariant comparison operations to improve MISRA C:2012 compliance

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0101: Avoid invariant comparison operations to improve MISRA C:2012 compliance** |
| Description | To improve MISRA C:2012 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 |
| Notes | You can use the design error detection functionality in Simulink Design Verifier to perform the analysis. For more information, see Dead Logic Detection (Simulink Design Verifier). If you have a Simulink Design Verifier license, you can use Model Advisor check Detect Dead Logic. |
| Rationale | Improve MISRA C:2012 compliance of the generated code. |
| References | * ISO 26262-6, Table 6 (1h) 'No hidden data flow or control flow' * MISRA C:2012, Rule 2.1 MISRA C:2012, Rule 14.3 |
| Last Changed | R2018a |
| Examples | 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 uint8 is always greater than or equal to 0.    **Simple**: A uint8 cannot have a value greater than 256.    **Compound**: The comparison operations are mutually exclusive.    **Stateflow**: |

hisl\_0102: Data type of loop control variables to improve MISRA C:2012 compliance

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0102: Data type of loop control variables to improve MISRA C:2012 compliance** |
| Description | To improve MISRA C:2012 compliance of generated code, use integer data type for variables that are used as loop control counter variables in:   * For loops constructed in Stateflow and MATLAB. * For Iterator blocks. |
| Rationale | Improve MISRA C:2012 compliance of the generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Code > Check data type of loop control variables**  For check details, see Check data type of loop control variables. |
| References | * ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Rule 14.1 |
| Last Changed | R2018a |

Stateflow Chart Considerations

* hisf\_0065: Type cast operations in Stateflow to improve code compliance
* hisf\_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance

hisf\_0065: Type cast operations in Stateflow to improve code compliance

|  |  |
| --- | --- |
| **ID: Title** | **hisf\_0065: Type cast operations in Stateflow to improve code compliance** |
| Description | In Stateflow charts that use the C action language, use the := notation to protect against Stateflow casting integer and fixed-point calculations to wider data types than the input data types. |
| Notes | If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the coding standards. |
| Rationale | To avoid implicit casts in the generated code that might violate coding standards. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check assignment operations in Stateflow charts**  For check details, see Check assignment operations in Stateflow charts. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 6 (1g) 'No implicit type conversions' * MISRA C:2012, Rule 10.1 MISRA C:2012, Rule 12.2 |
| Prerequisites | hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance |
| Last Changed | R2021a |

hisf\_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance** | |
| Description | To improve code compliance of the generated code: | |
| A | Do not use unary minus operators on unsigned data types |
| Notes | The MATLAB and C action languages do not restrict the use of unary minus operators on unsigned expressions. | |
| Rationale | A | Improve code compliance of the generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Stateflow > Check Stateflow charts for unary operators**  For check details, see Check Stateflow charts for unary operators. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 10.1 | |
| Last Changed | R2017b | |

## MathWorks Advisory Board Guidelines

The MathWorks Advisory Board (MAB) Guidelines promote best practices for developing high-quality models of embedded software designs.

*<The MathWorks Advisory Board (MAB) Guidelines support the readability, reusability, and overall quality of Simulink models. Consider selecting dedicated MAB Guidelines for this Chapter.>*

The Software Modeling Guidelines given in this document only include a very selective subset of MAB Guidelines.

### Simulink

* jc\_0602: Consistency in model element names
* jc\_0281: Trigger signal names
* db\_0143: Usable block types in model hierarchy

jc\_0602: Consistency in model element names

|  |  |  |  |
| --- | --- | --- | --- |
| **ID: Title** | | **jc\_0602: Consistency in model element names** | |
| **Sub ID Recommendations** | | * NA-MAAB — No recommendations * JMAAB — a | |
| **MATLAB Versions** | | All | |
| **Sub ID** | **Rule** | | **Custom Parameter** |
| a | These names shall match when they are directly connected by using signal lines.   * Inport block name * Outport block name * Structural subsystem input port label name * Structural subsystem output port label name * From tag name * Goto tag name * Signal line signal name   **Exceptions**  A signal line that connects to one of the following subsystem types can have a name that differs from that of the subsystem port label:   * Subsystems linked to a library * Reusable subsystems   When a combination of Inport, Outport, and other blocks have the same name, use a suffix or prefix for the Inport and Outport blocks. Any prefix or suffix can be used for ports, but they must be consistent. For example, the Inport block uses “in” and Outport block uses “out”.  **Note**  Inport and Outport blocks must have different names and signal names. | | Not Applicable |
| **Example — Correct**  Names of model elements that connect directly to signal lines are consistent.    **Example — Incorrect**  Inconsistent names for model elements that connect directly to signal lines. | | |
| **Sub ID** | **Rationale** | | |
| a | * Prevent misconnected signal lines. * Readability is impaired. * Deviation from the rule can make it difficult to maintain the integrity of the model and code. | | |
| **Model Advisor Checks** | | Modeling Standards for MAB > Simulink > Diagram Appearance > Check for consistency in model element names | |
| **See Also** | | * Manage Signal Lines (Simulink) * Types of Subsystems (Simulink) | |
| **Last Changed** | | R2020a | |

jc\_0281: Trigger signal names

|  |  |  |  |
| --- | --- | --- | --- |
| **ID: Title** | | **jc\_0281: Trigger signal names** | |
| **Sub ID Recommendations** | | * NA-MAAB — No recommendations * JMAAB — a1/a2/a3/a4, b1/b2/b3/b4 | |
| **MATLAB Versions** | | All | |
| **Sub ID** | **Rule** | | **Custom Parameter** |
| a1 | The name of the conditional input block at the destination shall include the name of the block at the origin of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| a2 | The name of the conditional subsystem at the destination shall include the name of the block at the origin of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| a3 | The name of the conditional input block at the destination shall include the name of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| a4 | The name of the conditional subsystem at the destination shall include the name of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| b1 | The name of the Stateflow® block event at the destination shall include the name of the block at the origin of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| b2 | The name of Stateflow Chart at the destination shall include the name of the block at the origin of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| b3 | The name of the Stateflow block event at the destination shall include the name of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| b4 | The name of the trigger signal and the Stateflow Chart name at the destination must include the same name. The name of the Chart block at the destination shall include the name of the trigger signal. | | Not Applicable |
| **Example — Correct**    **Example — Incorrect** | | |
| **Sub ID** | **Rationale** | | |
| a1, a2, a3, a4, b1, b2, b3, b4 | * Reduces connection mistakes. * Increases understanding of the relationship between the origin of the trigger signal and the destination. | | |
| **Model Advisor Checks** | | Modeling Standards for MAB > Simulink > Diagram Appearance > Check trigger signal names | |
| **See Also** | | * Sub ID a1, a2, a3, a4, see MISRA AC SLSF 026C * Signal Basics (Simulink) * Use Events to Execute Charts (Stateflow) * Types of Subsystems (Simulink) | |
| **Last Changed** | | R2020a | |

db\_0143: Usable block types in model hierarchy

|  |  |  |  |
| --- | --- | --- | --- |
| **ID: Title** | | **db\_0143: Usable block types in model hierarchy** | |
| **Sub ID Recommendations** | | * NA-MAAB — a * JMAAB — a | |
| **MATLAB Versions** | | All | |
| **Sub ID** | **Rule** | | **Custom Parameter** |
| a | Model levels shall use only the block types that are defined for the layer type. Clearly defined layer types restrict the number of blocks that can be used.  Block restrictions:   * (R2011a and earlier) Enable block cannot be used at the root level of the model. * Action ports are not permitted at the root level of a model.   Layer restrictions:   * Data flow layers that are used for basic blocks only. * Other than data flow layers, layers can include blocks that are used for structural subsystems and all other layers.   Blocks that can be used for all layers include:   * Inport * Outport * Mux * Demux * Bus Selector * Bus Creator * Selector * Ground * Terminator * From * Goto * Merge * Unit Delay * Rate Transition * Data Type Conversion * Data Store Memory * If * Switch Case * Function-Call Generator * Function-Call Split * Inport Shadow | | Layer type  Block type |
| **Sub ID** | **Rationale** | | |
| a | * Readability is impaired when subsystems and basic blocks are used in the same layer. | | |
| **Model Advisor Checks** | | Modeling Standards for MAB > Simulink > Diagram Appearance > Check for mixing basic blocks and subsystems | |
| **See Also** | |  | |
| **Last Changed** | | R2020a | |

### Stateflow

* db\_0125: Stateflow local data
* db\_0137: States in state machines

db\_0125: Stateflow local data

|  |  |  |  |
| --- | --- | --- | --- |
| **ID: Title** | | **db\_0125: Stateflow local data** | |
| **Sub ID Recommendations** | | * NA-MAAB — a, b, c, d * JMAAB — a, b, c, d | |
| **MATLAB Versions** | | All | |
| **Sub ID** | **Rule** | | **Custom Parameter** |
| a | Data objects shall not be defined with **Scope** set to Local at the machine level. | | Not Applicable |
| **Example — Correct**      **Example — Incorrect**  **Scope** has set Local local data at the machine level. | | |
| b | Data objects shall not be defined with **Scope** set to Constant at the machine level. | | Not Applicable |
| **Example — Correct**      **Example — Incorrect**  **Scope** has set Constant local data at the machine level. | | |
| c | Data objects shall not be defined with **Scope** set to Parameter at the machine level. | | Not Applicable |
| **Example — Correct**      **Example — Incorrect**  **Scope** has set Parameter local data at the machine level. | | |
| d | A Stateflow® block with parent-child relationships shall not include local data with the same name. | | Not Applicable |
| **Example — Correct**      **Example — Incorrect**  A Stateflow block with parent-child relationships has local data with the same name. | | |
| **Sub ID** | **Rationale** | | |
| a | * When local data is defined at the machine level, it is shared with all blocks in the model. The data will not behave like a local variable and can be influenced by any operation. * Adherence to the rules prevent the definition from disappearing when copying a Stateflow block to another model. | | |
| b, c | * Adherence to the rules prevent the definition from disappearing when copying a Stateflow block to another model. | | |
| d | * When a Stateflow block with parent-child relationships includes local data with the same name, readability decreases due to lack of clarity with regard to the influence of the local data. | | |
| **Model Advisor Checks** | | Modeling Standards for MAB > Stateflow > Block/Data/Events > Check definition of Stateflow data | |
| **See Also** | | * Stateflow Data Properties (Stateflow) * Use the Model Explorer with Stateflow Objects (Stateflow) * State Hierarchy (Stateflow) | |
| **Last Changed** | | R2020a | |

db\_0137: States in state machines

|  |  |  |  |
| --- | --- | --- | --- |
| **ID: Title** | | **db\_0137: States in state machines** | |
| **Sub ID Recommendations** | | * NA-MAAB — a * JMAAB — a | |
| **MATLAB Versions** | | All | |
| **Sub ID** | **Rule** | | **Custom Parameter** |
| a | When the **Decomposition** for the Chart block or State is set to OR (Exclusive), there shall be at least two states in the hierarchy. | | Not Applicable |
| **Example — Incorrect**  The hierarchy contains only one state when the **Decomposition** option is set to OR (Exclusive). | | |
| **Sub ID** | **Rationale** | | |
| a | * Redundant descriptions impair readability. * Generated code includes unnecessary state variables. | | |
| **Model Advisor Checks** | | Modeling Standards for MAB > Stateflow > Diagram > Check for exclusive states in state machines | |
| **See Also** | | * States (Stateflow) * State Decomposition (Stateflow) | |
| **Last Changed** | | R2020a | |

## Company-Specific Guidelines

*<Insert your own additional guidelines here. It is recommended to specify an additional project specific subset of MATLAB, Simulink, and Stateflow elements and settings to be used for design models.>*

# Constraints on Modeling Tools

This section describes the constraints on the use of the modeling tools MATLAB, Simulink, and Stateflow.

*<*The following guidelines provide a starting point for defining constraints on the use of modeling tools within your project. The guidelines are available at the [*MathWorks Documentation Center*](https://www.mathworks.com/help)*.* Tailor the following set of guidelines based on your planned use of the tools and applicability to your project.>

## Modeling Guidelines for High-Integrity Systems

*<Use the Simulink Modeling Guidelines for High-Integrity Systems when defining your own subset of guidelines. The guidelines are available at the* [*MathWorks Documentation Center*](https://www.mathworks.com/help)*. For this chapter, consider listing the following types of High-Integrity Guidelines:*

* *Configuration Parameter Considerations*
* *MISRA C:2012 Compliance Considerations: Configuration Settings>*

### Configuration Parameter Considerations

Solver

* hisl\_0040: Configuration Parameters > Solver > Simulation time
* hisl\_0041: Configuration Parameters > Solver > Solver options
* hisl\_0042: Configuration Parameters > Solver > Tasking and sample time options

hisl\_0040: Configuration Parameters > Solver > Simulation time

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0040: Configuration Parameters > Solver > Simulation time** | |
| Description | Set these simulation time configuration parameters 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)**. |
| Notes | Simulink allows nonzero start times for simulation. However, production code generation requires a zero start time.  **Stop time** in seconds and **Application lifespan (days)** is in days.  When configuration parameter **Application lifespan (days)** is set to auto (default), any positive value for **Stop time** is valid. | |
| Rationale | A, B | Generate code that is valid for production code generation. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related solver settings for simulation time**  For check details, see Check safety-related solver settings for simulation time. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' | |
| See Also | * hisl\_0048: Configuration Parameters > Math and Data Types > Application lifespan (days) * Solver Pane (Simulink) | |
| Last Changed | R2017b | |

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). |
| Notes | Generating code for production requires a fixed-step, discrete solver. | |
| Rationale | A, B | Generate code that is valid for production code generation. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related solver settings for solver options**  For check details, see Check safety-related solver settings for solver options. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' | |
| See Also | Solver Pane (Simulink) | |
| Last Changed | R2017b | |

hisl\_0042: Configuration Parameters > Solver > Tasking and sample time options

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0042: Configuration Parameters > Solver > Tasking and sample time options** |
| Description | Clear configuration parameter **Automatically handle rate transition for data transfer**. |
| Notes | Selecting the **Automatically handle rate transition for data transfer** 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 | Support fully specified models and unambiguous code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related solver settings for tasking and sample-time**  For check details, see Check safety-related solver settings for tasking and sample-time. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | Solver Pane (Simulink) |
| Last Changed | R2018a |

Math and Data Types

* hisl\_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)
* hisl\_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)

hisl\_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)** |
| Description | To support unambiguous behavior when using logical operators, relational operators, and the Combinatorial Logic block, select Configuration Parameter **Implement logic signals as Boolean data (vs. double)**. |
| 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 | Avoid ambiguous model behavior and optimize memory for generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related optimization settings for logic signals**  For check details, see Check safety-related optimization settings for logic signals. |
| References | * ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' * MISRA C:2012, Rule 10.1 |
| See Also | Implement logic signals as Boolean data (vs. double) (Simulink) |
| Last Changed | R2018b |

hisl\_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)** |
| Description | To support the robustness of systems that run continuously, set Configuration Parameter **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 | Support robustness of systems that run continuously. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related optimization settings for application lifespan**  For check details, see Check safety-related optimization settings for application lifespan. |
| References | * ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| See Also | * Application lifespan (days) (Simulink) * hisl\_0040: Configuration Parameters > Solver > Simulation time |
| Last Changed | R2018b |

Diagnostics

* hisl\_0036: Configuration Parameters > Diagnostics > Saving
* hisl\_0043: Configuration Parameters > Diagnostics > Solver
* hisl\_0044: Configuration Parameters > Diagnostics > Sample Time
* hisl\_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants
* hisl\_0301: Configuration Parameters > Diagnostics > Compatibility
* hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters
* hisl\_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block
* hisl\_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization
* hisl\_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging
* hisl\_0306: Configuration Parameters > Diagnostics > Connectivity > Signals
* hisl\_0307: Configuration Parameters > Diagnostics > Connectivity > Buses
* hisl\_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls
* hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion
* hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing
* hisl\_0311: Configuration Parameters > Diagnostics > Stateflow
* hisl\_0314: Configuration Parameters > Diagnostics > Data Validity > Signals

hisl\_0036: Configuration Parameters > Diagnostics > Saving

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0036: Configuration Parameters > Diagnostics > Saving** |
| Description | Set these configuration parameters to error:   * **Block diagram contains disabled library links** * **Block diagram contains parameterized library links** |
| Rationale | Prevent unexpected results. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety- related diagnostic settings for saving**  For check details, see Check safety-related diagnostic settings for saving. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' |
| See Also | Model Configuration Parameters: Diagnostics (Simulink) |
| Last Changed | R2021a |

hisl\_0043: Configuration Parameters > Diagnostics > Solver

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0043: Configuration Parameters > Diagnostics > Solver** | |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics** pane, set the **Solver** parameters as follows:   * **Algebraic loop** to error. * **Minimize algebraic loop** to error. * **Block priority violation** to error if you are using block priorities. * **Automatic solver parameter selection** to error. * **State name clash** to warning. | |
| Notes | Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.  This table clarifies the result of not specifying the configuration parameter as indicated above. | |
| **If Diagnostic Parameter ...** | **Is Not Set As Indicated, Then ...** |
| **Algebraic loop** | Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution. |
| **Minimize algebraic loop** | Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution. |
| **Block priority violation** | Block execution order can include undetected conflicts that might result in unpredictable block order execution. |
| **Automatic solver parameter selection** | An automatic change to the solver, step size, or simulation stop time can go undetected and might the operation of generated code. |
| **State name clash** | A name being used for more than one state might go undetected. |
| Rationale | Support generation of robust and unambiguous code. | |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for solvers**  For check details, see Check safety-related diagnostic settings for solvers. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' | |
| See Also | * Model Configuration Parameters: Diagnostics (Simulink) * jc\_0021: Model diagnostic settings (Simulink) | |
| Last Changed | R2018b | |

hisl\_0044: Configuration Parameters > Diagnostics > Sample Time

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0044: Configuration Parameters > Diagnostics > Sample Time** | |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics > Sample Time** pane, set the following **Sample Time** parameters to error:   * **Source block specifies -1 sample time** * **Multitask rate transfer** * **Single task rate transfer** * **Multitask conditionally executed subsystem** * **Tasks with equal priority** * **Enforce sample times specified by Signal Specification blocks** * **Unspecified inheritability of sample times**   If the target system does not allow preemption between tasks that have equal priority, set **Tasks with equal priority** to none. | |
| Notes | Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.  This table clarifies the result of not specifying the configuration parameter as indicated above. | |
| **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. |
| **Multitask rate transfer** | 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 transfer** | 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. |
| **Unspecified inheritability of sample times** | An S-function that is not explicitly set to inherit sample time can go undetected and result in unpredictable behavior. |
| Rationale | Support generation of robust and unambiguous code. | |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for sample time**  For check details, see Check safety-related diagnostic settings for sample time. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1i) 'Concurrency aspects' | |
| See Also | Model Configuration Parameters: Sample Time Diagnostics (Simulink) | |
| Last Changed | R2017b | |

hisl\_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants** |
| Description | Set these configuration parameters to error:   * **Arithmetic operations in variant conditions** * **Variant condition mismatch at signal source and destination** |
| Rationale | To maintain a consistent behavior between the simulation and generated code and to prevent the creation of unused variables in generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check safety-related diagnostic settings for variants**  For check details, see Check safety-related diagnostic settings for variants. |
| References | * ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' * MISRA C:2012, Rule 2.2 |
| Last Changed | R2021b |

hisl\_0301: Configuration Parameters > Diagnostics > Compatibility

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0301: Configuration Parameters > Diagnostics > Compatibility** |
| Description | Set configuration parameter **S-function upgrades needed** to error. |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for compatibility**  For check details, see Check safety-related diagnostic settings for compatibility. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| See Also | Model Configuration Parameters: Compatibility Diagnostics (Simulink) |
| Last Changed | R2017b |

hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters** |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set the **Parameters** parameters as follows:   * **Detect downcast** to error * **Detect underflow** to error * **Detect loss of tunability** to error * **Detect overflow** to error * **Detect precision loss** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for parameters**  For check details, see Check safety-related diagnostic settings for parameters. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| See Also | Model Configuration Parameters: Data Validity Diagnostics (Simulink) |
| Last Changed | R2018b |

hisl\_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0303: Configuration Parameters > Diagnostics > Data Validity > Merge block** |
| Description | Set configuration parameter **Detect multiple driving blocks executing at the same time step** to error. |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for Merge blocks**  For check details, see Check safety-related diagnostic settings for Merge blocks. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | Detect multiple driving blocks executing at the same time step (Simulink) |
| Last Changed | R2017b |

hisl\_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0304: Configuration Parameters > Diagnostics > Data Validity > Model Initialization** |
| Description | Set configuration parameter **Underspecified initialization** to Simplified. |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for model initialization**  For check details, see Check safety-related diagnostic settings for model initialization. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' * MISRA C:2012, Rule 9.1 |
| See Also | Underspecified initialization detection (Simulink) |
| Last Changed | R2017b |

hisl\_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging** |
| Description | Set configuration parameter **Model Verification block enabling** to Disable all. |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for data used for debugging**  For check details, see Check safety-related diagnostic settings for data used for debugging. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | Model Verification block enabling (Simulink) |
| Last Changed | R2017b |

hisl\_0306: Configuration Parameters > Diagnostics > Connectivity > Signals

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0306: Configuration Parameters > Diagnostics > Connectivity > Signals** |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics > Connectivity** pane, set the **Signals** parameters as follows:   * **Signal label mismatch** to error * **Unconnected block input ports** to error * **Unconnected block output ports** to error * **Unconnected line** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for signal connectivity**  For check details, see Check safety-related diagnostic settings for signal connectivity. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' |
| See Also | Model Configuration Parameters: Connectivity Diagnostics (Simulink) |
| Last Changed | R2017b |

hisl\_0307: Configuration Parameters > Diagnostics > Connectivity > Buses

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0307: Configuration Parameters > Diagnostics > Connectivity > Buses** |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics > Connectivity** pane, set the **Buses** parameters as follows:   * **Unspecified bus object at root Outport block** to error * **Element name mismatch** to error * **Bus signal treated as vector** to error * **Non-bus signals treated as bus signals** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for bus connectivity**  For check details, see Check safety-related diagnostic settings for bus connectivity. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | Model Configuration Parameters: Connectivity Diagnostics (Simulink) |
| Last Changed | R2020a |

hisl\_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls** |
| Description | Set configuration parameter **Context-dependent inputs** to error. |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings that apply to function-call connectivity**  For check details, see Check safety-related diagnostic settings that apply to function-call connectivity. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | Model Configuration Parameters: Connectivity Diagnostics (Simulink) |
| Last Changed | R2017b |

hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion** |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics > Type Conversion** pane, set these parameters as follows:   * **Unnecessary type conversion** to warning * **Vector/matrix block input conversion** to error * **32-bit integer to single precision float conversion** to warning |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for type conversions**  For check details, see Check safety-related diagnostic settings for type conversions. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| See Also | Model Configuration Parameters: Type Conversion Diagnostics (Simulink) |
| Last Changed | R2017b |

hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing** |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics > Model Referencing** pane, set these parameters as follows:   * **Port and parameter mismatch** to error * **Invalid root Inport/Outport block connection** to error * **Unsupported data logging** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for model referencing**  For check details, see Check safety-related diagnostic settings for model referencing. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | Model Configuration Parameters: Model Referencing Diagnostics (Simulink) |
| Last Changed | R2020a |

hisl\_0311: Configuration Parameters > Diagnostics > Stateflow

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0311: Configuration Parameters > Diagnostics > Stateflow** |
| Description | On the **Diagnostics > Stateflow** pane, set these configuration parameters to error:   * **Unexpected backtracking** * **Invalid input data access in chart initialization** * **No unconditional default transitions** * **Transitions outside natural parent** * **Undirected event broadcasts** * **Transition action specified before condition action** * **Read-before-write to output in Moore chart** * **Absolute time temporal value shorter than sampling period** * **Self transition on leaf state** * **Execute-at-Initialization disabled in presence of input events** * **Use of machine-parented data instead of Data Store Memory** * **Unreachable execution path** |
| Rationale | Improve robustness of design and promote a clear modeling style. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for Stateflow**  For check details, see Check safety-related diagnostic settings for Stateflow. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 6 (1j) 'No recursions' * MISRA C:2012, Rule 17.2 |
| See Also | Model Configuration Parameters: Stateflow Diagnostics (Simulink) |
| Last Changed | R2021a |

hisl\_0314: Configuration Parameters > Diagnostics > Data Validity > Signals

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0314: Configuration Parameters > Diagnostics > Data Validity > Signals** |
| Description | In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set the **Signals** parameters as follows:   * **Signal resolution** to Explicit only * **Division by singular matrix** to error * **Underspecified data types** to error * **Inf or NaN block output** to error * **"rt" prefix for identifiers** to error * **Wrap on overflow** to error * **Saturate on overflow** to error * **Simulation range checking** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related diagnostic settings for signal data**  For check details, see Check safety-related diagnostic settings for signal data. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 |
| See Also | Model Configuration Parameters: Data Validity Diagnostics (Simulink) |
| Last Changed | R2018a |

Hardware Implementation

* hisl\_0071: Configuration Parameters > Model Referencing

hisl\_0071: Configuration Parameters > Hardware Implementation > Inconsistent hardware implementation settings

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0071: Configuration Parameters > Hardware Implementation > Inconsistent hardware implementation settings** |
| Description | Inconsistencies or under-specification of hardware attributes can result in incompatible code generation for production hardware. For compatible code generation, these configuration parameters must be the same between production hardware and test hardware:   * **Byte ordering** * **Signed integer division rounds to** |
| Notes | Simulink and Simulink Coder require two sets of target specifications. The first set describes the final intended production target. The second set describes the currently selected target. If the configuration parameters do not match, the code generator creates extra code to emulate the behavior of the production target.  Inconsistent hardware parameters between production hardware and test hardware can be avoided by selecting configuration parameter **Test hardware is the same as production hardware.** |
| Rationale | Efficient code generation |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check the hardware implementation**  For check details, see Check safety-related settings for hardware implementation. |
| References |  |
| See Also | Set Byte Ordering for Device (Simulink Coder) |
| Last Changed | R2021a |

Model Referencing

* hisl\_0037: Configuration Parameters > Model Referencing

hisl\_0037: Configuration Parameters > Model Referencing

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0037: Configuration Parameters > Model Referencing** | |
| Description | Set these Configuration Parameters as follows: | |
| A | **Rebuild** to either Never or If any changes detected. |
| B | **Never rebuild diagnostic** to Error if rebuild required. |
| C | Clear **Pass fixed-size scalar root inputs by value for code generation**. |
| D | Clear **Minimize algebraic loop occurrences**. |
| Rationale | A | To prevent unnecessary regeneration of the code, resulting in changing only the date of the file and slowing down the build process when using model references. |
| B | For safety-related applications, an error should alert model developers that the parent and referenced models are inconsistent. |
| C | To prevent unpredictable data because scalar values can change during a time step. |
| D | To be compatible with the recommended setting of **Single output/update function** for embedded systems code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related model referencing settings**  For check details, see Check safety-related model referencing settings. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' | |
| See Also | Model Configuration Parameters: Model Referencing (Simulink) | |
| Last Changed | R2021a | |

Simulation Target

* hisl\_0046: Configuration Parameters > Code Generation > Optimization > Block reduction

hisl\_0046: Configuration Parameters > Code Generation > Optimization > Block reduction

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0046: Configuration Parameters > Code Generation > Optimization > Block reduction** |
| Description | To support unambiguous presentation of the generated code and support traceability between a model and generated code, clear configuration parameter **Block reduction**. |
| 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 | Supports:   * Unambiguous presentation of generated code * Traceability between a model and generated code |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related block reduction optimization settings**  For check details, see Check safety-related block reduction optimization settings. |
| References | * ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' |
| See Also | Block reduction (Simulink) |
| Last Changed | R2018b |

Code Generation

* hisl\_0052: Configuration Parameters > Code Generation > Optimization > Data initialization
* hisl\_0053: Configuration Parameters > Code Generation > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values
* hisl\_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions
* hisl\_0056: Configuration Parameters > Code Generation > Optimization > Optimize using the specified minimum and maximum values
* hisl\_0038: Configuration Parameters > Code Generation > Comments
* hisl\_0039: Configuration Parameters > Code Generation > Interface
* hisl\_0047: Configuration Parameters > Code Generation > Code Style
* hisl\_0049: Configuration Parameters > Code Generation > Identifiers

hisl\_0052: Configuration Parameters > Code Generation > Optimization > Data initialization

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0052: Configuration Parameters > Code Generation > Optimization > Data initialization** | |
| Description | To support complete definition of data and initialize internal and external data to zero, clear these configuration parameters: | |
| A | **Remove root level I/O zero initialization**. |
| B | **Remove internal data zero initialization**. |
| Notes | 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.  The configuration parameters are applicable only when these **Code Generation** configuration parameters are set as follows:   * **System target file** is an ERT-based target. * **Interface > Code interface packaging** is set to either Nonreusable function or Reusable function. | |
| Rationale | A, B | Support fully defined data in generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related optimization settings for data initialization**  For check details, see Check safety-related optimization settings for data initialization. | |
| References | * ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 6 (1c) 'Initialization of variables' | |
| See Also | * Remove root level I/O zero initialization (Embedded Coder) * Remove internal data zero initialization (Embedded Coder) | |
| Last Changed | R2021a | |

hisl\_0053: Configuration Parameters > Code Generation > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0053: Configuration Parameters > Code Generation > Optimization > Remove code from floating- point to integer conversions that wraps out-of-range values** |
| Description | To support verifiable code, select configuration parameter **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 parameter **Saturate on integer overflow** cleared, clearing configuration parameter **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 | Support generation of code that can be verified. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related optimization settings for data type conversions**  For check details, see Check safety-related optimization settings for data type conversions. |
| References | * ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Rule 2.1 |
| See Also | Remove code from floating-point to integer conversions that wraps out-of- range values (Simulink Coder) |
| Last Changed | R2021b |

hisl\_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions** |
| Description | To support the robustness of the operations, clear configuration parameter **Remove code that protects against division arithmetic exceptions**. |
| Notes | 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.  This configuration parameter is applicable only when the **System target file** is an ERT-based target. |
| Rationale | Protect against divide-by-zero exceptions for fixed-point code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related optimization settings for division arithmetic exceptions**  For check details, see Check safety-related optimization settings for division arithmetic exceptions. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' * MISRA C:2012, Dir 4.1 |
| See Also | Remove code that protects against division arithmetic exceptions (Embedded Coder) |
| Last Changed | R2021b |

hisl\_0056: Configuration Parameters > Code Generation > Optimization > Optimize using the specified minimum and maximum values

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0056: Configuration Parameters > Code Generation > Optimization > Optimize using the specified minimum and maximum values** |
| Description | To support verifiable code, clear configuration parameter **Optimize using the specified minimum and maximum values**. |
| Notes | Selecting **Optimize using the specified minimum and maximum values** can result in requirements without associated code and violates traceability objectives. |
| Rationale | Support traceability between a model and generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related optimization settings for specified minimum and maximum values**  For check details, see Check safety-related optimization settings for specified minimum and maximum values. |
| References | * ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' |
| See Also | Optimize using the specified minimum and maximum values (Embedded Coder) |
| Last Changed | R2018b |

hisl\_0038: Configuration Parameters > Code Generation > Comments

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0038: Configuration Parameters > Code Generation > Comments** | |
| Description | In the Configuration Parameters dialog box, on the **Code Generation > Comments** pane, select these parameters: | |
| A | **Include comments**. |
| B | **Simulink block comments**. |
| C | **Show eliminated blocks**. |
| D | **Verbose comments for 'Model default' storage class**. |
| E | **Requirements in block comments**. |
| Rationale | A | Including comments provide good traceability between the code and the model. |
| B | Including comments that describe the code for blocks provide good traceability between the code and the model. |
| C | Including comments that describe the code for blocks eliminated from a model provide good traceability between the code and the model. |
| D | Including the names of parameter variables and source blocks as comments in the model parameter structure declaration in *model*\_prm.h provides good traceability between the code and the model. |
| E | Including requirement descriptions assigned to Simulink blocks as comments provide good traceability between the code and the model. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related code generation settings for comments**  For check details, see Check safety-related code generation settings for comments. | |
| References | * ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' | |
| See Also | Model Configuration Parameters: Comments (Embedded Coder) | |
| Last Changed | R2021a | |

hisl\_0039: Configuration Parameters > Code Generation > Interface

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0039: Configuration Parameters > Code Generation > Interface** | |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, set the **Software environment**, **Code interface**, and **Data exchange interface** parameters as follows: | |
| A | Clear **Support: non-finite numbers**. |
| B | Clear **Support: absolute time**. |
| C | Clear **Support: continuous time**. |
| D | Clear **Support: non-inlined S-functions**. |
| E | Clear **Classic call interface**. |
| F | Select **Single output/update function**. |
| G | Clear **Terminate function required**. |
| H | Select **Remove error status field in real-time model data structure**. |
| I | Clear **MAT-file logging**. |
| Rationale | A | Support for nonfinite numbers is inappropriate for real-time safety-related systems. |
| B | Support for absolute time is inappropriate for real-time safety-related systems. |
| C | Support for continuous time is inappropriate for real-time safety-related systems. |
| D | Support for noninlined S-functions requires support of nonfinite numbers, which is inappropriate for real-time safety-related systems. |
| E | To eliminate model function calls compatible with the main program module of the pre-2012a GRT target that is inappropriate for real-time safety-related systems. |
| F | To simplify the interface to the real-time operating system (RTOS) and simplify verification of the generated code by creating a single call to both the output and update functions. |
| G | To eliminate *model*\_terminate function used for deallocating dynamic memory that is unsuitable for real-time safety-related systems. |
| H | To eliminate extra code for logging and monitoring error status that might not be reachable for testing. |
| I | To eliminate extra code for logging test points to a MAT file that is not supported by embedded targets. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related code generation interface settings**  For check details, see Check safety-related code generation interface settings. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' | |
| See Also | Model Configuration Parameters: Code Generation Interface (Embedded Coder) | |
| Last Changed | R2021a | |

hisl\_0047: Configuration Parameters > Code Generation > Code Style

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0047: Configuration Parameters > Code Generation > Code Style** | |
| Description | In the Configuration Parameters dialog box, on the **Code Generation > Code Style** pane, set these parameters: | |
| A | Set **Parenthesis level** to Maximum (Specify precedence with parenthesis). |
| B | Select **Preserve operand order in expression**. |
| Rationale | A | To prevent unexpected results. |
| B | To improve traceability of the generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related code generation settings for code style**  For check details, see Check safety-related code generation settings for code style. | |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' * MISRA C:2012, Rule 12.1 | |
| See Also | Model Configuration Parameters: Code Style (Embedded Coder) | |
| Last Changed | R2019b | |

hisl\_0049: Configuration Parameters > Code Generation > Identifiers

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0049: Configuration Parameters > Code Generation > Identifiers** |
| Description | To minimize the likelihood that parameter and signal names will change during code generation when the model changes, set configuration parameter **Minimal mangle length** to 4 or greater. |
| Rationale | Decrease the effort to perform code review. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Configuration > Check safety-related code generation identifier settings**  For check details, see Check safety-related code generation identifier settings. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' |
| See Also | Model Configuration Parameters: Code Generation Identifies (Embedded Coder) |
| Last Changed | R2021a |

### MISRA C:2012 Compliance Considerations

Configuration Settings

* hisl\_0060: Configuration Parameters that improves MISRA C:2012 compliance

hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance** |
| Description | Set these model configuration parameters as specified:   * **System target file** as an ERT-based target. * **Use division for fixed-point net slope computation to** On or Use division for reciprocals of integers only. * **Inf or NaN block output to** warning or error. * **Model Verification block enabling to** Disable all. * **Undirected event broadcasts to** error. * **Wrap on overflow to** warning or error. * **Production hardware signed integer division rounds to to** Zero or Floor. * **Compile-time recursion limit for MATLAB functions** to 0. * **Casting modes to** Standards Compliant. * **Code replacement library to** None or AUTOSAR 4.0. * **Maximum identifier length to** the implementation dependent limit. The default is 31. * **Parentheses level to** Maximum (Specify precedence with parentheses). * **Shared code placement to** Shared location. * **Standard math library to** C89/C90 (ANSI) or C99 (ISO), depending on toolchain. * **Bitfield declarator type specifier to** uint\_T when any of these parameters are selected: * **Pack Boolean data into bitfields** * **Use bitsets for storing state configuration** * **Use bitsets for storing Boolean data**   Select (on) these configuration parameters:   * **Include comments** * **MATLAB user comments** * **Preserve static keyword in function declarations** (Select only when configuration parameter **File packaging format** is set to Compact or CompactWithDataFile.)   Deselect (off) these configuration parameters:   * **Shift right on a signed integer as arithmetic shift** * **Dynamic memory allocation in MATLAB functions** * **Enable run-time recursion for MATLAB functions** * **External mode** * **Generate shared constants** * **Mat-file logging** * **Replace multiplications by powers of two with signed bitwise shifts** * **Support complex numbers (Only if you do not need complex number support.)** * **Support continuous time** * **Support non-finite numbers** * **Support non-inlined S-functions** * **Use dynamic memory allocation for model initialization** (Keep this parameter selected only when configuration parameter **Code interface packaging** is set to Reusable function.) |
| Rationale | Improve MISRA C:2012 compliance of the generated code. |
| Model Advisor Checks | **Modeling Standards for ISO 26262 > High-Integrity Systems > Code > Check configuration parameters for MISRA C:2012**  For check details, see Check configuration parameters for MISRA C:2012. |
| References | * ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 6 (1b) 'No dynamic objects or variables, or else online test during their creation' |
| Last Changed | R2021b |

## Company-Specific Constraints

*<Insert your own additional constraints here. It is recommended to specify an additional project specific subset of settings to be used for design models (e.g. settings regarding the project specific target platform).>*

# Model Requirements and Traceability

This section describes the method to identify and delimit the requirements contained in the model and the means to establish traceability between requirements and other lifecycle data.

Design models contain the software architecture and unit designs. Software architecture and unit designs are expressed by the subset of MATLAB, Simulink, and Stateflow, as defined in [Chapter 3: Style Guidelines and Complexity Restrictions](#_Style_Guidelines_and). Model elements that do not express requirements are documented in [Chapter 7: Additional Model Elements](#_Additional_Model_Elements).

Use Simulink® Requirements™ to trace between the design model and the requirements the model was developed from. Single blocks or a combination of blocks can express and link to requirements.

Traceability between the design model and the generated code is established by comments contained in the generated source code. The comments provide links to the design model element.

To facilitate review, a traceability report is generated with the data between the design model and the requirements from which the model was developed.

*<Insert your strategy for establishing model requirements and traceability here.>*

# Derived Requirements

This section describes the method to identify and delimit the derived requirements contained in the model. It also describes the method to provide derived requirements to the system processes, including the system safety assessment process.

|  |  |
| --- | --- |
| **Note** | Derived requirements can be part of a design model or they can be treated like regular requirements outside of a design model. The following information refers to derived requirements contained in a design model. |

Derived requirements that are contained in a design model will be marked by *<Insert your strategy here>* to clearly identify them as derived requirements and to provide them to the safety assessment process. A combination of model review and model coverage analysis is used to verify that all derived requirements were properly marked and analyzed. Additional test cases will be created, based on the derived requirements contained in the design model. The additional test cases must be executed on the executable object code. An analysis of the results from model coverage analysis and structural coverage analysis will verify that all derived requirements are identified correctly.

|  |  |
| --- | --- |
| **Note** | You can create test cases for derived requirements using Simulink® Design Verifier™. |

# Additional Model Elements

This section describes the means to identify each model element that does not contribute to the representation of a software requirement or of the software architecture and is not an input to a subsequent software development process or activity. For example, a comment block.

In addition to [Chapter 3: Style Guidelines and Complexity Restrictions](#_Style_Guidelines_and), the following model elements are part of a design model, but will not be subject to code generation:

*<List all additional elements of your design model that are not subject to code generation in here (e.g. annotations, Model Info block, DocBlock and so on).>*

A model review will be performed to verify the correct use of the additional model elements and the absence of undefined elements.

# Suitability of Technique

This section provides a rationale for the suitability of the technique for the type of information expressed by a Design Model.

*<Adapt the following paragraph to your industry domain specific needs.>*

Block diagrams are a well-established way of describing flow and logic-based applications for control and monitoring systems. State charts and flow charts are a basic notation form in information technology and other industries.

MATLAB, Simulink, and Stateflow are long-time standard tools among a wide variety of industries. The tools documentation is available at the [MathWorks Documentation Center.](https://www.mathworks.com/help) People involved in a software development project can use the documentation to understand the content of a design model.

# Appendix A: Mapping of Modeling Guideline to ISO 26262-6

This section associates the Software Modeling Guidelines to the applicable software design principles documented in ISO 26262-6:2018.

## Topics to be Covered by Modeling and Coding Guidelines

Table 1 of ISO 26262-6:2018 provides a list of topics to be covered by modeling and coding guidelines. The Simulink Modeling Guidelines for High-Integrity Systems and the MathWorks Advisory Board (MAB) Guidelines cover many of these topics. The table below identifies the modeling guidelines that are relevant to each topic.

Keep in mind that the collection of guidelines you select should address a combination of topics that are applicable for the ASIL under consideration.

Mapping of Guidelines to Topics to be Covered by Modeling and Coding Guidelines

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Topic** | **Guideline ID** | | | |
| **High-Integrity Systems** | | | **MAB** |
| 1a | Enforcement of low complexity | hisl\_0024 |  | himl\_0003 himl\_0013 |  |
| 1b | Use of language subsets | hisl\_0001 hisl\_0002 hisl\_0003 hisl\_0004 hisl\_0005 hisl\_0006 hisl\_0007 hisl\_0008 hisl\_0010 hisl\_0011 hisl\_0012 hisl\_0013 hisl\_0015 hisl\_0016 hisl\_0017 hisl\_0018 hisl\_0019 hisl\_0020 hisl\_0021 hisl\_0022 hisl\_0028 hisl\_0029 hisl\_0033 hisl\_0034 hisl\_0036 hisl\_0037 hisl\_0039 hisl\_0040 hisl\_0041 hisl\_0042 hisl\_0043 hisl\_0044 hisl\_0047 hisl\_0049 hisl\_0054 hisl\_0060 hisl\_0061 hisl\_0062 hisl\_0066 hisl\_0067 hisl\_0072 hisl\_0073 hisl\_0301 hisl\_0302 hisl\_0303 hisl\_0304 hisl\_0305 hisl\_0306 hisl\_0307 hisl\_0308 hisl\_0309 hisl\_0310 hisl\_0311 hisl\_0314 | hisf\_0001 hisf\_0002 hisf\_0003 hisf\_0009 hisf\_0011 hisf\_0013 hisf\_0014 hisf\_0016 hisf\_0017 hisf\_0065 hisf\_0211 | himl\_0004 himl\_0006 himl\_0007 himl\_0008 himl\_0010 himl\_0011 himl\_0012 | db\_0125 db\_0143 |
| 1c | Enforcement of strong typing | hisl\_0016 hisl\_0017 hisl\_0018 hisl\_0019 hisl\_0022 hisl\_0024 hisl\_0025 hisl\_0026 hisl\_0045 hisl\_0061 hisl\_0073 hisl\_0074 hisl\_0102 hisl\_0309 | hisf\_0003 hisf\_0009 hisf\_0011 hisf\_0015 hisf\_0065 hisf\_0211 | himl\_0002 himl\_0008 himl\_0010 himl\_0011 |  |
| 1d | Use of defensive implementation techniques | hisl\_0001 hisl\_0002 hisl\_0003 hisl\_0004 hisl\_0005 hisl\_0006 hisl\_0007 hisl\_0008 hisl\_0010 hisl\_0011 hisl\_0012 hisl\_0013 hisl\_0015 hisl\_0019 hisl\_0022 hisl\_0028 hisl\_0029 hisl\_0033 hisl\_0034 hisl\_0037 hisl\_0048 hisl\_0052 hisl\_0053 hisl\_0054 hisl\_0056 hisl\_0061 hisl\_0066 hisl\_0067 hisl\_0072 hisl\_0073 hisl\_0301 hisl\_0302 hisl\_0309 hisl\_0314 | hisf\_0009 hisf\_0011 hisf\_0065 hisf\_0211 | himl\_0004 himl\_0006 himl\_0007 |  |
| 1e | Use of well-trusted design principles | hisl\_0021 hisl\_0023 hisl\_0038 hisl\_0061 hisl\_0074 | hisf\_0002 hisf\_0007 hisf\_0009 hisf\_0011 | himl\_0004 |  |
| 1f | Use of unambiguous graphical representation | hisl\_0021 hisl\_0024 hisl\_0036 hisl\_0044 hisl\_0046 hisl\_0061 hisl\_0062 hisl\_0074 hisl\_0306 | hisf\_0002 hisf\_0009 hisf\_0011 | himl\_0002 himl\_0003 himl\_0004 |  |
| 1g | Use of style guides | hisl\_0021 hisl\_0061 | hisf\_0002 hisf\_0009 hisf\_0011 | himl\_0001 himl\_0004 | db\_0137 |
| 1h | Use of naming conventions | hisl\_0031 hisl\_0032 hisl\_0061 hisl\_0062 | hisf\_0009 | himl\_0004 |  |
| 1i | Concurrency aspects | hisl\_0013 hisl\_0044 |  |  |  |

## Design Principles for Software Unit Design and Implementation

Table 6 of ISO 26262-6:2018 provides a list of design principles for software unit design and implementation. The Simulink Modeling Guidelines for High-Integrity Systems and the MathWorks Advisory Board (MAB) Guidelines cover many of these principles. The table below identifies the modeling guidelines that are relevant to each design principle.

Keep in mind that the collection of guidelines you select should address a combination of principles that are applicable for the ASIL under consideration.

Mapping of Guidelines to Design Principles for Software Unit Design and Implementation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Design Principle** | **Guideline ID** | | | |
| **High-Integrity Systems** | | | **MAB** |
| 1a | One entry and one exit point in subprograms and functions |  |  |  |  |
| 1b | No dynamic objects or variables, or else online test during their creation | hisl\_0060 |  |  |  |
| 1c | Initialization of variables | hisl\_0052 |  |  |  |
| 1d | No multiple use of variable names | hisl\_0061 hisl\_0063 |  |  |  |
| 1e | Avoid global variables or else justify their usage | hisl\_0062 |  |  |  |
| 1f | Restricted use of pointers |  |  |  |  |
| 1g | No implicit type conversions |  | hisf\_0065 | himl\_0008 |  |
| 1h | No hidden data flow or control flow | hisl\_0101 |  |  |  |
| 1i | No unconditional jumps |  |  |  |  |
| 1j | No recursions | hisl\_0311 | hisf\_0004 |  |  |