DO Qualification Kit

Software Model Standards (SMS)

R2017b

How to Contact MathWorks

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

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

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

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

Phone: 508-647-7000

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

DO Qualification Kit: Software Model Standards (SMS)

© COPYRIGHT 2014 – 2015 by The MathWorks, Inc.

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

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

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](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](http://www.mathworks.com/patents) for more information.

Revision History

March 2014 New for Version 2.3 (Applies to Release 2014a)

October 2014 Revised for Version 2.4 (Applies to Release 2014b)

March 2015 Revised for Version 2.5 (Applies to Release 2015a)

September 2015 Revised for DO Qualification Kit Version 3.0 (Applies to Release 2015b)

Document Title: **Software Model Standards for <*Project***>.

**Document Number:** **<*DocNo*>**

Revision: **<*Revision*>**

Project: **<*Project*>**

|  |  |  |
| --- | --- | --- |
| Approvals: |  |  |
| *<Name 1>*, Author |  | Date |
|  |  |  |
| *<Name 2>*, Project Management |  | Date |
|  |  |  |
| *<Name 3>*, Engineering |  | Date |
|  |  |  |
| *<Name 4>*, Quality Engineering |  | Date |

| Change History | | | | |
| --- | --- | --- | --- | --- |
| Rev. | Modification / Description | Date | Author | Checked |
| 1.0 | *First release of Software Model Standards (SMS)* |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[1 Introduction 1-1](#_Toc503436676)

[1.1 Purpose and Scope 1-2](#_Toc503436677)

[1.2 Applicable Documents 1-3](#_Toc503436678)

[Table 1 - Regulations and Standards 1-3](#_Toc503436679)

[Table 2 - Company and Project Plans, Standards, and Documents 1-3](#_Toc503436680)

[1.3 Referenced Documents 1-4](#_Toc503436681)

[Table 3 - Referenced Documents 1-4](#_Toc503436682)

[2 Methods, Tools and Modeling Languages 2-1](#_Toc503436683)

[Simulink 2-1](#_Toc503436684)

[Stateflow 2-1](#_Toc503436685)

[MATLAB 2-2](#_Toc503436686)

[3 Style Guidelines and Complexity Restrictions 3-1](#_Toc503436687)

[3.1 Modeling Guidelines for High-Integrity Systems 3-2](#_Toc503436688)

[Simulink Block Considerations – Naming Conventions 3-2](#_Toc503436689)

[hisl\_0032: Model object names 3-3](#_Toc503436690)

[Simulink Block Considerations – Math Operations 3-5](#_Toc503436691)

[hisl\_0001: Usage of Abs block 3-6](#_Toc503436692)

[hisl\_0002: Usage of Math Function blocks (rem and reciprocal) 3-7](#_Toc503436693)

[hisl\_0003: Usage of Square Root blocks 3-8](#_Toc503436694)

[hisl\_0028: Usage of Reciprocal Square Root blocks 3-9](#_Toc503436695)

[hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm) 3-10](#_Toc503436696)

[hisl\_0005: Usage of Product blocks 3-12](#_Toc503436697)

[hisl\_0029: Usage of Assignment blocks 3-13](#_Toc503436698)

[hisl\_0066: Usage of Gain blocks 3-15](#_Toc503436699)

[Simulink Block Considerations – Ports & Subsystems 3-16](#_Toc503436700)

[hisl\_0006: Usage of While Iterator blocks 3-17](#_Toc503436701)

[hisl\_0007: Usage of While Iterator subsystems 3-18](#_Toc503436702)

[hisl\_0008: Usage of For Iterator Blocks 3-19](#_Toc503436703)

[hisl\_0009: Usage of For Iterator Subsystem blocks 3-20](#_Toc503436704)

[hisl\_0010: Usage of If blocks and If Action Subsystem blocks 3-21](#_Toc503436705)

[hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks 3-23](#_Toc503436706)

[hisl\_0012: Usage of conditionally executed subsystems 3-24](#_Toc503436707)

[hisl\_0024: Inport interface definition 3-25](#_Toc503436708)

[hisl\_0025: Design min/max specification of input interfaces 3-26](#_Toc503436709)

[hisl\_0026: Design min/max specification of output interfaces 3-27](#_Toc503436710)

[Simulink Block Considerations – Signal Routing 3-28](#_Toc503436711)

[hisl\_0013: Usage of data store blocks 3-29](#_Toc503436712)

[hisl\_0015: Usage of Merge blocks 3-31](#_Toc503436713)

[hisl\_0021: Consistent vector indexing method 3-33](#_Toc503436714)

[hisl\_0022: Data type selection for index signals 3-34](#_Toc503436715)

[hisl\_0023: Verification of model and subsystem variants 3-35](#_Toc503436716)

[hisl\_0034: Usage of Signal Routing blocks 3-36](#_Toc503436717)

[Simulink Block Considerations – Logic and Bit Operations 3-37](#_Toc503436718)

[hisl\_0016: Usage of blocks that compute relational operators 3-38](#_Toc503436719)

[hisl\_0017: Usage of blocks that compute relational operators (2) 3-39](#_Toc503436720)

[hisl\_0018: Usage of Logical Operator block 3-40](#_Toc503436721)

[hisl\_0019: Usage of Bitwise Operator block 3-41](#_Toc503436722)

[Simulink Block Considerations – Lookup Table Blocks 3-42](#_Toc503436723)

[hisl\_0033: Usage of Lookup Table blocks 3-43](#_Toc503436724)

[Stateflow Chart Considerations – Chart Properties 3-44](#_Toc503436725)

[hisf\_0001: Mealy and Moore semantics 3-45](#_Toc503436726)

[hisf\_0002: User-specified state/transition execution order 3-46](#_Toc503436727)

[hisf\_0009: Strong data typing (Simulink and Stateflow boundary) 3-47](#_Toc503436728)

[hisf\_0011: Stateflow debugging settings 3-48](#_Toc503436729)

[Stateflow Chart Considerations – Chart Architecture 3-49](#_Toc503436730)

[hisf\_0003: Usage of bitwise operations 3-50](#_Toc503436731)

[hisf\_0004: Usage of recursive behavior 3-51](#_Toc503436732)

[hisf\_0007: Usage of junction conditions (maintaining mutual exclusion) 3-53](#_Toc503436733)

[hisf\_0013: Usage of transition paths (crossing parallel state boundaries) 3-54](#_Toc503436734)

[hisf\_0014: Usage of transition paths (passing through states) 3-56](#_Toc503436735)

[hisf\_0015: Strong data typing (casting variables and parameters in expressions) 3-57](#_Toc503436736)

[MATLAB Function and MATLAB Code Considerations – MATLAB Functions 3-58](#_Toc503436737)

[himl\_0001: Usage of standardized MATLAB function headers 3-59](#_Toc503436738)

[himl\_0002: Strong data typing at MATLAB function boundaries 3-60](#_Toc503436739)

[himl\_0003: Limitation of MATLAB function complexity 3-61](#_Toc503436740)

[himl\_0005: Usage of global variables in MATLAB functions 3-62](#_Toc503436741)

[MATLAB Function and MATLAB Code Considerations – MATLAB Code 3-64](#_Toc503436742)

[himl\_0004: MATLAB Code Analyzer recommendations for code generation 3-65](#_Toc503436743)

[himl\_0006: MATLAB code if / elseif / else patterns 3-67](#_Toc503436744)

[himl\_0007: MATLAB code switch / case / otherwise patterns 3-69](#_Toc503436745)

[himl\_0008: MATLAB code relational operator data types 3-71](#_Toc503436746)

[himl\_0009: MATLAB code with equal / not equal relational operators 3-72](#_Toc503436747)

[himl\_0010: MATLAB code with logical operators and functions 3-73](#_Toc503436748)

[MISRA C:2012 Compliance Considerations – Modeling Style 3-74](#_Toc503436749)

[hisl\_0061: Unique identifiers for clarity 3-75](#_Toc503436750)

[hisl\_0062: Global variables in graphical functions 3-78](#_Toc503436751)

[hisl\_0063: Length of user-defined object names to improve MISRA C:2012 compliance 3-80](#_Toc503436752)

[hisl\_0201: Define reserved keywords to improve MISRA C:2012 compliance 3-81](#_Toc503436753)

[MISRA C:2012 Compliance Considerations – Block Usage 3-82](#_Toc503436754)

[hisl\_0020: Blocks not recommended for C/C++ production code deployment and MISRA C:2012 compliance 3-83](#_Toc503436755)

[hisl\_0101: Avoid invariant comparison operations to improve MISRA C:2012 compliance 3-84](#_Toc503436756)

[hisl\_0102: Data type of loop control variables to improve MISRA C:2012 compliance 3-86](#_Toc503436757)

[MISRA C:2012 Compliance Considerations – Stateflow Chart Considerations 3-87](#_Toc503436758)

[hisf\_0064: Shift operations for Stateflow data to improve code compliance 3-88](#_Toc503436759)

[hisf\_0065: Type cast operations in Stateflow to improve code compliance 3-89](#_Toc503436760)

[hisf\_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance 3-90](#_Toc503436761)

[hisf\_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA C:2012 compliance 3-91](#_Toc503436762)

[3.2 Simulink Model Advisor DO-178C/DO-331 Checks 3-93](#_Toc503436763)

[Applicable qualifiable Model Advisor checks 3-94](#_Toc503436764)

[Applicable non-qualifiable Model Advisor checks 3-96](#_Toc503436765)

[Rules without applicable Model Advisor checks 3-97](#_Toc503436766)

[3.3 MathWorks Automotive Advisory Board 3-98](#_Toc503436767)

[3.4 Company-Specific Guidelines 3-99](#_Toc503436768)

[4 Constraints on Modeling Tools 4-1](#_Toc503436769)

[4.1 Modeling Guidelines for High-Integrity Systems 4-2](#_Toc503436770)

[Configuration Parameter Considerations – Solver 4-2](#_Toc503436771)

[hisl\_0040: Configuration Parameters > Solver > Simulation time 4-3](#_Toc503436772)

[hisl\_0041: Configuration Parameters > Solver > Solver options 4-4](#_Toc503436773)

[hisl\_0042: Configuration Parameters > Solver > Tasking and sample time options 4-5](#_Toc503436774)

[Configuration Parameter Considerations – Diagnostics 4-6](#_Toc503436775)

[hisl\_0036: Configuration Parameters > Diagnostics > Saving 4-7](#_Toc503436776)

[hisl\_0043: Configuration Parameters > Diagnostics > Solver 4-8](#_Toc503436777)

[hisl\_0044: Configuration Parameters > Diagnostics > Sample Time 4-10](#_Toc503436778)

[hisl\_0301: Configuration Parameters > Diagnostics > Compatibility 4-12](#_Toc503436779)

[hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters 4-13](#_Toc503436780)

[hisl\_0303: Configuration Parameters > Diagnostics > Merge block 4-14](#_Toc503436781)

[hisl\_0304: Configuration Parameters > Diagnostics > Model Initialization 4-15](#_Toc503436782)

[hisl\_0305: Configuration Parameters > Diagnostics > Debugging 4-16](#_Toc503436783)

[hisl\_0306: Configuration Parameters > Diagnostics > Connectivity > Signals 4-17](#_Toc503436784)

[hisl\_0307: Configuration Parameters > Diagnostics > Connectivity > Buses 4-18](#_Toc503436785)

[hisl\_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls 4-19](#_Toc503436786)

[hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion 4-20](#_Toc503436787)

[hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing 4-21](#_Toc503436788)

[hisl\_0311: Configuration Parameters > Diagnostics > Stateflow 4-22](#_Toc503436789)

[hisl\_0314: Configuration Parameters > Diagnostics > Data Validity > Signals 4-23](#_Toc503436790)

[Configuration Parameter Considerations – Optimizations 4-24](#_Toc503436791)

[hisl\_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double) 4-25](#_Toc503436792)

[hisl\_0046: Configuration Parameters > Optimization > Block reduction 4-26](#_Toc503436793)

[hisl\_0048: Configuration Parameters > Optimization > Application lifespan (days) 4-27](#_Toc503436794)

[hisl\_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold 4-28](#_Toc503436795)

[hisl\_0052: Configuration Parameters > Optimization > Data initialization 4-29](#_Toc503436796)

[hisl\_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values 4-30](#_Toc503436797)

[hisl\_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions 4-31](#_Toc503436798)

[Configuration Parameter Considerations – Model Referencing 4-32](#_Toc503436799)

[hisl\_0037: Configuration Parameters > Model Referencing 4-33](#_Toc503436800)

[Configuration Parameter Considerations – Code Generation 4-34](#_Toc503436801)

[hisl\_0038: Configuration Parameters > Code Generation > Comments 4-35](#_Toc503436802)

[hisl\_0039: Configuration Parameters > Code Generation > Interface 4-36](#_Toc503436803)

[hisl\_0047: Configuration Parameters > Code Generation > Code Style 4-38](#_Toc503436804)

[hisl\_0049: Configuration Parameters > Code Generation > Symbols 4-39](#_Toc503436805)

[MISRA C:2012 Compliance Considerations – Configuration Settings 4-40](#_Toc503436806)

[hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance 4-41](#_Toc503436807)

[4.2 Simulink Model Advisor DO-178C/DO-331 Checks 4-43](#_Toc503436808)

[Applicable qualifiable Model Advisor checks 4-44](#_Toc503436809)

[4.3 Company-Specific Constraints 4-46](#_Toc503436810)

[5 Model Requirements and Traceability 5-1](#_Toc503436811)

[6 Derived Requirements 6-1](#_Toc503436812)

[7 Additional Model Elements 7-1](#_Toc503436813)

[8 Suitability of Technique 8-1](#_Toc503436814)

# Introduction

## Purpose and Scope

This document comprises the Software Model Standards as referenced by the Software Development Plan (SDP), according to DO-331 MB.11.2 for the project <*Project*>. As specified in DO-331 MB.11.23, the Software Model Standards define the modeling techniques for each type of model.

This document defines all of 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 for design models as specified in DO-331 MB.1.6.2.

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

Note**:** With the exception of MATLAB®, Simulink®, and Stateflow®, this document does not include definitions or information for modeling environments.

## Applicable Documents

#### Table - Regulations and Standards

| ID | Document Title |
| --- | --- |
| DO-178C | *Software Considerations in Airborne Systems and Equipment Certification*.  RTCA, Inc., 2011 |
| DO-330 | *Software Tool Qualification Considerations*.  RTCA, Inc., 2011 |
| DO-331 | *Model-Based Development and Verification Supplement to DO-178C and DO-278A*.  RTCA, Inc., 2011. |
|  | *<List additional documents here, e.g. Advisory Circulars, EASA Certification Memos, etc.>* |

#### Table - Company and Project Plans, Standards, and Documents

| Document | Document Title |
| --- | --- |
| PSAC | ***Plan for Software Aspects of Certification for*** <*Project*> |
| SDP | ***Software Development Plan for*** <*Project*> |
| SVP | ***Software Verification Plan for*** <*Project*> |
| SCMP | ***Software Configuration Management Plan for*** <*Project*> |
| SQAP | ***Software Quality Assurance Plan for*** <*Project*> |
| SRS | ***Software Requirements Standards for*** <*Project*> |
| SDS | ***Software Design Standards for*** <*Project*> |
| SCS | ***Software Code Standards for*** <*Project*> This document. |
| SMS | ***Software Model Standards for*** <*Project*> |
| SCI | ***Software Configuration Index for*** <*Project*> \* |
| SECI | ***Software Life Cycle Environment Configuration Index for*** <*Project*> \*\* |
| SAS | *Software Accomplishment Summary* ***for*** <*Project*> |
|  | *<List additional documents>* |

\* The information that defines the software configuration for the project <*Project*> is recorded in the *Software Configuration Index* (SCI).

\*\* The tool environment used for the project <*Project*> is defined in the *Software Life Cycle Environment Configuration Index* (SECI).

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

At the completion of the project, the SCIs and SECI will be updated with the final configuration information and final document version information.

If any of the plans are revised during the project, the reasons for the changes are captured and documented in the *Software Accomplishment Summary* (SAS).

## Referenced Documents

#### Table - Referenced Documents

| ID | Document Title |
| --- | --- |
| SLHI | *Simulink Modeling Guidelines for High-Integrity System* |
| MAAB | *MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow* |
| SLCG | *Simulink Modeling Guidelines for Code Generation* |
| SLUG | *Simulink User's Guide* |
| SLREF | *Simulink Reference* |
| ECUG | *Embedded Coder User's Guide* |
| SLCIREF | *Simulink Code Inspector Reference* |
| FPUG | *Fixed-Point Designer User's Guide* |
| FPREF | *Fixed-Point Designer Reference* |
| SFUG | *Stateflow User's Guide* |
| SFREF | *Stateflow Reference* |
| MCUG | *MATLAB Coder User´s Guide* |
|  | *<List additional documents here.>* |

# Methods, Tools and Modeling Languages

This section specifies the methods, tools, and modeling languages for the development of design models, according to (DO-331 MB.11.23.a and MB.11.23.b).

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, according to (DO-331 MB.11.23.c).

*<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, R201*](http://www.mathworks.com/help/releases/R2015b/index.html)*5b. 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

### Simulink Block Considerations – Naming Conventions

hisl\_0032: Model object names

#### 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Naming** > **Check model object names**   For DO-178C/DO-331 check details, see “Check model object names”. |

|  |  |
| --- | --- |
| See Also | * MAAB guideline, Version 3.0: jc\_0201: Usable characters for Subsystem names * MAAB guideline, Version 3.0: jc\_0211: Usable characters for Inport blocks and Outport blocks * MAAB guideline, Version 3.0: jc\_0221: Usable characters for signal line names * MAAB guideline, Version 3.0: jc\_0231: Usable characters for block names * MAAB guideline, Version 3.0: na\_0019: Restricted Variable Names * MAAB guideline, Version 3.0: na\_0030: Usable characters for Simulink Bus names |
| Last Changed | R2018a |
| Examples | **Recommended**   * Block name: My\_Controller * Signal name: a\_b   **Not Recommended**   * Block name: My Controller * Signal name: 12a\_\_b |

### Simulink Block Considerations – Math Operations

hisl\_0001: Usage of Abs block

hisl\_0002: Usage of Math Function blocks (remainder and reciprocal)

hisl\_0003: Usage of Square Root blocks

hisl\_0028: Usage of Reciprocal Square Root blocks

hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)

hisl\_0005: Usage of Product blocks

hisl\_0029: Usage of Assignment block

hisl\_0066: Usage of Gain blocks

#### hisl\_0001: Usage of Abs block

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0001: Usage of Abs block** | |
| Description | To support robustness of generated code, when using the Abs block, | |
| A | Avoid Boolean and unsigned integer data types as inputs to the Abs block. |
| B | In the Abs block parameter dialog box, select **Saturate on integer overflow**. |
| Notes | The Abs block does not support Boolean data types. Specifying an unsigned input data type, might optimize the Abs block out of the generated code, resulting in a block you cannot trace to the generated code.  For signed data types, Simulink does not represent the absolute value of the most negative value. When you select **Saturate on integer overflow**, the absolute value of the data type saturates to the most positive representable value. When you clear **Saturate on integer overflow**, absolute value calculations in the simulation and generated code might not be consistent or expected. | |
| Rationale | A | Support generation of traceable code. |
| B | Achieve consistent and expected behavior of model simulation and generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Math Operations blocks**   For DO-178C/DO-331 check details, see “Check usage of Math Operations blocks”. | |
| References | * DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2016a | |
| Examples | **Recommended**    **Not Recommended** | |

#### 

#### hisl\_0002: Usage of Math Function blocks (rem and reciprocal)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0002: Usage of Math Function blocks (rem and reciprocal)** | |
| Description | To support robustness of generated code, when using the Math Function block with remainder-after-division (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. |
| Note | 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 | A, B | Protect against overflows and undefined numerical results. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Math Operations blocks**   For check details, see “Check usage of Math Operations blocks”. | |
| References | * DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2017b | |
| Examples | In the following example, when the input signal oscillates around zero, the output exhibits a large change in value. You need further protection against the large change in value. | |

#### hisl\_0003: Usage of Square Root blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0003: Usage of Square Root blocks** | |
| Description | To support robustness of generated code, when using the Square Root block, do one of the following: | |
| A | Account for complex numbers as the output. |
| B | Protect the input from going negative. |
| Rationale | A, B | Avoid undesirable results in generated code. |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check usage of Square Root blocks** | |
| References | * DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2016a | |
| Examples |  | |

#### 

#### hisl\_0028: Usage of Reciprocal Square Root blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0028: Usage of Reciprocal Square Root blocks** | |
| Description | To support robustness of generated code, when using the Reciprocal Square Root block, do one of the following: | |
| A | Protect the input from going negative. |
| B | Protect the input from going to zero. |
| Note | You can get a divide-by-zero operation, resulting in an (Inf) output value for the reciprocal function. To avoid overflows or undefined values, protect the corresponding input from going to zero. | |
| Rationale | A, B | Avoid undesirable results in generated code. |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check usage of Reciprocal Square Root blocks** | |
| References | * DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2016a | |
| Examples |  | |

#### 

#### hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm)** | |
| Description | To support robustness of generated code, when using the Math Function block with natural logarithm (log) or base 10 logarithm (log10) function parameters, | |
| A | Protect the input from going negative. |
| B | Protect the input from equaling zero. |
| C | Account for complex numbers as the output value. |
| Notes | If you set the output data type to complex, the natural logarithm and base 10 logarithm functions output complex values for negative input values. If you set the output data type to real, the functions output NaN for negative numbers, and minus infinity (-Inf) for zero values. | |
| Rationale | A, B, C | Support generation of robust code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Math Operations blocks**   For check details, see “Check usage of Math Operations blocks”. | |
| References | * DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2017b | |
| 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 | To support robustness of generated code, when using the Product block with divisor inputs, | |
| A | In Element-wise(.\*) mode, protect divisor inputs from going to zero. |
| B | In Matrix(\*) mode, protect divisor inputs from becoming singular input matrices. |
| C | Set the model configuration parameter **Diagnostics** > **Data Validity** > **Signals** > **Division by singular matrix** to error. |
| Notes | When using Product blocks for element-wise divisions, you might get a divide by zero, resulting in a NaN output. To avoid overflows, protect divisor inputs from going to zero.  When using Product blocks to compute the inverse of a matrix, or a matrix division, you might get a divide by a singular matrix. This division results in a NaN output. To avoid overflows, protect divisor inputs from becoming singular input matrices.  During simulation, while the software inverts one of the input values of a Product block that is in matrix multiplication mode, the **Division by singular matrix** diagnostic can detect a singular matrix. | |
| Rationale | A, B, C | Protect against overflows. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **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 | * DO-331, Section MB.6.4.2.2 'Robustness Test Cases' DO-331, Section MB.6.4.3 'Requirements-Based Testing Methods' DO-331, Section MB.6.3.1.e 'High-level requirements confirm to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements confirm to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2017b | |

#### 

#### 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Math Operations blocks**   For DO-178C/DO-331 check details, see “Check usage of Math Operations blocks”. |
| References | * DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' * MISRA C:2012, Rule 9.1 |
| Last Changed | R2016a |
| 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 Assignment blocks** |
| Description | To support traceability of generated code, the value of the Gain block should 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.  Setting the Gain value to a named parameter data object with a non-auto storage class is an exception to this rule. |
| Rationale | Avoid undesirable results in generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Math Operations blocks**   For DO-178C/DO-331 check details, see “Check usage of Math Operations blocks”. |
| References | * DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' |
| Last Changed | R2018a |

### Simulink Block Considerations – Ports & Subsystems

hisl\_0006: Usage of While Iterator blocks

hisl\_0007: Usage of While Iterator subsystems

hisl\_0008: Usage of For Iterator Blocks

hisl\_0009: Usage of For Iterator Subsystem 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\_0006: Usage of While Iterator blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0006: Usage of While Iterator blocks** | |
| Description | To support bounded iterative behavior in the generated code when using the While Iterator block, in the While Iterator block parameters dialog box: | |
| A | Set **Maximum number of iterations** to a positive integer value; do not set value to -1 for unlimited. |
| B | Consider selecting **Show iteration number port** to observe the iteration value during simulation. |
| Note | When you use While Iterator subsystems, set the maximum number of iterations. If you use an unlimited number of iterations, the generated code might include infinite loops, which lead to execution-time overruns.  To observe the iteration value during simulation and determine whether the loop reaches the maximum number of iterations, select the While Iterator block parameter **Show iteration number port**. If the loop reaches the maximum number of iterations, verify the output values of the While Iterator block. | |
| Rationale | A, B | Support bounded iterative in the generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Ports and Subsystems blocks**   For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”. | |
| References | * DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' * MISRA C:2012, Dir 4.1 | |
| Last Changed | R2016a | |

#### 

#### hisl\_0007: Usage of While Iterator subsystems

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0007: Usage of While Iterator subsystems** | |
| Description | To support unambiguous behavior, when using While Iterator subsystems, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems. | |
| Rationale | A, B | Avoid ambiguous behavior from the subsystem. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Ports and Subsystems blocks**   For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”. | |
| References | * DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' | |
| Last Changed | R2016a | |
| 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Ports and Subsystems blocks**   For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”. | |
| References | * DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' * MISRA C:2012, Rule 14.2 | |
| Last Changed | R2016a | |

#### 

#### hisl\_0009: Usage of For Iterator Subsystem blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0009: Usage of For Iterator Subsystem blocks** | |
| Description | To support unambiguous behavior, when using the For Iterator Subsystem block, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions within the subsystems. | |
| Rationale | A, B | Avoid ambiguous behavior from the subsystem. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Ports and Subsystems blocks**   For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”. | |
| References | * DO-331, Sections MB.6.3.1.g and MB.6.3.2.g 'Algorithms are accurate' | |
| Last Changed | R2016b | |
| Examples | See [“hisl\_0007: Usage of While Iterator subsystems”](#_bookmark15). | |

#### 

#### hisl\_0010: Usage of If blocks and If Action Subsystem blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0010: Usage of If blocks and If Action Subsystem blocks** | |
| Description | To support verifiable generated code, when using the If block with nonempty  Elseif expressions, | |
| A | In the block parameter dialog box, select **Show else condition**. |
| B | Connect the outports of the If block to If Action Subsystem blocks. |
| Prerequisites | [“hisl\_0016: Usage of blocks that compute relational operators”](#_bookmark31) | |
| 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Ports and Subsystems blocks**   For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”. | |
| References | * DO-331, Sections MB.6.3.1.g and MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' | |
| See Also | “na\_0012: Use of Switch vs. If-Then-Else Action Subsystem” in the Simulink documentation | |
| Last Changed | R2016b | |
| Examples | **Recommended: Elseif with Else**    **Not Recommended: No Else Path**    **Recommended: Only an If, no Else required**  C:\Users\rsnah\AppData\Local\Temp\SNAGHTML545998f.PNG | |

#### 

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

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks** | |
| Description | To support verifiable generated code, when using the Switch Case block: | |
| A | In the Switch Case block parameter dialog box, select **Show default case**. |
| B | Connect the outports of the Switch Case block to a Switch Case Action Subsystem block. |
| C | Use an integer data type or an enumeration value for the inputs to Switch Case blocks. |
| Prerequisites | [“hisl\_0016: Usage of blocks that compute relational operators”](#_bookmark31) | |
| 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Ports and Subsystems blocks**   For DO-178C/DO-331 check details, see “Check usage of Ports and Subsystems blocks”. | |
| References | * DO-331, Sections MB.6.3.1.g and MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' * MISRA C:2012, Rule 16.4 | |
| See Also | “db\_0115: Simulink patterns for case constructs” in the Simulink documentation. | |
| 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. |
| Notes | Conditionally executed subsystems include:   * If Action * Switch Case Action * Function-Call * Triggered * Enabled   Sample time-dependent blocks include:   * Discrete State-Space * Discrete-Time Integrator * Discrete FIR Filter * Discrete Filter * Discrete Transfer Fcn * Discrete Zero-Pole * Transfer Fcn First Order * Transfer Fcn Real Zero * Transfer Fcn Lead or Lag | |
| Rationale | A, B | Support unambiguous behavior. |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check usage of conditionally executed subsystems** | |
| References | * DO-331, Sections MB.6.3.1.g and MB.6.3.2.g 'Algorithms are accurate' | |
| Last Changed | R2016b | |
| Examples | When using discrete blocks, the behavior depends on the operation across multiple contiguous time steps. When the blocks are called intermittently, the results may not conform to your expectations. | |

#### hisl\_0024: Inport interface definition

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0024: Inport interface definition** |
| Description | To support strong data typing and unambiguous behavior of the model and the generated code, for each root-level Inport block, explicitly set the following block parameters:   * **Data type** * **Port dimensions** * **Sample time** |
| Note | Using root-level Inport blocks without fully defined dimensions, sample times, or data type can lead to ambiguous simulation results. If you do not explicitly define these parameters, Simulink back-propagates dimensions, sample times, and data types from downstream blocks. |
| Rationale | * Avoid unambiguous behavior. * Support full specification of software interface. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check for root Inports with missing properties**   For DO-178C/DO-331 check details, see “Check for root Inports with missing properties”. |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' |
| 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 capabilitiesa. Examples include: * Detection of overflows through simulation range checking. * Code optimizations using Embedded Coder. * Design model verification using Simulink Design Verifier™. * Fixed-point autoscaling using Fixed-Point Designer™. * Specified design ranges can be used by Embedded Coder to optimize the generated code. If you want to use design ranges for optimization, in the Configuration Parameters dialog box, on the **Code Generation** pane, consider selecting **Optimize using the specified minimum and maximum values**. * Ranges for bus-type Inport blocks are specified with the bus elements of the defining bus object. Simulink ignores range specifications provided directly at Inport blocks that are bus-type. |
| Rationale | Support precise specification of the input interface. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check for root Inports with missing range definitions**   For DO-178C/DO-331 check details, see “Check for root Inports with missing range definitions”. |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' |
| Last Changed | R2017b |

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

#### hisl\_0026: Design min/max specification of output interfaces

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0026: Design min/max specification of output interfaces** |
| Description | Provide design min/max information for root-level Outport blocks to specify the output interface ranges. |
| Notes | * Specifying the range of Outport blocks on the root level enables additional capabilitiesa. Examples include: * Detection of overflows through simulation range checking. * Code optimizations using Embedded Coder. * Design model verification using Simulink Design Verifier. * Fixed-point autoscaling using Fixed-Point Designer. * Specified design ranges can be used by Embedded Coder to optimize the generated code. If you want to use design ranges for optimization, in the Configuration Parameters dialog box, on the **Code Generation** pane, consider selecting **Optimize using the specified minimum and maximum values**. * Ranges for bus-type Outport blocks are specified with the bus elements of the defining bus object. Simulink ignores range specifications provided directly at Outport blocks that are bus-type. |
| Rationale | Support precise specification of the output interface. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check for root Outports with missing range definitions**   For DO-178C/DO-331 check details, see “Check for root Outports with missing range definitions”. |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' |
| Last Changed | R2017b |

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

### Simulink Block Considerations – 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 model and subsystem variants

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: | |
| A | In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, under **Data Store Memory Block**, set the following parameters to error:   * **Detect read before write** * **Detect write after read** * **Detect write after write** * **Multitask data store** * **Duplicate data store names** |
| B | Avoid data store reads and writes that occur across model and atomic subsystem boundaries. |
| C | Avoid using data stores to write and read data at different rates, because different rates can result in inconsistent exchanges of data. To provide deterministic data coupling in multirate systems, use Rate Transition blocks before Data Store Write blocks, or after Data Store Read blocks. |
| Notes | The sorting algorithm in Simulink does not take into account data coupling between models and atomic subsystems.  Using data store memory blocks can have significant impact on your software verification effort. Models and subsystems that use only inports and outports to pass data provide a directly traceable interface, simplifying the verification process. | |
| Rationale | A, B, C | Support consistent data values across different sample times or models. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **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 | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' | |
| Last Changed | R2017b | |

|  |  |
| --- | --- |
| Examples | The following examples use Rate Transition blocks to provide deterministic data coupling in multirate systems   * For fast-to-slow transitions:   Set the rate of the slow sample time on either the Rate Transition block or the Data Store Write block.    Do not place the Rate Transition block after the Data Store Read block.     * For slow-to-fast transitions:   If the Rate Transition block is after the Data Store Read block, specify the slow rate on the Data Store Read block.    If the Rate Transition block is before the Data Store Write block, use the inherited sample time for the blocks. |

#### 

#### hisl\_0015: Usage of Merge blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0015: Usage of Merge blocks** | |
| Description | To support unambiguous behavior from Merge blocks, | |
| A | Use Merge blocks only with conditionally executed subsystems. |
| B | Specify execution of the conditionally executed subsystems such that only one subsystem executes during a time step. |
| C | Clear the Merge block parameter **Allow unequal port widths**. |
| Notes | Simulink combines the inputs of the Merge block into a single output. The output value at any time is equal to the most recently computed output of the blocks that drive the Merge block. Therefore, the Merge block output is dependent upon the execution order of the input computations.  To provide predictable behavior of the Merge block output, you must have mutual exclusion between the conditionally executed subsystems feeding a Merge block. If the inputs are not mutually exclusive, Simulink uses the last input port.  Merge block parameter **Allow unequal port widths** is only available when configuration parameter **Underspecified initialization detection** is set to Classic. | |
| Prerequisites | [“hisl\_0303: Configuration Parameters > Diagnostics > Merge block”](#_bookmark31)  “hisl\_0304: Configuration Parameters > Diagnostics > Model initialization” | |
| Rationale | A, B, C | Avoid unambiguous behavior. |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check usage of Merge blocks** | |
| References | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' | |
| Last Changed | R2018a | |
| Examples | **Recommended**    **Not Recommended** | |

#### 

#### hisl\_0021: Consistent vector indexing method

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0021: Consistent vector indexing method** | |
| Description | Within a model, use: | |
| A | A consistent vector indexing method for all blocks. Blocks for which you should set the indexing method include:   * Index Vector * Multiport Switch * Assignment * Selector * For Iterator |
| Rationale | A | Reduce the risk of introducing errors due to inconsistent indexing. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check for inconsistent vector indexing methods**   For DO-178C/DO-331 check details, see “Check for inconsistent vector indexing methods”. | |
| References | * DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' | |
| See Also | “cgsl\_0101: Zero-based indexing” | |
| Last Changed | R2016a | |

#### 

#### 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 | * **By Task** > **High-Integrity Modeling Guidelines** > **Check data type selection for index signals** | |
| References | * DO-331, Section MB.6.3.4.f 'Accuracy and Consistency of Source Code' | |
| Last Changed | R2017b | |

#### 

#### hisl\_0023: Verification of model and subsystem variants

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0023: Verification of model and subsystem variants** | |
| Description | When verifying that a model is consistent with generated code, do one of the following: | |
| A | In each Model Variants block parameter dialog, verify that **Generate preprocessor conditionals** is cleared. |
| B | In each Variant Subsystem block parameter dialog, verify that **Analyze all choices during update diagram and generate preprocessor conditionals** is cleared. |
| C | Verify all combinations of model variants that might be active in the generated code. |
| Rationale | A, B | Simplify consistency testing between the model and generated code by restricting the code base to a single variant. |
| C | Make sure that consistency testing between the model and generated code is complete for all variants. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check for variant blocks with 'Generate preprocessor conditionals' active**   For DO-178C/DO-331 check details, see “Check for variant blocks with 'Generate preprocessor conditionals' active”. | |
| References | * DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' | |
| Last Changed | R2017b | |

#### hisl\_0034: Usage of Signal Routing blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0034: Usage of Signal Routing blocks** | |
| Description | To support the robustness of the operations when using Switch blocks | |
| A | Avoid comparisons using the ~= operator on floating-point data types. |
| Note | 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 | A | Improve model robustness. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Signal Routing blocks**   For check details, see “Check usage of Signal Routing blocks”. | |
| References | * DO-331, Sections MB.6.3.1.g and MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Dir 1.1 | |
| Last Changed | R2017b | |

### Simulink Block Considerations – 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 Operator block

#### hisl\_0016: Usage of blocks that compute relational operators

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0016: Usage of blocks that compute relational operators** | |
| Description | To support the robustness of the operations, when using blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare to Zero, and Detect Change | |
| A | Avoid comparisons using the == or ~= operator on floating-point data types. |
| Notes | Due to floating-point precision issues, do not test floating-point expressions for equality (==) or inequality (~=).  When the model contains a block computing a relational operator with the == or ~= operators, the inputs to the block must not be single, double, or any custom storage class that is a floating-point type. Change the data type of the input signals, or rework the model to eliminate using the == or ~= operators within blocks that compute relational operators. | |
| Rationale | A | Improve model robustness. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Logic and Bit Operations blocks**   For DO-178C/DO-331 check details, see “Check usage of Logic and Bit Operations blocks”. | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Dir 1.1 | |
| See Also | [“hisl\_0017: Usage of blocks that compute relational operators (2)”](#_bookmark32) | |
| Last Changed | R2016a | |
| Examples | Positive Pattern: To test whether two floating-point variables or expressions are equal, compare the difference of the two variables against a threshold that takes into account the floating-point relative accuracy (eps) and the magnitude of the numbers.  The following pattern shows how to test two double-precision input signals, In1 and In2, for equality. | |

#### 

#### hisl\_0017: Usage of blocks that compute relational operators (2)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0017: Usage of blocks that compute relational operators (2)** | |
| Description | To support unambiguous behavior in the generated code, when using blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare to Zero, and Detect Change | |
| A | Set the block **Output data type** parameter to Boolean. |
| Rationale | A | Support generation of code that produces unambiguous behavior. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Logic and Bit Operations blocks**   For DO-178C/DO-331 check details, see “Check usage of Logic and Bit Operations blocks”. | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Rule 10.1 | |
| See Also | [“hisl\_0016: Usage of blocks that compute relational operators”](#_bookmark31) | |
| Last Changed | R2016a | |

#### 

#### 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. |
| Prerequisites | [“hisl\_0045: Configuration Parameters > Optimization > Implement logic](#_bookmark83) [signals as Boolean data (vs. double)”](#_bookmark83) | |
| Rationale | A, B | Avoid ambiguous behavior of generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of Logic and Bit Operations blocks**   For DO-178C/DO-331 check details, see “Check usage of Logic and Bit Operations blocks”. | |
| References | * DO-331, Section MB.6.3.1.e 'High-level requirements confirm to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements confirm to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' * MISRA C:2012, Directive 1.1 | |
| Last Changed | R2017b | |

#### 

#### hisl\_0019: Usage of Bitwise Operator block

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0019: Usage of Bitwise Operator block** | |
| Description | To support unambiguous behavior, when using the Bitwise Operator block, | |
| A | Avoid signed integer data types as input to the block. |
| Notes | Bitwise operations on signed integers are not meaningful. If a shift operation moves a signed bit into a numeric bit, or a numeric bit into a signed bit, unpredictable and unwanted behavior can result. | |
| Rationale | A | Support unambiguous behavior of generated code. |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check usage of Bitwise Operator block** | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Rule 10.1 | |
| See Also | [“hisf\_0003: Usage of bitwise operations”](#_bookmark42) | |
| Last Changed | R2018a | |

### Simulink Block Considerations – Lookup Table Blocks

hisl\_0033: Usage of Lookup Table blocks

#### hisl\_0033: Usage of Lookup Table blocks

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0030: 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 | In each 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, or Prelookup block, verify that **Remove protection against out-of-range input in generated code** is cleared. |
| B | In each Interpolation Using Prelookup block, verify that **Remove protection against out-of-range index in generated code** is cleared. |
| Note | 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 | **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check usage of lookup table blocks**  For check details, see “Check usage of lookup table blocks”. | |
| References | * DO-331, Section MB.6.3.1.g and MB.6.3.2.g 'Algorithms are accurate' | |
| Last Changed | R2017b | |

### Stateflow Chart Considerations – Chart Properties

hisf\_0001: Mealy and Moore semantics

hisf\_0002: User-specified state/transition execution order

hisf\_0009: Strong data typing (Simulink and Stateflow boundary)

hisf\_0011: Stateflow debugging settings

#### hisf\_0001: Mealy and Moore semantics

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0001: Mealy and Moore semantics** | |
| Description | To create Stateflow charts that implement a subset of Stateflow semantics, | |
| A | In the Chart properties dialog box, set **State Machine Type** to Mealy or Moore. |
| B | Apply consistent settings to the Stateflow charts in a model. |
| Note | Setting **State Machine Type** restricts the Stateflow semantics to pure Mealy or Moore semantics. Mealy and Moore charts might be easier to understand and use in high-integrity applications.  In Mealy charts, actions are associated with transitions. In the Moore charts, actions are associated with states.  At compile time, the Stateflow software verifies that the chart semantics comply with the formal definitions and rules of the selected type of state machine. If the chart semantics are not in compliance, the software provides a diagnostic message. | |
| Rationale | A, B | Promote a clear modeling style. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check state machine type of Stateflow charts**   For DO-178C/DO-331 check details, see “Check state machine type of Stateflow charts”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' DO-331, Section MB.6.3.3.e 'Software architecture conform to standards' | |
| See Also | “Create Mealy and Moore Charts” in the Stateflow documentation | |
| Last Changed | R2016a | |

#### 

#### hisf\_0002: User-specified state/transition execution order

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0002: User-specified state/transition execution order** | |
| Description | Do the following to explicitly set the execution order for active states and valid transitions in Stateflow charts: | |
| A | In the Chart Properties dialog box, select **User specified state/ transition execution order**. |
| B | In the Stateflow Editor **View** menu, select **Show Transition Execution Order**. |
| C | Set default transition to evaluate last. |
| Note | Selecting **User specified state/transition execution order** restricts the dependency of a Stateflow chart semantics on the geometric position of parallel states and transitions.  Specifying the execution order of states and transitions allows you to enforce determinism in the search order for active states and valid transitions. You have control of the order in which parallel states are executed and transitions originating from a source are tested for execution. If you do not explicitly set the execution order, the Stateflow software determines the execution order following a deterministic algorithm.  Selecting **Show Transition Execution Order** displays the transition testing order. | |
| Rationale | A, B, C | Promote an unambiguous modeling style. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check Stateflow charts for ordering of states and transitions**   For DO-178C/DO-331 check details, see “Check Stateflow charts for ordering of states and transitions”. | |
| * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check usage of Stateflow constructs**   For DO-178C/DO-331 check details, see “Check usage of Stateflow constructs”. | |
| References | * DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' DO-331, Section MB.6.3.3.e 'Software architecture conform to standards ' | |
| See Also | * “Transition Testing Order in Multilevel State Hierarchy” (Stateflow) * “Execution Order for Parallel States” (Stateflow) | |
| Last Changed | R2017b | |

#### 

#### hisf\_0009: Strong data typing (Simulink and Stateflow boundary)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0009: Strong data typing (Simulink and Stateflow boundary)** | |
| Description | To support strong data typing between Simulink and Stateflow, | |
| A | Select **Use Strong Data Typing with Simulink I/O**. |
| Notes | By default, input to and output from Stateflow charts are of type double. To interface directly with Simulink signals of data types other than double, select **Use Strong Data Typing with Simulink I/O**. In this mode, data types between the Simulink and Stateflow boundary are strongly typed, and the Simulink software does not treat the data types as double. The Stateflow chart accepts input signals of any data type supported by the Simulink software, provided that the type of the input signal matches the type of the corresponding Stateflow input data object. Otherwise, the software reports a type mismatch error. | |
| Rationale | A | Support strongly typed code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check usage of Stateflow constructs**   For check details, see “Check usage of Stateflow constructs”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' | |
| Last Changed | R2017b | |

#### hisf\_0011: Stateflow debugging settings

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0011: Stateflow debugging settings** | |
| Description | To protect against unreachable code and indeterminate execution time, | |
| A | * In the Configuration Parameters dialog box, set: * **Diagnostics** > **Data Validity** > **Wrap on overflow** to error. * **Diagnostics** > **Data Validity** > **Simulation range checking** to error. * In the model window, select: * **Simulation** > **Debug** > **Stateflow Error Checking Options** > **Detect Cycles**. |
| B | For each truth table in the model, in the **Settings** menu of the Truth Table Editor, set the following parameters to Error:   * **Underspecified** * **Overspecified** |
| Notes | Run-time diagnostics are only triggered during simulation. If the error condition is not reached during simulation, the error message is not triggered for code generation. | |
| Rationale | A, B | Protect against unreachable code and unpredictable execution time. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check Stateflow debugging options**   For DO-178C/DO-331 check details, see “Check Stateflow debugging options”. | |
| * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check usage of Stateflow constructs**   For DO-178C/DO-331 check details, see “Check usage of Stateflow constructs”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' | |
| Last Changed | R2017b | |

### Stateflow Chart Considerations – Chart Architecture

hisf\_0003: Usage of bitwise operations

hisf\_0004: Usage of recursive behavior

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\_0003: Usage of bitwise operations

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0003: Usage of bitwise operations** | |
| Description | When using bitwise operations in Stateflow blocks, | |
| A | Avoid signed integer data types as operands to the bitwise operations. |
| Notes | Normally, bitwise operations are not meaningful on signed integers. Undesired behavior can occur. For example, a shift operation might move the sign bit into the number, or a numeric bit into the sign bit. | |
| Rationale | A | Promote unambiguous modeling style. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for MAAB** > **Stateflow** > **Check for bitwise operations in Stateflow charts**   For check details, see “Check for bitwise operations in Stateflow charts”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section 6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Rule 10.1 | |
| See Also | “hisl\_0019: Usage of Bitwise Operator block” | |
| Last Changed | R2016a | |

#### 

#### hisf\_0004: Usage of recursive behavior

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0004: Usage of recursive behavior** | |
| Description | To support bounded function call behavior, avoid using design patterns that include unbounded recursive behavior. Recursive behavior is bound if you do the following: | |
| A | Use an explicit termination condition that is local to the recursive call. |
| B | Make sure the termination condition is reached. |
| Notes | This rule only applies if a chart is a classic Stateflow chart. If [“hisf\_0001:](#_bookmark37) [Mealy and Moore semantics”](#_bookmark37) is followed, recursive behavior is prevented due to restrictions in the chart semantics. Additionally, you can detect the error during simulation by enabling the Stateflow diagnostic **Detect Cycles**. | |
| Rationale | A, B | Promote bounded function call behavior. |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Rule 17.2 | |
| Last Changed | R2016a | |
| Examples | There are multiple patterns in Stateflow that can result in unbounded recursion.  **Recursive Function Calls**    When the default state A is entered, event Evn is broadcast in the entry action of A. Evn results in a recursive call of the interpretation algorithm. Since A is active, the outgoing transition of A is tested. Since the current event Evn matches the transition event (and because of the absence of condition) the condition action is executed, broadcasting Evn again. This results in a new call of the interpretation algorithm which repeats the same sequence of steps until stack overflow.  **Recursive Function Calls** | |

#### 

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

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0007: Usage of junction conditions (maintaining mutual exclusion)** | |
| Description | To enhance clarity and prevent the generation of unreachable code, | |
| A | Make junction conditions mutually exclusive. |
| Notes | You can use this guideline to maintain a modeling language subset in high- integrity projects. | |
| Rationale | A | Enhance clarity and prevent generation of unreachable code. |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.d 'High-level requirements are verifiable' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' | |
| Last Changed | R2012b | |

#### 

#### hisf\_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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check Stateflow charts for transition paths that cross parallel state boundaries**   For DO-178C/DO-331 check details, see “Check Stateflow charts for transition paths that cross parallel state boundaries”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' | |
| Last Changed | R2017b | |

|  |  |
| --- | --- |
| Example | In the following example, when Out\_A is 4, both parent states (A\_Parent and B\_Parent) are reentered. Reentering the parent states resets the values of Out\_A and Out\_B to zero. |

#### 

#### 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 | * **By Task** > **High-Integrity Modeling Guidelines** > **Check usage of transition paths (passing through states)** | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' | |
| Last Changed | R2016a | |
| Examples |  | |

#### 

#### hisf\_0015: Strong data typing (casting variables and parameters in expressions)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0015: Strong data typing (casting variables and parameters in expressions)** | |
| Description | To facilitate strong data typing, | |
| A | Explicitly type cast variables and parameters of different data types in:   * Transition evaluations * Transition assignments * Assignments in states |
| Notes | The Stateflow software automatically casts variables of different type into the same data type. This guideline helps clarify data types of the intermediate variables. | |
| Rationale | A | Apply strong data typing. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check Stateflow charts for strong data typing**   For DO-178C/DO-331 check details, see “Check Stateflow charts for strong data typing”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' | |
| Last Changed | R2017b | |
| 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: Limitation of MATLAB function complexity

himl\_0005: Usage of global variables in 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 | * **By Task** > **High-Integrity Modeling Guidelines** > **Check usage of standardized MATLAB function headers** |
| References | * DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' |
| See Also | * MathWorks Automotive Advisory Board (MAAB) guideline “na\_0025: MATLAB Function Header” * [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” |
| Last Changed | R2014a |
| Examples | A typical standardized function header includes:   * Function name * Description * Inputs and outputs (if possible, include size and type) * Assumptions and limitations * Revision history |

#### himl\_0002: Strong data typing 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **MATLAB** > **Check for MATLAB Function interfaces with inherited properties**   For DO-178C/DO-331 check details, see “Check for MATLAB Function interfaces with inherited properties”. |
| References | * DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent |
| See Also | * MathWorks Automotive Advisory Board (MAAB) guideline “na\_0034: MATLAB Function block input/output settings” * [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” |
| 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: Limitation of MATLAB function complexity

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **himl\_0003: Limitation of MATLAB function complexity** | |
| 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) | |
| Note | 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 |
| 1Pure Wrappers to external functions are not counted as separate levels.  2Standard MATLAB library functions do not count as separate levels. | |
| Rationale | * Readability * Comprehension * Traceability * Maintainability * Testability | |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **MATLAB** > **Check MATLAB Function metrics**   For DO-178C/DO-331 check details, see “Check MATLAB Function metrics”. | |
| References | * DO-331, Sections MB.6.3.1.e - High-level requirements conform to standards DO-331, Sections MB.6.3.2.e - Low-level requirements conform to standards | |
| See Also | * MathWorks Automotive Advisory Board (MAAB) guideline “na\_0016: Source lines of MATLAB Functions” * MathWorks Automotive Advisory Board (MAAB) guideline “na\_0017: Number of called function levels” * MathWorks Automotive Advisory Board (MAAB) guideline “na\_0018: Number of nested if/else and case statement” * [Orion GN&C: MATLAB and Simulink Standards](http://www.mathworks.com/aerospace-defense/standards/FltDyn-CEV-08-148_MATLAB_Standards_v9_20111202.pdf), “jh\_0084: eML Comments” * “MATLAB Function Block Editor” | |
| Last Changed | R2016a | |

#### himl\_0005: Usage of global variables in MATLAB functions

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0005: Usage of global variables in MATLAB functions** |
| Description | Avoid using global variables in MATLAB functions. To access shared data, use signal lines or persistent data. |
| Notes | Using global data in MATLAB code requires the definition of Data Store Memory blocks or Custom Storage class objects. If the read and write access order is not specified correctly, usage of this type of storage can lead to unexpected results. |
| Rationale | * Readability * Maintainability * Deterministic Behavior |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **MATLAB** > **Check MATLAB code for global variables**   For DO-178C/DO-331 check details, see “Check MATLAB code for global variables”. |
| References | * DO-331, Section MB.6.3.3.b 'Consistency’ |
| See Also | * “na\_0024: Global Variables” * “hisl\_0013: Usage of data store blocks” |
| Last Changed | R2016a |
| Examples | * **Recommended**   function [Y,newG] = ...  fcn(U,oldG)  %#codegen  Y = oldG \* U;  newG = oldG + 1;  end     * **Recommended**   function Y = fcn(U)  %#codegen  persistent G;  if isempty(G)  G = 1;  end     * **Not Recommended**   Write to global data function:  function fcn(U)  %#codegen  global G;  G = U;  end  Read from global data function:  function Y = fcn  %#codegen  global G;  Y = G;  end |

### MATLAB Function and MATLAB Code Considerations – 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\_0009: MATLAB code with equal / not equal relational operators

himl\_0010: MATLAB code with logical operators and functions

#### 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **MATLAB** > **Check MATLAB Code Analyzer messages**   For DO-178C/DO-331 check details, see “Check MATLAB Code Analyzer messages”. | |
| References | * DO-331, Section MB.6.3.1.b 'Accuracy and consistency' DO-331, Section MB.6.3.2.b 'Accuracy and consistency' | |
| See Also | “Check Code for Errors and Warnings” (MATLAB) | |
| 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 | * **By Task** > **High-Integrity Modeling Guidelines** > **Check MATLAB code if / elseif / else patterns** |
| References | * DO-331, Section MB.6.3.1.e 'Conformance to standards' DO-331, Section MB.6.3.2.e 'Conformance to standards' DO-331, Section MB.6.3.3.e 'Conformance to standards' |
| See Also | * “hisl\_0010: Usage of If blocks and If Action Subsystem blocks” |
| Last Changed | R2016a |
| 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. |
| Note | If there is only one case and one otherwise statement, consider using an  if / else statement. |
| Rationale | * Defensive programming * Readability * Traceability |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check MATLAB code switch / case / otherwise patterns** |
| References | * DO-331, Section MB.6.3.1.e 'Conformance to standards' DO-331, Section MB.6.3.2.e 'Conformance to standards' DO-331, Section MB.6.3.3.e 'Conformance to standards' * MISRA C:2012, Rule 16.4 |
| See Also | * “na\_0022: Recommended patterns for Switch/Case statements” * “hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks” |
| Last Changed | R2016a |
| 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. |
| Note | If the two operands have different data types, MATLAB will promote both operands to a common data type. This can lead to unexpected results. |
| Rationale | * Prevent implicit casts * Prevent unexpected results |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check MATLAB code relational operator data types** |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' |
| See Also | * “hisl\_0016: Usage of blocks that compute relational operators” * “hisl\_0017: Usage of blocks that compute relational operators (2)” |
| Last Changed | R2014a |
| Examples | **Recommended**   * myBool == true   myInt8 == int8(1)  **Not Recommended**   * myBool == 1   myInt8 == true  myInt8 == 1  myInt8 == int16(1)  myEnum1.EnumVal == int32(1) |

#### 

#### himl\_0009: MATLAB code with equal / not equal relational operators

|  |  |
| --- | --- |
| **ID: Title** | **himl\_0009: MATLAB code with equal / not equal relational operators** |
| Description | For MATLAB code with equal or not equal relational operators, avoid using the following data types:   * Single * Double * Types derived from single or double data types |
| Note | Consider the following code fragments:   1. sqrt(2)^2 == 2 2. sqrt(2^2) == 2   Mathematically, both fragments are true. However, because of floating point rounding effects, the results are:   1. false 2. true |
| Rationale | * Prevent unexpected results |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check MATLAB code with equal / not equal relational operators** |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' * MISRA C:2012, Dir 1.1 |
| See Also | * “jc\_0481: Use of hard equality comparisons for floating point numbers in Stateflow” * “hisl\_0016: Usage of blocks that compute relational operators” |
| Last Changed | R2016a |
| Examples | **Recommended**   * myDouble >= 0.99 && myDouble <= 1.01; % test range   **Not Recommended**   * myDouble == 1.0   mySingle ~= 15.0 |

#### 

#### 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 | * **By Task** > **High-Integrity Modeling Guidelines** > **Check MATLAB code with logical operators and functions** |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' |
| Last Changed | R2016a |
| Examples | **Recommended**   * ~myLogical   (myInt8 > int8(4)) && myLogical  xor(myLogical1,myLogical2)  **Not Recommended**   * ~myInt8   myInt8 && myDouble |

### 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\_0201: Define reserved keywords 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check Stateflow charts for uniquely defined data objects**   For DO-178C/DO-331 check details, see “Check Stateflow charts for uniquely defined data objects”. | |
| * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check usage of Stateflow constructs**   For DO-178C/DO-331 check details, see “Check usage of Stateflow constructs”. | |
| References | * DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' | |
| See Also | “Code Appearance” in the Simulink Coder™ documentation | |
| 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 | * **By Task** > **High-Integrity Modeling Guidelines** > **Check global variables in graphical functions** |
| References | * DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Rule 13.2 MISRA C:2012, Rule 13.5 |
| Last Changed | R2016a |
| 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, limit the length of user defined names to **Maximum identifier length** (MaxIdLength). | |
| **Note:** The default of Maximum identifier length is 31. | |
| A | When working with Subsystem blocks with the block parameter Function name options set to User specified, limit the length of function names to parameter Maximum identifier length (MaxIdLength) characters or fewer. |
| 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 Alias name. |
| Rationale | User defined names 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 Alias name. | |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check length of user-defined object names to improve MISRA C:2012 compliance** | |
| References | * DO-331, Section MB.6.3.4.d 'Source code conforms to standards' * 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”](#_bookmark31) | |
| Last Changed | R2017a | |

#### hisl\_0201: Define reserved keywords to improve MISRA C:2012 compliance

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0201: Define reserved keywords to improve MISRA C:2012 compliance** | |
| Description | To improve MISRA C:2012 compliance of the generated code, define reserved keywords to prevent identifier clashes within the project namespace. | |
| A | In the Configuration Parameters dialog box, on the **Simulation Target** pane, define reserved identifiers. |
| B | Use a consistent set of reserved identifiers for all models. |
| Notes | Simulink Coder checks models for standard C language key words. Expand the list of reserved identifiers to include project specific identifiers. Examples include target-specific clashes, standard and custom library clashes, and other identified clashes. | |
| Rationale | Improve MISRA C:2012 compliance of the generated code. | |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check reserved keywords to improve MISRA C:2012 compliance** | |
| See Also | * “Simulation Target Pane: Symbols” in the Simulink documentation * “Reserved Keywords” in the Simulink Coder documentation * “Reserved names” in the Simulink Coder documentation | |
| References | * DO-331, Section MB.6.3.4.d 'Source code conforms to standards' * MISRA C:2012, Rule 21.2 | |
| Last Changed | R2015b | |

### 

### MISRA C:2012 Compliance Considerations – Block Usage

hisl\_0020: Blocks not recommended for C/C++ production code deployment and 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 C/C++ production code deployment and MISRA C:2012 compliance

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0020: Blocks not recommended for C/C++ production code deployment and 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. |
| 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. |
| 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.  Choose Simulink **Help** > **Simulink** > **Block Data Types & Code Generation Support** > **All Tables** 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 | Improve quality and MISRA C:2012 compliance of the generated code. |
| Model Advisor Checks | To check model for conditions A, B, C, D, E, and F:  **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Code** > **Check for blocks not recommended for MISRA C:2012**  For DO-178C/DO-331 check details, see “Check for blocks not recommended for MISRA C:2012”. | |
| To check model for conditions A and B:  **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Simulink** > **Check for blocks not recommended for C/C++ production code deployment**  For DO-178C/DO-331 check details, see “Check for blocks not recommended for C/C++ production code deployment”. | |
| References | * DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.4.d 'Source code conforms to standards' * MISRA C:2012 | |
| Last Changed | R2017b | |

#### 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 |
| Note | 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. |
| Model Advisor Checks | * **By Task** > **Simulink Design Verifier Design Error Checks** > **Detect Dead Logic** |
| References | * DO-331, Section MB.6.3.1.d 'High-level requirements are verifiable' DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' DO-331, Section MB.6.3.4.d 'Source code conforms to standards' * MISRA C:2012, Rule 14.3 MISRA C:2012, Rule 2.1 |
| Last Changed | R2015b |
| Example | 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 then 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 and while loops constructed in Stateflow and MATLAB. * For Iterator blocks. |
| Rationale | Improve MISRA C:2012 compliance of the generated code. |
| Model Advisor Checks | * **By Task** > **High-Integrity Modeling Guidelines** > **Check data type of loop control variables to improve MISRA C:2012 compliance** |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate'  DO-331, Section MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.4.d 'Source code conforms to standards' * MISRA C:2012, Rule 14.1 |
| Last Changed | R2018a |

### MISRA C:2012 Compliance Considerations – Stateflow Chart Considerations

hisf\_0064: Shift operations for Stateflow data to improve code compliance

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\_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA C:2012 compliance

#### hisf\_0064: Shift operations for Stateflow data to improve code compliance

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0064: Shift operations for Stateflow data to improve code compliance** | |
| Description | To improve code compliance of the generated code with Stateflow bit-shifting operations, do not perform: | |
| A | Right-shift operations greater than the bit-width of the input type, or by a negative value. |
| B | Left-shift operations greater than the bit-width of the output type, or by a negative value. |
| Note | If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the coding standards. | |
| Rationale | A, B | To avoid shift operations in the generated code that might be coding standard violation. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check usage of shift operations for Stateflow data**   For DO-178C/DO-331 check details, see “Check usage of shift operations for Stateflow data”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' | |
| Prerequisites | [“hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance”](#_bookmark31) | |
| Last Changed | R2017b | |

#### 

#### 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 | To improve code compliance of the generated code, protect against Stateflow casting integer and fixed-point calculations to wider data types than the input data types by: | |
| A | Using the := notation in Stateflow that use the C action language |
| Note | If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the coding standards. | |
| Rationale | A | To avoid implicit casts in the generated code that might be a coding standard violation. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check assignment operations in Stateflow charts**   For DO-178C/DO-331 check details, see “Check assignment operations in Stateflow charts”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' | |
| Prerequisites | [“hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance”](#_bookmark31) | |
| Last Changed | R2017b | |

#### 

#### 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 |
| Note | 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Stateflow** > **Check Stateflow charts for unary operators**   For DO-178C/DO-331 check details, see “Check Stateflow charts for unary operators”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' * MISRA C:2012, Rule 10.1 | |
| Last Changed | R2017b | |

#### 

#### hisf\_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA C:2012 compliance

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisf\_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA C:2012 compliance** | |
| Description | To improve MISRA C:2012 compliance of the generated code for floating point and integer-based operations, do one of the following: | |
| A | Perform static analysis of the model to prove that division by zero is not possible |
| B | Provide run-time error checking in the generated C code by explicitly modeling the error checking in Stateflow |
| C | Modify the code generation process using Code Replacement Libraries (CRLs) to protect against division by zero |
| D | For integer-based operations, clear configuration parameter **Remove code that protects against division arithmetic exceptions** |
| Note | Using run-time error checking introduces additional computational and memory overhead in the generated code. Therefore, it is preferable to use static analysis tools to limit errors in the generated code.  You can use the design error detection functionality in Simulink® Design Verifier™ to perform the static analysis. For more information, see “Static Run-Time Error Detection” (Simulink Design Verifier). Alternatively, if you have a Simulink Design Verifier license, you can use Model Advisor check **Detect Division By Zero** to identify division-by-zero errors in your model.  If static analysis determines that sections of the code can have a division by zero, then add run-time protection into that section of the model (see example). Using a modified CRL or selecting the parameter **Remove code that protects against division arithmetic exceptions** protects division operations against divide-by-zero operations. However, this action does introduce additional computational and memory overhead.  Use only one of the run-time protections (B, C or D) in a model. Using more than one option can result in redundant protection operations. | |
| Rationale | A, B, C, D | Improve MISRA C:2012 compliance of the generated code |
| Model Advisor Checks | * **By Task** > **Simulink Design Verifier Design Error Checks** > **Detect Division By Zero** | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate'  DO-331, Section MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.4.d 'Source code conforms to standards' * MISRA C:2012, Dir 4.1 | |

|  |  |
| --- | --- |
| See Also | * “What Is Code Replacement?” (Simulink Coder) and “Code Replacement Libraries” (Simulink Coder) * “hisl\_0002: Usage of Math Function blocks (rem and reciprocal)” * “hisl\_0005: Usage of Product blocks” * “hisl\_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions” * “Detect Division By Zero” |
| Last Changed | R2018a |
| Example | Run-time divide by zero protection can be realized using a graphical function. Unique functions should be provided for each data type. |

## Simulink Model Advisor DO-178C/DO-331 Checks

Apply the following checks documented in the "DO Qualification Kit Simulink® Verification and Validation™ DO-178C/DO-331 Checks and Model Advisor User Information":

Display model version information

Check usage of lookup table blocks

Check for inconsistent vector indexing methods

Check for blocks not recommended for C and C++ production code deployment

Check for variant blocks with ‘Generate preprocessor conditionals’ active

Check for root Inports with missing properties

Check usage of Math Operations blocks

Check usage of Signal Routing blocks

Check usage of Logic and Bit Operations blocks

Check usage of Ports and Subsystems blocks

Check for root Inports with missing range definitions

Check for root Outports with missing range definitions

Check state machine type of Stateflow charts

Check Stateflow charts for transition paths that cross parallel state boundaries

Check Stateflow charts for ordering of states and transitions

Check Stateflow debugging options

Check Stateflow charts for uniquely defined data objects

Check Stateflow charts for strong data typing

Check usage of shift operations for Stateflow data

Check assignment operations in Stateflow charts

Check Stateflow charts for unary operators

Check usage of Stateflow constructs

Check for MATLAB Function interfaces with inherited properties

Check MATLAB Function metrics

Check MATLAB Code Analyzer messages

Check MATLAB code for global variables

Check safety-related diagnostic settings for signal data

Check safety-related diagnostic settings for data store memory

Check model object names

Check for blocks not recommended for MISRA C:2012

#### Applicable qualifiable Model Advisor checks

|  |  |
| --- | --- |
| **Model Advisor Checks** | **Applicable High-Integrity System Modeling Guidelines** |
| Display model version information | Not a rule checker. This is used for inserting model checksum information into the Model Advisor report. |
| Check usage of lookup table blocks | hisl\_0033: Usage of Lookup Table blocks |
| Check for inconsistent vector indexing methods | hisl\_0021: Consistent vector indexing method |
| Check for blocks not recommended for C and C++ production code deployment | hisl\_0020: Blocks not recommended for C/C++ production code deployment and MISRA C:2012 compliance |
| Check for variant blocks with ‘Generate preprocessor conditionals’ active | hisl\_0023: Verification of model and subsystem variants |
| Check for root Inports with missing properties | hisl\_0024: Inport interface definition |
| Check usage of Math Operations blocks | * hisl\_0001: Usage of Abs block * hisl\_0002: Usage of Math Function blocks (rem and reciprocal) * hisl\_0004: Usage of Math Function blocks (natural logarithm and base 10 logarithm) * hisl\_0029: Usage of Assignment blocks |
| Check usage of Signal Routing blocks | hisl\_0034: Usage of Signal Routing blocks |
| Check usage of Logic and Bit Operations blocks | * 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 |
| Check usage of Ports and Subsystems blocks | * hisl\_0006: Usage of While Iterator blocks * hisl\_0007: Usage of While Iterator subsystems * hisl\_0008: Usage of For Iterator blocks * hisl\_0009: Usage of For Iterator Subsystem blocks * hisl\_0010: Usage of If blocks and If Action Subsystem blocks * hisl\_0011: Usage of Switch Case blocks and Action Subsystem blocks |
| Check for root Inports with missing range definitions | hisl\_0025: Design min/max specification of input interfaces |
| Check for root Outports with missing range definitions | hisl\_0026: Design min/max specification of output interfaces |
| Check state machine type of Stateflow charts | hisf\_0001: Mealy and Moore semantics |
| Check Stateflow charts for transition paths that cross parallel state boundaries | hisf\_0013: Usage of transition paths (crossing parallel state boundaries) |
| Check Stateflow charts for ordering of states and transitions | hisf\_0002: User-specified state/transition execution order |
| Check Stateflow debugging options | hisf\_0011: Stateflow debugging settings |
| Check Stateflow charts for uniquely defined data objects | hisl\_0061: Unique identifiers for clarity |
| Check Stateflow charts for strong data typing | hisf\_0015: Strong data typing (casting variables and parameters in expressions) |
| Check usage of shift operations for Stateflow data | hisf\_0064: Shift operations for Stateflow data to improve code compliance |
| Check assignment operations in Stateflow charts | hisf\_0065: Type cast operations in Stateflow to improve code compliance |
| Check Stateflow charts for unary operators | hisf\_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance |
| Check usage of Stateflow constructs | * hisf\_0002: User-specified state/transition execution order * hisf\_0009: Strong data typing (Simulink and Stateflow boundary) * hisf\_0011: Stateflow debugging settings * hisl\_0061: Unique identifiers for clarity |
| Check for MATLAB Function interfaces with inherited properties | himl\_0002: Strong data typing at MATLAB function boundaries |
| Check MATLAB Function metrics | himl\_0003: Limitation of MATLAB function complexity |
| Check MATLAB Code Analyzer messages | himl\_0004: MATLAB Code Analyzer recommendations for code generation |
| Check MATLAB code for global variables | himl\_0005: Usage of global variables in MATLAB functions |
| Check safety-related diagnostic settings for signal data | hisl\_0005: Usage of Product blocks |
| Check safety-related diagnostic settings for data store memory | hisl\_0013: Usage of data store blocks |
| Check model object names | hisl\_0032: Model object names |
| Check for blocks not recommended for MISRA C:2012 | hisl\_0020: Blocks not recommended for C/C++ production code deployment and MISRA C:2012 compliance |
| Detect Dead Logic | hisl\_0101: Avoid invariant comparison operations to improve MISRA C:2012 compliance |
| Detect Division By Zero | hisf\_0213: Protect against divide-by-zero calculations in Stateflow charts to improve MISRA C:2012 compliance |

#### Applicable non-qualifiable Model Advisor checks

|  |  |
| --- | --- |
| **Model Advisor Checks** | **Applicable High-Integrity System Modeling Guidelines** |
| Check for bitwise operations in Stateflow charts | hisf\_0003: Usage of bitwise operations |
| Check usage of Square Root blocks | hisl\_0003: Usage of Square Root blocks |
| Check usage of Reciprocal Square Root blocks | hisl\_0028: Usage of Reciprocal Square Root blocks |
| Check usage of conditionally executed subsystems | hisl\_0012: Usage of conditionally executed subsystems |
| Check usage of Merge blocks | hisl\_0015: Usage of Merge blocks |
| Check data type selection for index signals | hisl\_0022: Data type selection for index signals |
| Check usage of Bitwise Operator Block | hisl\_0019: Usage of Bitwise Operator block |
| Check usage of transition paths (passing through states) | hisf\_0014: Usage of transition paths (passing through states) |
| Check usage of standardized MATLAB function headers | himl\_0001: Usage of standardized MATLAB function headers |
| Check MATLAB code if / elseif / else patterns | himl\_0006: MATLAB code if / elseif / else patterns |
| Check MATLAB code switch / case / otherwise patterns | himl\_0007: MATLAB code switch / case / otherwise patterns |
| Check MATLAB code relational operator data types | himl\_0008: MATLAB code relational operator data types |
| Check MATLAB code with equal / not equal relational operators | himl\_0009: MATLAB code with equal / not equal relational operators |
| Check MATLAB code with logical operators and functions | himl\_0010: MATLAB code with logical operators and functions |
| Check global variables in graphical functions | hisl\_0062: Global variables in graphical functions |
| Check length of user-defined object names to improve MISRA C:2012 compliance | hisl\_0063: Length of user-defined function names to improve MISRA C:2012 compliance |
| Check reserved keywords to improve MISRA C:2012 compliance | hisl\_0201: Define reserved keywords to improve MISRA C:2012 compliance |
| Check data type of loop control variables to improve MISRA C:2012 compliance | hisl\_0102: Data type of loop control variables to improve MISRA C:2012 compliance |

The Model Advisor checks for the above rules are not qualified. These rules must be manually reviewed using the supplied checklist.

#### Rules without applicable Model Advisor checks

|  |  |
| --- | --- |
| **Applicable High-Integrity System Modeling Guidelines that Must Be Manually Reviewed** | **Alternative Method** |
| hisf\_0004: Usage of recursive behavior | Run the model with **Simulation** > **Debug** > **MATLAB & Stateflow Error Checking Options** > **Detect Cycles** enabled |
| hisf\_0007: Usage of junction conditions (maintaining mutual exclusion) | Manual design review |

There is no Model Advisor checks for the above rules. These rules must be manually reviewed using the supplied checklist.

## MathWorks Automotive Advisory Board

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

## Company-Specific Guidelines

*<Please 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 (DO-331MB.11.23.d).

*<Please 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, R201*](http://www.mathworks.com/help/releases/R2015b/index.html)*5b. Delete the following guidelines if they are not applicable in your project>*

## Modeling Guidelines for High-Integrity Systems

<Describe the means of compliance to be used for this project; normally this will reference DO-178. This section should also include references to TC, STC, TSO, Advisory Circulars, Special Conditions, Etc.>

### 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 | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Solver** pane, set parameters for simulation time as follows: | |
| A | **Start time** to 0.0 |
| B | **Stop time** to a positive value that is less than the value of **Application lifespan (days)**. |
| Note | Simulink® allows nonzero start times for simulation. However, production code generation requires a zero start time.  By default, **Application lifespan (days)** is auto. If you do not change this setting, any positive value for **Stop time** is valid.  You specify **Stop time** in seconds and **Application lifespan (days)** is in days. | |
| Rationale | A | Generate code that is valid for production code generation. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related solver settings for simulation time**   For DO-178C/DO-331 check details, see “Check safety-related solver settings for simulation time”. | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' | |
| See Also | * [“hisl\_0048: Configuration Parameters > Optimization > Application](#_bookmark85) [lifespan (days)”](#_bookmark85) * Solver Pane section of the Simulink documentation | |
| 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). |
| Note | 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related solver settings for solver options**   For DO-178C/DO-331 check details, see “Check safety-related solver settings for solver options”. | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' | |
| See Also | “Solver Pane” in the Simulink documentation | |
| 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 | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Solver** pane, set parameters for tasking and sample time as follows: | |
| A | **Periodic sample time constraint** to Specified and assign values to **Sample time properties**. |
| **Caution:** If you use a referenced model as a reusable function, set **Periodic sample time constraint** to Ensure sample time independent. |
| B | Clear the parameter **Automatically handle data transfers between tasks**. |
| Notes | Selecting the **Automatically handle data transfers between tasks** check box might result in inserting rate transition code without a corresponding model construct. This might impede establishing full traceability or showing that unintended functions are not introduced.  You can select or clear the **Higher priority value indicates higher task priority** check box. Selecting this check box determines whether the priority for **Sample time properties** uses the lowest values as highest priority, or the highest values as highest priority. | |
| Rationale | A, B | Support fully specified models and unambiguous code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related solver settings for tasking and sample-time**   For DO-178C/DO-331 check details, see “Check safety-related solver settings for tasking and sample-time”. | |
| References | * DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' | |
| See Also | “Solver Pane” in the Simulink documentation | |
| Last Changed | R2017b | |

### Configuration Parameter Considerations – Diagnostics

hisl\_0036: Configuration Parameters > Diagnostics > Saving

hisl\_0043: Configuration Parameters > Diagnostics > Solver

hisl\_0044: Configuration Parameters > Diagnostics > Sample Time

hisl\_0301: Configuration Parameters > Diagnostics > Compatibility

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

hisl\_0303: Configuration Parameters > Diagnostics > Merge block

hisl\_0304: Configuration Parameters > Diagnostics > Model Initialization

hisl\_0305: Configuration Parameters > Diagnostics > 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\_0036: Configuration Parameters > Diagnostics > Saving

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0036: Configuration Parameters > Diagnostics > Saving** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, set these parameters:   * **Block diagram contains disabled library links** to error * **Block diagram contains parameterized library links** to error |
| Rationale | Prevent unexpected results. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety- related diagnostic settings for saving**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for saving”. |
| References | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' |
| Last Changed | R2017b |

#### hisl\_0043: Configuration Parameters > Diagnostics > Solver

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0043: Configuration Parameters > Diagnostics > Solver** | |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** pane, set the **Solver** parameters as follows:   * **Algebraic loop** to error. * **Minimize algebraic loop** to error. * **Unspecified inheritability of sample times** to error. * **Automatic solver parameter selection** to error. * **State name clash** to warning. * **Block priority violation** to error if you are using block priorities. | |
| Note | Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines. | |
| **If Diagnostic Parameter...** | **Is Not Set As Indicated, Then ...** |
| **Algebraic loop** | Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution. |
| **Minimize algebraic loop** | Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution. |
| **Block priority violation** | Block execution order can include undetected conflicts that might result in unpredictable block order execution. |
| **Unspecified inheritability of sample times** | An S-function that is not explicitly set to inherit sample time can go undetected and result in unpredictable behavior. |
| **Automatic solver parameter selection** | An automatic change to the solver, step size, or simulation stop time 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. |
| You can set the following solver diagnostic parameters to any value:   * **Min step size violation** * **Consecutive zero crossings violation** * **Solver data inconsistency** * **Extraneous discrete derivative signals** | |
| Rationale | Support generation of robust and unambiguous code. | |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for solvers**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for solvers”. | |
| References | * DO-331, MB.6.3.3.e 'Software architecture conforms to standards' | |
| See Also | * “Model Configuration Parameters: Diagnostics” in the Simulink® documentation * “jc\_0021: Model diagnostic settings” in the Simulink documentation | |
| Last Changed | R2017b | |

#### hisl\_0044: Configuration Parameters > Diagnostics > Sample Time

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0044: Configuration Parameters > Diagnostics > Sample Time** | |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Sample Time** pane, set the following **Sample Time** parameters to error:   * **Source block specifies -1 sample time** * **Multitask rate transition** * **Single task rate transition** * **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. | |
| Note | Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines. | |
| **If Diagnostic Parameter...** | **Is Not Set As Indicated, Then ...** |
| **Source block specifies -1 sample time** | Use of inherited sample times for a source block, such as Sine Wave, can go undetected and result in unpredictable execution rates for source and downstream blocks. |
| **Multitask rate transition** | Invalid rate transitions between two blocks operating in multitasking mode can go undetected. You cannot use invalid rate transitions for embedded real-time software applications. |
| **Single task rate transition** | A rate transition between two blocks operating in single-tasking mode can go undetected. You cannot use single- tasking rate transitions for embedded real-time software applications. |
| **Multitask conditionally executed subsystems** | A conditionally executed multirate subsystem, operating in multitasking mode might go undetected and corrupt data or show unexpected behavior in a target system that allows preemption. |
| **Tasks with equal priority** | Two asynchronous tasks with equal priority might go undetected and show unexpected behavior in target systems that allow preemption. |
| **Enforce sample times specified by Signal Specification blocks** | Inconsistent sample times for a Signal Specification block and the connected destination block might go undetected and result in unpredictable execution rates. |
| **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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for sample time**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for sample time”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' | |
| See Also | “Model Configuration Parameters: Sample Time Diagnostics” in the Simulink documentation | |
| Last Changed | R2017b | |

#### hisl\_0301: Configuration Parameters > Diagnostics > Compatibility

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0301: Configuration Parameters > Diagnostics > Compatibility** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Compatibility** pane, set the **Compatibility** parameters as follows:   * **S-function upgrades needed** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for compatibility**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for compatibility”. |
| References | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' |
| See Also | “Model Configuration Parameters: Compatibility Diagnostics” in the Simulink documentation |
| Last Changed | R2017b |

#### 

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

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set the **Parameters** parameters as follows:   * **Detect downcast** to error * **Detect precision loss** to error * **Detect overflow** to error * **Detect underflow** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for parameters**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for parameters”. |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate'  DO-331, Section MB.6.3.2.g 'Algorithms are accurate' |
| See Also | “Model Configuration Parameters: Data Validity Diagnostics” in the Simulink documentation |
| Last Changed | R2017b |

#### 

#### hisl\_0303: Configuration Parameters > Diagnostics > Merge block

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0303: Configuration Parameters > Diagnostics > Merge block** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, set:   * **Detect multiple driving blocks executing at the same time step** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for Merge blocks**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for Merge blocks”. |
| References | * DO-331, Section MB.6.3.2.b 'Accuracy and Consistency' |
| See Also | “Detect multiple driving blocks executing at the same time step” in the Simulink documentation |
| Last Changed | R2017b |

#### 

#### hisl\_0304: Configuration Parameters > Diagnostics > Model Initialization

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0304: Configuration Parameters > Diagnostics > Model Initialization** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, set:   * **Underspecified initialization detection** to Simplified |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for model initialization**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for model initialization”. |
| References | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' |
| See Also | “Underspecified initialization detection” in the Simulink documentation |
| Last Changed | R2017b |

#### 

#### hisl\_0305: Configuration Parameters > Diagnostics > Debugging

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0305: Configuration Parameters > Diagnostics > Debugging** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, set **Model Verification block enabling** to Disable all. |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for data used for debugging**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for data used for debugging”. |
| References | * DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' |
| See Also | “Model Verification block enabling” in the Simulink documentation |
| Last Changed | R2017b |

#### 

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

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0306: Configuration Parameters > Diagnostics > Connectivity > Signals** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for signal connectivity**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for signal connectivity”. |
| References | * DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' |
| See Also | “Model Configuration Parameters: Connectivity Diagnostics” in the Simulink documentation |
| Last Changed | R2017b |

#### 

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

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0307: Configuration Parameters > Diagnostics > Connectivity > Buses** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Connectivity** pane, set the **Buses** parameters as follows:   * **Unspecified bus object at root Outport block** to error * **Element name mismatch** to error * **Non-bus signals treated as bus signals** to error * **Repair bus selections** to Warn and repair |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for bus connectivity**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for bus connectivity”. |
| References | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' |
| See Also | “Model Configuration Parameters: Connectivity Diagnostics” in the Simulink documentation |
| Last Changed | R2017b |

#### 

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

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Connectivity** pane, set the **Function calls** parameters as follows:   * **Invalid function-call connection** to error * **Context-dependent inputs** to Enable all as errors |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings that apply to function-call connectivity**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings that apply to function-call connectivity”. |
| References | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' |
| See Also | “Diagnostics Pane: Connectivity” in the Simulink documentation |
| Last Changed | R2017b |

#### 

#### hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Type Conversion** pane, set the **Type Conversion** parameters as follows:   * **Vector/matrix block input conversion** to error * **Unnecessary type conversion** to warning * **32-bit integer to single precision float conversion** to warning |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for type conversions**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for type conversions”. |
| References | * DO-331, Section MB.6.3.1.g 'Algorithm are accurate' DO-331, Section MB.6.3.2.g 'Algorithm are accurate' |
| See Also | “Model Configuration Parameters: Type Conversion Diagnostics” in the Simulink documentation |
| Last Changed | R2017b |

#### 

#### hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Model Referencing** pane, set the **Model Referencing** parameters as follows:   * **Model block version mismatch** to none * **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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for model referencing**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for model referencing”. |
| References | * DO-331, Section MB.6.3.1.d 'High-level requirements are verifiable' DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' |
| See Also | “Model Configuration Parameters: Model Referencing Diagnostics” in the Simulink documentation |
| Last Changed | R2018a |

#### 

#### hisl\_0311: Configuration Parameters > Diagnostics > Stateflow

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0311: Configuration Parameters > Diagnostics > Stateflow** |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Diagnostics** > **Stateflow** pane, set the **Stateflow** parameters as follows:   * **Unexpected backtracking** to error * **Invalid input data access in chart initialization** to error * **No unconditional default transitions** to error * **Transitions outside natural parent** to error * **Transition shadowing** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for Stateflow**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for Stateflow”. |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.1.e 'High-level requirements confirm to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' DO-331, Section MB.6.3.2.e 'Low-level requirements confirm to standards' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' |
| See Also | “Model Configuration Parameters: Stateflow Diagnostics” in the Simulink documentation |
| Last Changed | R2017b |

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

|  |  |
| --- | --- |
| **ID: Title** | **hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Signals** |
| Description | For models used to develop high-integrity systems, 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 * **Wrap on overflow** to error * **Saturate on overflow** to error * **Inf or NaN block output** to error * **“rt” prefix for identifiers** to error * **Simulation range checking** to error |
| Rationale | Improve robustness of design. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related diagnostic settings for signal data**   For DO-178C/DO-331 check details, see “Check safety-related diagnostic settings for signal data”. |
| References | * DO-331, Section MB.6.4.2.2 'Robustness Test Cases' DO-331, Section MB.6.4.3 'Requirements-Based Testing Methods' DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' DO-331, Section MB.6.3.1.g 'Algorithms are accurate'  DO-331, Section MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' * MISRA C:2012, Dir 4.1 |
| See Also | “Model Configuration Parameters: Data Validity Diagnostics” |
| Last Changed | R2018a |

### Configuration Parameter Considerations – Optimizations

hisl\_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)

hisl\_0046: Configuration Parameters > Optimization > Block reduction

hisl\_0048: Configuration Parameters > Optimization > Application lifespan (days)

hisl\_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold

hisl\_0052: Configuration Parameters > Optimization > Data initialization

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

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

#### hisl\_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double)** | |
| Description | To support unambiguous behavior when using logical operators, relational operators, and the Combinatorial Logic block, | |
| A | Select **Implement logic signals as Boolean data (vs. double)** in the Configuration Parameters dialog box. |
| Notes | Selecting the **Implement logic signals as Boolean data (vs. double)** parameter, enables Boolean type checking, which produces an error when blocks that prefer Boolean inputs connect to double signals. This checking results in generating code that requires less memory. | |
| Rationale | A | Avoid ambiguous model behavior and optimize memory for generated code. |
| Model Advisor Checks | **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related optimization settings**  For DO-178C/DO-331 check details, see “Check safety-related optimization settings”. | |
| References | * DO-331, MB.6.3.1.e 'High-level requirements conform to standards' DO-331, MB.6.3,2.e 'Low-level requirements conform to standards' * MISRA C:2012, Rule 10.1 | |
| Last Changed | R2017b | |

#### 

#### hisl\_0046: Configuration Parameters > Optimization > Block reduction

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0046: Configuration Parameters > Optimization > Block reduction** | |
| Description | To support unambiguous presentation of the generated code and support traceability between a model and generated code, | |
| A | Clear the **Block reduction** parameter on the **Optimization** pane of the Configuration Parameters dialog box. |
| Notes | Selecting **Block reduction** might optimize blocks out of the code generated for a model. This results in requirements without associated code and violates traceability objectives. | |
| Rationale | A | Support unambiguous presentation of generated code. |
| A | Support traceability between a model and generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related optimization settings**   For DO-178C/DO-331 check details, see “Check safety-related optimization settings”. | |
| References | * DO-331, Section MB.6.3.4.e ‘Source code is traceable to low-level requirements’ | |
| See Also | “Block reduction” in the Simulink® documentation | |
| Last Changed | R2017b | |

#### 

#### hisl\_0048: Configuration Parameters > Optimization > Application lifespan (days)

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0048: Configuration Parameters > Optimization > Application lifespan (days)** | |
| Description | To support the robustness of systems that run continuously, in the Configuration Parameters dialog box, on the **Optimization** pane: | |
| A | Set **Application lifespan (days)** to Inf. |
| Notes | Embedded applications might run continuously. Do not assume a limited lifespan for timers and counters. When you set **Application lifespan (days)** to Inf, the simulation time is less than the application lifespan. | |
| Rationale | A | Support robustness of systems that run continuously. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related optimization settings**   For DO-178C/DO-331 check details, see “Check safety-related optimization settings”. | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' | |
| See Also | * “Application lifespan (days)” in the Simulink documentation * [“hisl\_0040: Configuration Parameters > Solver > Simulation time”](#_bookmark65) | |
| Last Changed | R2017b | |

#### 

#### hisl\_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold** | |
| Description | To support unambiguous code, set the minimum signal or parameter width for generating a for loop. In the Configuration Parameters dialog box, on the **Optimization** > **Signals and Parameters** pane, | |
| A | Set **Loop unrolling threshold** to 2 or greater. |
| Notes | The **Loop unrolling threshold** parameter specifies the array size at which the code generator begins to use a for loop, instead of separate assignment statements, to assign values to the elements of a signal or parameter array. The default value is 5. | |
| Rationale | A | Support unambiguous generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related optimization settings for loop unrolling threshold**   For DO-178C/DO-331 check details, see “Check safety-related optimization settings for loop unrolling threshold”. | |
| References | * DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' * MISRA C:2012, Rule 6.1 | |
| See Also | “Loop unrolling threshold” in the Simulink documentation | |
| Last Changed | R2017b | |

#### 

#### hisl\_0052: Configuration Parameters > Optimization > Data initialization

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0052: Configuration Parameters > Optimization > Data initialization** | |
| Description | To support complete definition of data and initialize internal and external data to zero, in the Configuration Parameters dialog box, on the **Optimization** pane, | |
| A | Clear **Remove root level I/O zero initialization**. |
| B | Clear **Remove internal data zero initialization**. |
| Note | Explicitly initialize all variables. If the run-time environment of the target system provides mechanisms to initialize all I/O and state variables, consider using the initialization of the target as an alternative to the suggested settings. | |
| Rationale | A, B | Support fully defined data in generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related optimization settings**   For DO-178C/DO-331 check details, see “Check safety-related optimization settings”. | |
| References | * DO-331, Section MB.6.3.3.b 'Software architecture is consistent' | |
| See Also | * Information about the following parameters in the Simulink documentation: * “Remove root level I/O zero initialization” * “Remove internal data zero initialization” | |
| Last Changed | R2017b | |

#### 

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

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0053: Configuration Parameters > Optimization > Remove code from floating- point to integer conversions that wraps out-of-range values** | |
| Description | To support verifiable code, In the Configuration Parameters dialog box, on the **Optimization** pane, | |
| A | Consider selecting **Remove code from floating-point to integer conversions that wraps out-of-range values**. |
| Notes | Avoid overflows as opposed to handling them with wrapper code. For blocks that have the parameter **Saturate on overflow** cleared, clearing **Remove code from floating-point to integer conversions that wraps out-of-range values** might add code that wraps out of range values, resulting in unreachable code that cannot be tested. | |
| Rationale | A | Support generation of code that can be verified. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related optimization settings**   For DO-178C/DO-331 check details, see “Check safety-related optimization settings”. | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate’  DO-331, Section MB.6.3.2.g 'Algorithms are accurate’ * MISRA C:2012, Rule 2.1 | |
| See Also | “Remove code from floating-point to integer conversions that wraps out-of- range values” in the Simulink documentation | |
| Last Changed | R2017b | |

#### 

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

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions** | |
| Description | To support the robustness of the operations, in the Configuration Parameters dialog box, on the **Optimization** pane, | |
| A | Clear **Remove code that protects against division arithmetic exceptions**. |
| Note | Avoid division-by-zero exceptions. If you clear **Remove code that protects against division arithmetic exceptions**, the code generator produces code that guards against division by zero for fixed-point data. | |
| Rationale | A | Protect against divide-by-zero exceptions for fixed-point code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related optimization settings**   For DO-178C/DO-331 check details, see “Check safety-related optimization settings”. | |
| References | * DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' * MISRA C:2012, Dir 4.1 | |
| See Also | “Remove code that protects against division arithmetic exceptions” in the Simulink documentation | |
| Last Changed | R2017b | |

#### 

### Configuration Parameter Considerations – Model Referencing

hisl\_0037: Configuration Parameters > Model Referencing

#### hisl\_0037: Configuration Parameters > Model Referencing

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0037: Configuration Parameters > Model Referencing** | |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Model Referencing** pane, set the **Options for all referenced models** and **Options for referencing this model** parameters as follows: | |
| A | Set **Rebuild** to either Never or If any changes detected. |
| B | Set **Never rebuild diagnostic** to Error if rebuild required. This diagnostic parameter is available only if **Rebuild** is set to Never. |
| 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related model referencing settings**   For DO-178C/DO-331 check details, see “Check safety-related model referencing settings”. | |
| References | * DO-331, Section MB.6.3.1.b 'High-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.3.b 'Software architecture is consistent' | |
| Last Changed | R2017b | |

### Configuration Parameter Considerations – Code Generation

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 > Symbols

#### hisl\_0038: Configuration Parameters > Code Generation > Comments

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0038: Configuration Parameters > Code Generation > Comments** | |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Code Generation** > **Comments** pane, set the **Overall control**, **Auto generated comments**, and **Custom comments** parameters as follows: | |
| A | Select **Include comments**. |
| B | Select **Simulink block comments**. |
| C | Select **Show eliminated blocks**. |
| D | Select **Verbose comments for SimulinkGlobal storage class**. |
| E | Select **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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related code generation settings**   For DO-178C/DO-331 check details, see “Check safety-related code generation settings”. | |
| References | * DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' | |
| Last Changed | R2017b | |

#### 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** > **Code Style** pane, set the **Software environment**, **Code interface**, and **Data exchange** 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 **Suppress error status 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 | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related code generation settings**   For DO-178C/DO-331 check details, see “Check safety-related code generation settings”. |
| References | * DO-331, Section MB.6.3.1.c 'High-level requirements are compatible with target computer' DO-331, Section MB.6.3.2.c 'Low-level requirements are compatible with target computer' |
| Last Changed | R2017b |

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

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0047: Configuration Parameters > Code Generation > Code Style** | |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, set the **Code Style** parameters as follows: | |
| A | Set **Parenthesis level** to Maximum (Specify precedence with parenthesis). |
| B | Select **Preserve operand order in expression**. |
| C | Select **Preserve condition expression in if statement**. |
| Rationale | A | To prevent unexpected results. |
| B, C | To improve traceability of the generated code. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related code generation settings**   For DO-178C/DO-331 check details, see “Check safety-related code generation settings”. | |
| References | * DO-331, Section MB.6.3.1.c 'High-level requirements are compatible with target computer' DO-331, Section MB.6.3.2.c 'Low-level requirements are compatible with target computer' DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements' * MISRA C:2012, Rule 12.1 | |
| Last Changed | R2017b | |

#### hisl\_0049: Configuration Parameters > Code Generation > Symbols

|  |  |  |
| --- | --- | --- |
| **ID: Title** | **hisl\_0049: Configuration Parameters > Code Generation > Symbols** | |
| Description | For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Code Generation** > **Symbols** pane, set the **Auto-generated identifier naming rules** parameters as follows: | |
| A | Set **Minimal mangle length** to 4 or greater. |
| Rationale | A | To minimize the likelihood that parameter and signal names will change during code generation when the model changes. Thus the option can decrease the effort to perform code review. |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Configuration** > **Check safety-related code generation settings**   For DO-178C/DO-331 check details, see “Check safety-related code generation settings”. | |
| References | * DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements’ | |
| Last Changed | R2017b | |

### 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** | **hisf\_0064: Shift operations for Stateflow data to improve code compliance** | |
| Description | To improve MISRA C:2012 compliance of the generated code, set the following model configuration parameters as specified: | |
| **Configuration Parameter** | **Value** |
| **Optimization** > **Simulation and code generation**: | |
| **Use division for fixed-point net slope computation** | on or Use division for reciprocals of integers only |
| **Optimization** > **Signals and Parameters** > **Code generation**: | |
| **Bitfield declarator type specifier** | uint\_T if any of the following Optimization parameters are enabled:   * **Optimization** > **Signals** and **Parameters** > **Code Generation** > **Pack Boolean data into bitfields** * **Optimization** > **Stateflow** > **Code Generation** > **Use bitsets for storing state configuration** * **Optimization** > **Stateflow** > **Code Generation** > **Use bitsets for storing Boolean data** |
| **Diagnostics**: | |
| **Model Verification block enabling** | Disable all |
| **Diagnostics** > **Data Validity** > **Signals**: | |
| **Wrap on overflow** | warning or error |
| **Inf or NaN block output** | warning or error |
| **Hardware Implementation** > **Device details**: | |
| **Production hardware signed integer division rounds to** | Zero or Floor |
| **Simulation Target**: | |
| **Dynamic memory allocation in MATLAB Function blocks** | Cleared (off) |
| **Code Generation** > **Target selection**: | |
| **System target file** | ERT-based target |
| **Code Generation** > **Symbols** > **Auto-generated identifier naming rules**: | |
| **Maximum identifier length** | This should be set to the implementation dependent limit. The default is 31. |
| **System-generated identifiers** | Shortened |

|  |  |  |
| --- | --- | --- |
|  | **Code Generation** > **Interface** > **Software environment**: | |
| **Code replacement library** | None or AUTOSAR 4.0 |
| **Shared code placement** | Shared location |
| **Support non-finite numbers** | Cleared (off) |
| **Support complex numbers** | Cleared (off) if you do not need complex number support |
| **Support continuous time** | Cleared (off) |
| **Code Generation** > **Code Style** > **Code Style**: | |
| **Parentheses level** | Maximum (Specify precedence with parentheses) |
| **Preserve static keyword in function declarations** | Selected (on)  Select only when **Code Generation** > **Code Placement** > **Code packaging** > **File packaging format** is set to Compact or CompactWithDataFile. |
| **Replace multiplications by powers of two with signed bitwise shifts** | Cleared (off) |
| **Casting modes** | Standards Compliant |
| **Code Generation** > **Interface** > **Advanced parameters**: | |
| **Generate shared constants** | Cleared (off) |
| **Mat-file logging** | Cleared (off) |
| **Standard math library** | C89/C90 (ANSI) or C99 (ISO) depending on toolchain |
| **Support non-inlined S-functions** | Cleared (off) |
| **Use dynamic memory allocation for model initialization** | Cleared (off)  Select only when **Code Generation** > **Interface** > **Code interface** > **Code interface packaging** is set to Reusable function. |
| Rationale | Improve MISRA C:2012 compliance of the generated code. | |
| Model Advisor Checks | * **By Task** > **Modeling Standards for DO-178C/DO-331** > **High-Integrity Systems** > **Code** > **Check configuration parameters for MISRA C:2012**   For DO-178C/DO-331 check details, see “Check configuration parameters for MISRA C:2012”. | |
| References | * MISRA C:2012 | |
| Last Changed | R2017b | |

## Simulink Model Advisor DO-178C/DO-331 Checks

Apply the following checks documented in "*DO Qualification Kit Simulink*® *Verification and Validation™ DO-178C/DO-331 Checks and Model Advisor User Information*":

Check safety-related optimization settings

Check safety-related model referencing settings

Check safety-related code generation settings

Check safety-related diagnostic settings for solvers

Check safety-related solver settings for simulation time

Check safety-related solver settings for solver options

Check safety-related solver settings for tasking and sample-time

Check safety-related diagnostic settings for sample time

Check safety-related diagnostic settings for signal data

Check safety-related diagnostic settings for parameters

Check safety-related diagnostic settings for data used for debugging

Check safety-related diagnostic settings for type conversions

Check safety-related diagnostic settings for signal connectivity

Check safety-related diagnostic settings for bus connectivity

Check safety-related diagnostic settings that apply to function-call connectivity

Check safety-related diagnostic settings for compatibility

Check safety-related diagnostic settings for model initialization

Check safety-related diagnostic settings for model referencing

Check safety-related diagnostic settings for saving

Check safety-related diagnostic settings for Merge blocks

Check safety-related diagnostic settings for Stateflow

Check safety-related optimization settings for loop unrolling threshold

Check configuration parameters for MISRA C:2012

#### Applicable qualifiable Model Advisor checks

|  |  |
| --- | --- |
| **Model Advisor Checks** | **Applicable High-Integrity System Modeling Guidelines** |
| Check safety-related optimization settings | * hisl\_0045: Configuration Parameters > Optimization > Implement logic signals as Boolean data (vs. double) * hisl\_0046: Configuration Parameters > Optimization > Block reduction * hisl\_0048: Configuration Parameters > Optimization > Application lifespan (days) * hisl\_0052: Configuration Parameters > Optimization > Data initialization * hisl\_0053: Configuration Parameters > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values * hisl\_0054: Configuration Parameters > Optimization > Remove code that protects against division arithmetic exceptions |
| Check safety-related model referencing settings | hisl\_0037: Configuration Parameters > Model Referencing |
| Check safety-related code generation settings | * 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 > Symbols |
| Check safety-related diagnostic settings for solvers | hisl\_0043: Configuration Parameters > Diagnostics > Solver |
| Check safety-related solver settings for simulation time | hisl\_0040: Configuration Parameters > Solver > Simulation time |
| Check safety-related solver settings for solver options | hisl\_0041: Configuration Parameters > Solver > Solver options |
| Check safety-related solver settings for tasking and sample-time | hisl\_0042: Configuration Parameters > Solver > Tasking and sample time options |
| Check safety-related diagnostic settings for sample time | hisl\_0044: Configuration Parameters > Diagnostics > Sample Time |
| Check safety-related diagnostic settings for signal data | hisl\_0314: Configuration Parameters > Diagnostics > Data Validity > Signals |
| Check safety-related diagnostic settings for parameters | hisl\_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters |
| Check safety-related diagnostic settings for data used for debugging | hisl\_0305: Configuration Parameters > Diagnostics > Debugging |
| Check safety-related diagnostic settings for type conversions | hisl\_0309: Configuration Parameters > Diagnostics > Type Conversion |
| Check safety-related diagnostic settings for signal connectivity | hisl\_0306: Configuration Parameters > Diagnostics > Connectivity> Signals |
| Check safety-related diagnostic settings for bus connectivity | hisl\_0307: Configuration Parameters > Diagnostics > Connectivity> Buses |
| Check safety-related diagnostic settings that apply to function-call connectivity | hisl\_0308: Configuration Parameters > Diagnostics > Connectivity> Function calls |
| Check safety-related diagnostic settings for compatibility | hisl\_0301: Configuration Parameters > Diagnostics > Compatibility |
| Check safety-related diagnostic settings for model initialization | hisl\_0304: Configuration Parameters > Diagnostics > Model Initialization |
| Check safety-related diagnostic settings for model referencing | hisl\_0310: Configuration Parameters > Diagnostics > Model Referencing |
| Check safety-related diagnostic settings for saving | hisl\_0036: Configuration Parameters > Diagnostics > Saving |
| Check safety-related diagnostic settings for Merge blocks | hisl\_0303: Configuration Parameters > Diagnostics > Merge block |
| Check safety-related diagnostic settings for Stateflow | hisl\_0311: Configuration Parameters > Diagnostics > Stateflow |
| Check safety-related optimization settings for loop unrolling threshold | hisl\_0051: Configuration Parameters > Optimization > Signals and Parameters > Loop unrolling threshold |
| Check configuration parameters for MISRA C:2012 | hisl\_0060: Configuration parameters that improve MISRA C:2012 compliance |

## Company-Specific Constraints

*<Please 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 (DO-331MB.11.23.e).

As defined in DO-331 MB.1.6.2, design models contain the software architecture and the software low-level requirements. Low-level requirements are expressed by the subset of MATLAB, Simulink, and Stateflow, as defined in Chapter 3 Style Guidelines and Complexity Restrictions. See Chapter 7 Additional Model Elements for model elements that do not express requirements.

Use the Simulink Verification and Validation Requirements Management Interface (RMI) 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. The traceability between the design model and the generated source code is automatically checked using the Simulink Code Inspector™ and by reviewing the generated Code Inspector Report.

# 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 (DO-331MB.11.23.f).

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 <*Please 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 (as defined in DO-331 Section 6.7) is used to verify that all derived requirements were properly marked and analyzed. As described in DO-331 Table MB.6-1, additional test cases will be created, based on the derived requirements contained in the design model. The additional test cases have to 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 (DO-331MB.11.23.g).

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

*<Please 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 (DO-331MB.11.23.h).

*<Please 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](http://www.mathworks.com/help/releases/R2015b/index.html). People involved in a software development project can use the documentation to understand the content of a design model.