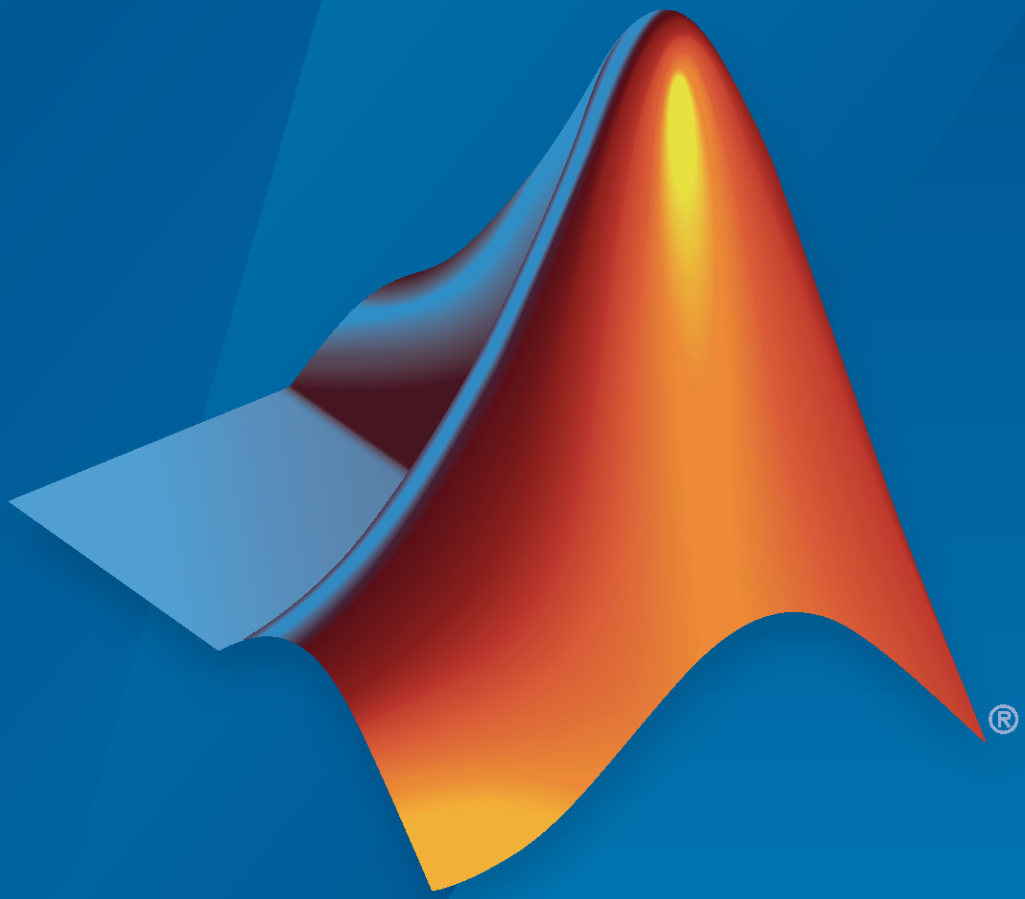
Simulink®

Modeling Guidelines for High-Integrity Systems



MATLAB® & SIMULINK®

R2023a

MathWorks®

How to Contact MathWorks

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

Phone: 508-647-7000

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

Modeling Guidelines for High-Integrity Systems

Revision History

September 2009	Online only	New for Version 1.0 (Release 2009b)
April 2010	Online only	Revised for Version 1.1 (Release 2010a)
September 2010	Online only	Revised for Version 1.2 (Release 2010b)
April 2011	Online only	Revised for Version 1.3 (Release 2011a)
September 2011	Online only	Revised for Version 1.4 (Release 2011b)
March 2012	Online only	Revised for Version 1.5 (Release 2012a)
September 2012	Online only	Revised for Version 1.6 (Release 2012b)
March 2013	Online only	Revised for Version 1.7 (Release 2013a)
September 2013	Online only	Revised for Version 1.8 (Release 2013b)
March 2014	Online only	Revised for Version 1.9 (Release 2014a)
October 2014	Online only	Revised for Version 1.10 (Release 2014b)
March 2015	Online only	Revised for Version 1.11 (Release 2015a)
September 2015	Online only	Revised for Version 1.12 (Release 2015b)
March 2016	Online only	Revised for Version 1.13 (Release 2016a)
September 2016	Online only	Revised for Version 1.14 (Release 2016b)
March 2017	Online only	Revised for Version 1.15 (Release 2017a)
September 2017	Online only	Revised for Version 1.16 (Release 2017b)
March 2018	Online only	Revised for Version 1.17 (Release 2018a)
September 2018	Online only	Revised for Version 1.18 (Release 2018b)
March 2019	Online only	Revised for Version 1.19 (Release 2019a)
September 2019	Online only	Revised for Version 1.20 (Release 2019b)
March 2020	Online only	Revised for Version 1.21 (Release 2020a)
September 2020	Online only	Revised for Version 1.22 (Release 2020b)
March 2021	Online only	Revised for Version 1.23 (Release 2021a)
September 2021	Online only	Revised for Version 1.24 (Release 2021b)
March 2022	Online only	Revised for Version 1.25 (Release 2022a)
September 2022	Online only	Revised for Version 1.26 (Release 2022b)
March 2023	Online only	Revised for Version 1.27 (Release 2023a)

Contents

1 Introduction

2 Simulink Block Considerations

5 Configuration Parameter Considerations

6 Naming Considerations

7 MISRA C:2012 Compliance Considerations

8 Requirements Considerations

Introduction

Motivation

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

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

- DO-178C/DO-331, *Software Considerations in Airborne Systems and Equipment Certification*
- DO-254, *Design Assurance Guidance for Airborne Electronic Hardware*
- IEC 61508, *Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems*
- IEC 62304, *Medical Device Software – Software Life Cycle Processes*
- ISO 26262, *Road vehicles — Functional Safety*
- EN 50128/EN 50657, *Railway applications - Communication, Signalling and Processing Systems - Software for Railway Control and Protection Systems*
- ISO 25119, *Tractors And Machinery For Agriculture And Forestry — Safety-Related Parts Of Control Systems*
- MISRA C, *Use of the C Language in Critical Systems*

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

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

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

Guideline Template

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

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

Model Advisor Checks for High-Integrity Systems Modeling Guidelines

The Simulink Check™ Model Advisor provides High-Integrity System Modelling checks that you can use to verify a compliance with safety standards, including:

- DO-178C / DO-331
- IEC 61508
- IEC 62304
- ISO 26262
- EN 50128 (and EN 50657)
- “ISO 25119 Standard” (Embedded Coder)

The high-integrity system modeling guidelines and their corresponding checks are summarized in this table. For the guidelines that do not have Model Advisor checks, it is not possible to automate checking of the guideline. Guidelines without a corresponding check are noted as not applicable.

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
“Check usage of lookup table blocks” (Simulink Check)	`mathworks.hism.hisl_0033`	“hisl_0033: Usage of Lookup Table blocks” on page 2-40
“Check for inconsistent vector indexing methods” (Simulink Check)	`mathworks.hism.hisl_0021`	“hisl_0021: Consistent vector indexing method” on page 2-30
“Check usage of variant blocks” (Simulink Check)	`mathworks.hism.hisl_0021`	“hisl_0023: Verification of variant blocks” on page 2-32
“Check for root Inports with missing properties” (Simulink Check)	`mathworks.hism.hisl_0024`	“hisl_0024: Inport interface definition” on page 2-22
“Check usage of Relational Operator blocks” (Simulink Check)	`mathworks.hism.hisl_0017`	“hisl_0017: Usage of blocks that compute relational operators (2)” on page 2-37
“Check relational comparisons on floating-point signals” (Simulink Check)	`mathworks.hism.hisl_0016`	“hisl_0016: Usage of blocks that compute relational operators” on page 2-35
“Check usage of Logical Operator blocks” (Simulink Check)	`mathworks.hism.hisl_0018`	“hisl_0018: Usage of Logical Operator block” on page 2-38
“Check usage of While Iterator blocks” (Simulink Check)	`mathworks.hism.hisl_0006`	“hisl_0006: Usage of While Iterator blocks” on page 2-17
“Check usage of For and While Iterator subsystems” (Simulink Check)	`mathworks.hism.hisl_0007`	“hisl_0007: Usage of For Iterator or While Iterator subsystems” on page 2-18
“Check usage of For Iterator blocks” (Simulink Check)	`mathworks.hism.hisl_0008`	“hisl_0008: Usage of For Iterator Blocks” on page 2-18

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
"Check usage of If blocks and If Action Subsystem blocks" (Simulink Check)	`mathworks.hism.hisl_0010`	"hisl_0010: Usage of If blocks and If Action Subsystem blocks" on page 2-19
"Check usage of Switch Case blocks and Switch Case Action Subsystem blocks" (Simulink Check)	`mathworks.hism.hisl_0011`	"hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks" on page 2-21
"Check safety-related optimization settings for logic signals" (Simulink Check)	`mathworks.hism.hisl_0045`	"hisl_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)" on page 5-5
"Check safety-related block reduction optimization settings" (Simulink Check)	`mathworks.hism.hisl_0046`	"hisl_0046: Configuration Parameters > Simulation Target > Block reduction" on page 5-21
"Check safety-related optimization settings for application lifespan" (Simulink Check)	`mathworks.hism.hisl_0048`	"hisl_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)" on page 5-5
"Check safety-related optimization settings for data initialization" (Simulink Check)	`mathworks.hism.hisl_0052`	"hisl_0052: Configuration Parameters > Code Generation > Optimization > Data initialization" on page 5-22
"Check safety-related optimization settings for data type conversions" (Simulink Check)	`mathworks.hism.hisl_0053`	"hisl_0053: Configuration Parameters > Code Generation > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values" on page 5-23
"Check safety-related optimization settings for division arithmetic exceptions" (Simulink Check)	`mathworks.hism.hisl_0054`	"hisl_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions" on page 5-24
"Check safety-related code generation settings for comments" (Simulink Check)	`mathworks.hism.hisl_0038`	"hisl_0038: Configuration Parameters > Code Generation > Comments" on page 5-26
"Check safety-related code generation interface settings" (Simulink Check)	`mathworks.hism.hisl_0039`	"hisl_0039: Configuration Parameters > Code Generation > Interface" on page 5-26
"Check safety-related code generation settings for code style" (Simulink Check)	`mathworks.hism.hisl_0047`	"hisl_0047: Configuration Parameters > Code Generation > Code Style" on page 5-27

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
“Check safety-related code generation identifier settings” (Simulink Check)	`mathworks.hism.hisl_0049`	“hisl_0049: Configuration Parameters > Code Generation > Identifiers” on page 5-28
“Check usage of Abs blocks” (Simulink Check)	`mathworks.hism.hisl_0001`	“hisl_0001: Usage of Abs block” on page 2-2
“Check usage of remainder and reciprocal operations” (Simulink Check)	`mathworks.sldv.hism.hisl_0002`	“hisl_0002: Usage of remainder and reciprocal operations” on page 2-3
“Check usage of square root operations” (Simulink Check)	`mathworks.hism.hisl_0003`	“hisl_0003: Usage of square root operations” on page 2-5
“Check usage of log and log10 operations” (Simulink Check)	`mathworks.sldv.hism.hisl_0004`	“hisl_0004: Usage of natural logarithm and base 10 logarithm operations” on page 2-6
“Check usage of Assignment blocks” (Simulink Check)	`mathworks.hism.hisl_0029`	“hisl_0029: Usage of Assignment blocks” on page 2-10
“Check usage of Signal Routing blocks” (Simulink Check)	`mathworks.hism.hisl_0034`	“hisl_0034: Usage of Signal Routing blocks” on page 2-32
“Check for root Inports with missing range definitions” (Simulink Check)	`mathworks.hism.hisl_0025`	“hisl_0025: Design min/max specification of input interfaces” on page 2-23
“Check for root Outports with missing range definitions” (Simulink Check)	`mathworks.hism.hisl_0026`	“hisl_0026: Design min/max specification of output interfaces” on page 2-24
“Check state machine type of Stateflow charts” (Simulink Check)	`mathworks.hism.hisf_0001`	“hisf_0001: State Machine Type” on page 3-2
“Check Stateflow charts for transition paths that cross parallel state boundaries” (Simulink Check)	`mathworks.hism.hisf_0013`	“hisf_0013: Usage of transition paths (crossing parallel state boundaries)” on page 3-6
“Check Stateflow charts for ordering of states and transitions” (Simulink Check)	`mathworks.hism.hisf_0002`	“hisf_0002: User-specified state/transition execution order” on page 3-2
“Check Stateflow debugging options” (Simulink Check)	`mathworks.hism.hisf_0011`	“hisf_0011: Stateflow debugging settings” on page 3-3
“Check Stateflow charts for uniquely defined data objects” (Simulink Check)	`mathworks.hism.hisl_0061`	“hisl_0061: Unique identifiers for clarity” on page 7-4
“Check Stateflow charts for strong data typing” (Simulink Check)	`mathworks.hism.hisf_0015`	“hisf_0015: Strong data typing (casting variables and parameters in expressions)” on page 3-9

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
“Check assignment operations in Stateflow Charts” (Simulink Check)	`mathworks.hism.hisf_0065`	“hisf_0065: Type cast operations in Stateflow to improve code compliance” on page 7-18
“Check Stateflow charts for unary operators” (Simulink Check)	`mathworks.hism.hisf_0211`	“hisf_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance” on page 7-18
“Check for MATLAB Function interfaces with inherited properties” (Simulink Check)	`mathworks.hism.himl_0002`	“himl_0002: Strong data typing at MATLAB function boundaries” on page 4-3
“Check MATLAB Function metrics” (Simulink Check)	`mathworks.hism.himl_0003`	“himl_0003: Complexity of user-defined MATLAB Functions” on page 4-4
“Check MATLAB Code Analyzer messages” (Simulink Check)	`mathworks.hism.himl_0004`	“himl_0004: MATLAB Code Analyzer recommendations for code generation” on page 4-6
“Check safety-related model referencing settings” (Simulink Check)	`mathworks.hism.hisl_0037`	“hisl_0037: Configuration Parameters > Model Referencing” on page 5-20
“Check safety-related diagnostic settings for solvers” (Simulink Check)	`mathworks.hism.hisl_0043`	“hisl_0043: Configuration Parameters > Diagnostics > Solver” on page 5-7
“Check safety-related solver settings for simulation time” (Simulink Check)	`mathworks.hism.hisl_0040`	“hisl_0040: Configuration Parameters > Solver > Simulation time” on page 5-2
“Check safety-related solver settings for solver options” (Simulink Check)	`mathworks.hism.hisl_0041`	“hisl_0041: Configuration Parameters > Solver > Solver options” on page 5-2
“Check safety-related solver settings for tasking and sample-time” (Simulink Check)	`mathworks.hism.hisl_0042`	“hisl_0042: Configuration Parameters > Solver > Tasking and sample time options” on page 5-3
“Check safety-related diagnostic settings for sample time” (Simulink Check)	`mathworks.hism.hisl_0044`	“hisl_0044: Configuration Parameters > Diagnostics > Sample Time” on page 5-9
“Check safety-related diagnostic settings for parameters” (Simulink Check)	`mathworks.hism.hisl_0302`	“hisl_0302: Configuration Parameters > Diagnostics > Data Validity > Parameters” on page 5-11
“Check safety-related diagnostic settings for data used for debugging” (Simulink Check)	`mathworks.hism.hisl_0305`	“hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging” on page 5-13

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
“Check safety-related diagnostic settings for data store memory” (Simulink Check)	`mathworks.hism.hisl_0013`	“hisl_0013: Usage of data store memory” on page 2-27
“Check safety-related diagnostic settings for type conversions” (Simulink Check)	`mathworks.hism.hisl_0309`	“hisl_0309: Configuration Parameters > Diagnostics > Type Conversion” on page 5-15
“Check safety-related diagnostic settings for signal connectivity” (Simulink Check)	`mathworks.hism.hisl_0306`	“hisl_0306: Configuration Parameters > Diagnostics > Connectivity > Signals” on page 5-13
“Check safety-related diagnostic settings for bus connectivity” (Simulink Check)	`mathworks.hism.hisl_0307`	“hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses” on page 5-14
“Check safety-related diagnostic settings that apply to function-call connectivity” (Simulink Check)	`mathworks.hism.hisl_0308`	“hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls” on page 5-15
“Check safety-related diagnostic settings for compatibility” (Simulink Check)	`mathworks.hism.hisl_0301`	“hisl_0301: Configuration Parameters > Diagnostics > Compatibility” on page 5-11
“Check safety-related diagnostic settings for model initialization” (Simulink Check)	`mathworks.hism.hisl_0304`	“hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model initialization” on page 5-13
“Check safety-related diagnostic settings for model referencing” (Simulink Check)	`mathworks.hism.hisl_0310`	“hisl_0310: Configuration Parameters > Diagnostics > Model Referencing” on page 5-16
“Check safety-related diagnostic settings for saving” (Simulink Check)	`mathworks.hism.hisl_0036`	“hisl_0036: Configuration Parameters > Diagnostics > Saving” on page 5-7
“Check safety-related diagnostic settings for Merge blocks” (Simulink Check)	`mathworks.hism.hisl_0303`	“hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge blocks” on page 5-12
“Check safety-related diagnostic settings for Stateflow” (Simulink Check)	`mathworks.hism.hisl_0311`	“hisl_0311: Configuration Parameters > Diagnostics > Stateflow” on page 5-16
“Check model object names” (Simulink Check)	`mathworks.hism.hisl_0032`	“hisl_0032: Model element names” on page 6-3
“Check for model elements that do not link to requirements” (Simulink Check)	`mathworks.hism.hisl_0070`	“hisl_0070: Placement of requirement links in a model” on page 8-2

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
“Check for inappropriate use of transition paths” (Simulink Check)	mathworks.hism.hisf_0014	“hisf_0014: Usage of transition paths (passing through states)” on page 3-8
“Check usage of bit operation blocks” (Simulink Check)	mathworks.hism.hisl_0019	“hisl_0019: Usage of bitwise operations” on page 2-38
“Check data types for blocks with index signals” (Simulink Check)	mathworks.hism.hisl_0022	“hisl_0022: Data type selection for index signals” on page 2-31
“Check model file name” (Simulink Check)	mathworks.hism.hisl_0031	“hisl_0031: Model file names” on page 6-2
“Check if/elseif/else patterns in MATLAB Function blocks” (Simulink Check)	mathworks.hism.hisl_0022	“himl_0006: MATLAB code if / elseif / else patterns” on page 4-8
“Check switch statements in MATLAB Function blocks” (Simulink Check)	mathworks.hism.himl_0007	“himl_0007: MATLAB code switch / case / otherwise patterns” on page 4-10
“Check global variables in graphical functions” (Simulink Check)	mathworks.hism.hisl_0062	“hisl_0062: Global variables in graphical functions” on page 7-7
“Check for length of user-defined object names” (Simulink Check)	mathworks.hism.hisl_0063	“hisl_0063: Length of user-defined object names to improve MISRA C:2012 compliance” on page 7-9
“Check usage of Merge blocks” (Simulink Check)	mathworks.hism.hisl_0015	“hisl_0015: Usage of Merge blocks” on page 2-28
“Check usage of conditionally executed subsystems” (Simulink Check)	mathworks.hism.hisl_0012	“hisl_0012: Usage of conditionally executed subsystems” on page 2-22
“Check usage of standardized MATLAB function headers” (Simulink Check)	mathworks.hism.himl_0001	“himl_0001: Usage of standardized MATLAB function headers” on page 4-2
“Check usage of relational operators in MATLAB Function blocks” (Simulink Check)	mathworks.hism.himl_0008	“himl_0008: MATLAB code relational operator data types” on page 4-12
“Check usage of logical operators and functions in MATLAB Function blocks” (Simulink Check)	mathworks.hism.himl_0010	“himl_0010: MATLAB code with logical operators and functions” on page 4-13
“Check type and size of condition expressions” (Simulink Check)	mathworks.hism.himl_0011	“himl_0011: Data type and size of condition expressions” on page 4-16
“Check naming of ports in Stateflow charts” (Simulink Check)	mathworks.hism.hisf_0016	“hisf_0016: Stateflow port names” on page 3-10

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
“Check scoping of Stateflow data objects” (Simulink Check)	`mathworks.hism.hisf_0017`	“hisf_0017: Stateflow data object scoping” on page 3-11
“Check usage of Gain blocks” (Simulink Check)	`mathworks.hism.hisl_0066`	“hisl_0066: Usage of Gain blocks” on page 2-13
“Check for divide-by-zero calculations” (Simulink Check)	`mathworks.hism.hisl_0067`	“hisl_0067: Protect against divide-by-zero calculations” on page 2-14
“Check data type of loop control variables” (Simulink Check)	`mathworks.hism.hisl_0102`	“hisl_0102: Data type of loop control variables to improve MISRA C:2012 compliance” on page 7-14
“Check configuration parameters for MISRA C:2012” (Simulink Check)	`mathworks.misra.CodeGenSettings`	“hisl_0060: Configuration parameters that improve MISRA C:2012 compliance” on page 7-15

“Check for blocks not recommended for C/C++ production code deployment” (Simulink Check)
 	
“Check for blocks not recommended for MISRA C:2012” (Simulink Check)
	`mathworks.hism.hisl_0020`
	
	`mathworks.misra.BlkSupport`
		“hisl_0020: Blocks not recommended for MISRA C:2012 compliance” on page 7-11

“Check safety-related optimization settings for specified minimum and maximum values” (Simulink Check)	`mathworks.hism.hisl_0056`	“hisl_0056: Configuration Parameters > Code Generation > Optimization > Optimize using the specified minimum and maximum values” on page 5-25
“Check usage of Reciprocal Sqrt blocks” (Simulink Check)	`mathworks.hism.hisl_0028`	“hisl_0028: Usage of Reciprocal Square Root blocks” on page 2-5
“Check safety-related settings for hardware implementation” (Simulink Check)	`mathworks.hism.hisl_0071`	“hisl_0071: Configuration Parameters > Hardware Implementation >Inconsistent hardware implementation settings” on page 5-19
“Check usage of recursions” (Simulink Check)	`mathworks.hism.hisf_0004`	“hisf_0004: Protect against recursive function calls to improve code compliance” on page 3-5
“Check MATLAB functions not supported for code generation” (Simulink Check)	`mathworks.hism.himl_0012`	“himl_0012: Usage of MATLAB functions for code generation” on page 4-14
“Metrics for generated code complexity” (Simulink Check)	`mathworks.hism.himl_0013`	“himl_0013: Limitation of built-in MATLAB Function complexity” on page 4-14

Model Advisor Check	Check ID	High-Integrity System Modeling Guidelines
“Check for parameter tunability ignored for referenced models” (Simulink Check)	`mathworks.hism.hisl_0072`	“hisl_0072: Usage of tunable parameters for referenced models” on page 2-43
“Check usage of bit-shift operations” (Simulink Check)	`mathworks.hism.hisl_0073`	“hisl_0073: Usage of bit-shift operations” on page 2-44
“Check safety-related diagnostic settings for variants” (Simulink Check)	`mathworks.hism.hisl_0074`	“hisl_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants” on page 5-29
“Check for disabled and parameterized library links” (Simulink Check)	`mathworks.hism.hisl_0075`	“hisl_0075: Usage of library links” on page 5-29
“Check for unreachable and dead code” (Simulink Check)	`mathworks.hism.hisl_0101`	“hisl_0101: Avoid operations that result in dead logic to improve code compliance” on page 7-12
“Check for root Outports with missing properties” (Simulink Check)	`mathworks.hism.hisl_0077`	“hisl_0077: Outport interface definition” on page 2-25

See Also

- “Check Your Model Using the Model Advisor”
- “High-Integrity System Modeling”

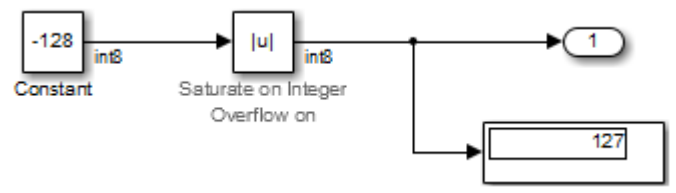
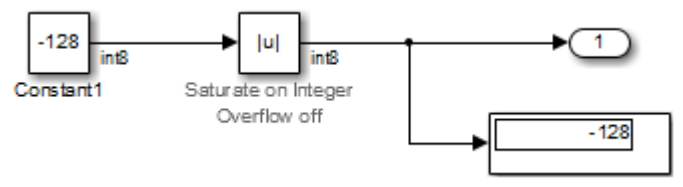
Simulink Block Considerations

Math Operations

In this section...

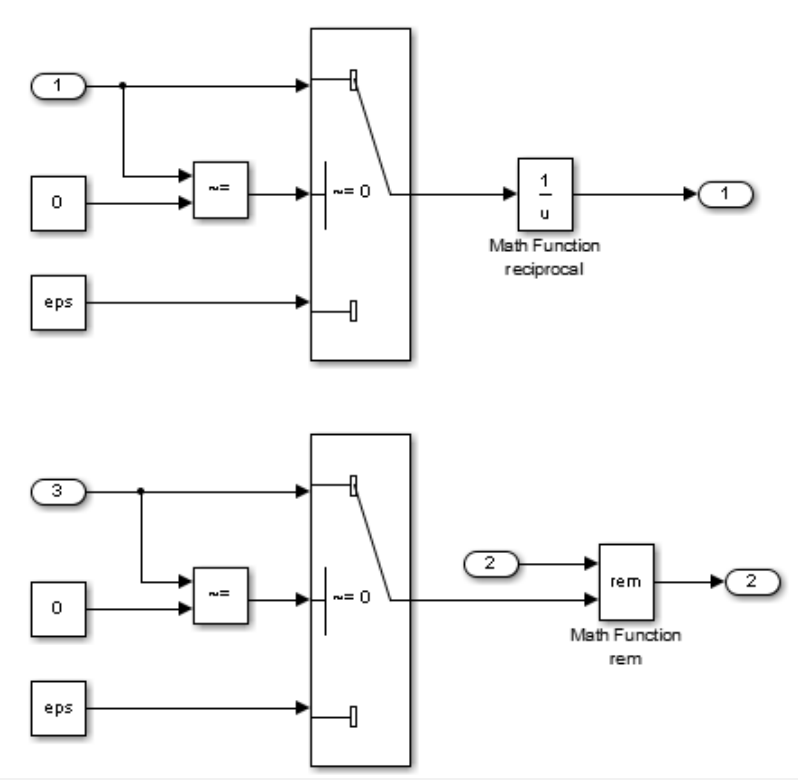
hisl_0001: Usage of Abs block

ID: Title	hisl_0001: Usage of Abs block	
Description	To support robustness of generated code, when using the Abs block,	
	A	Avoid Boolean and unsigned data types as inputs to the Abs block.
	B	Select block parameter **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	“Check usage of Abs blocks” (Simulink Check)	

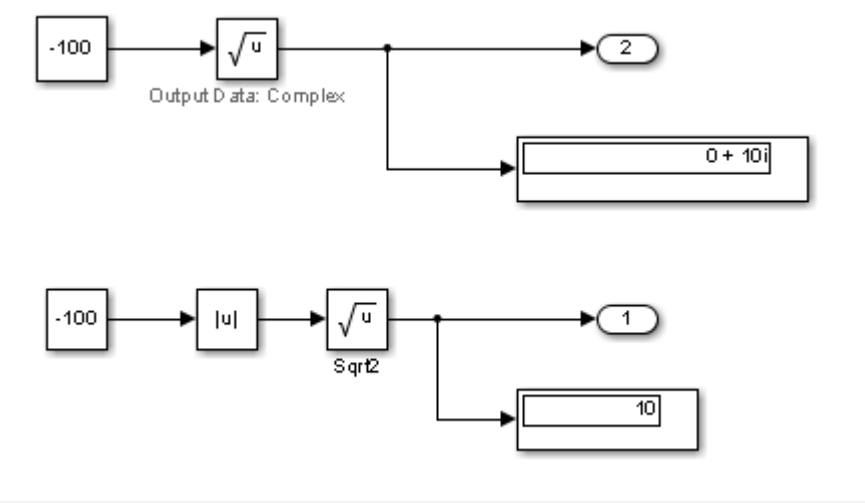
ID: Title	hisl_0001: Usage of Abs block
References	• IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' IEC 61508-3, Table B.8 (3) 'Control Flow Analysis' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 7 (1f) 'Control flow analysis' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.19 (3) 'Control Flow Analysis' • DO-331, Section MB.6.3.2.d 'Low-level requirements are verifiable' DO-331, Section MB.6.3.2.g – 'Algorithms are accurate' • MISRA C:2012, Dir 4.1 • INT32-C. Ensure that operations on signed integers do not result in overflow
Last Changed	R2021b
Examples	 **Recommended**  **Not Recommended**

hisl_0002: Usage of remainder and reciprocal operations

ID: Title	hisl_0002: Usage of remainder and reciprocal operations	
Description	To support robustness of generated code, when using the Math Function block with remainder-after-division (`rem`) or reciprocal (`reciprocal`) operations:	
	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.

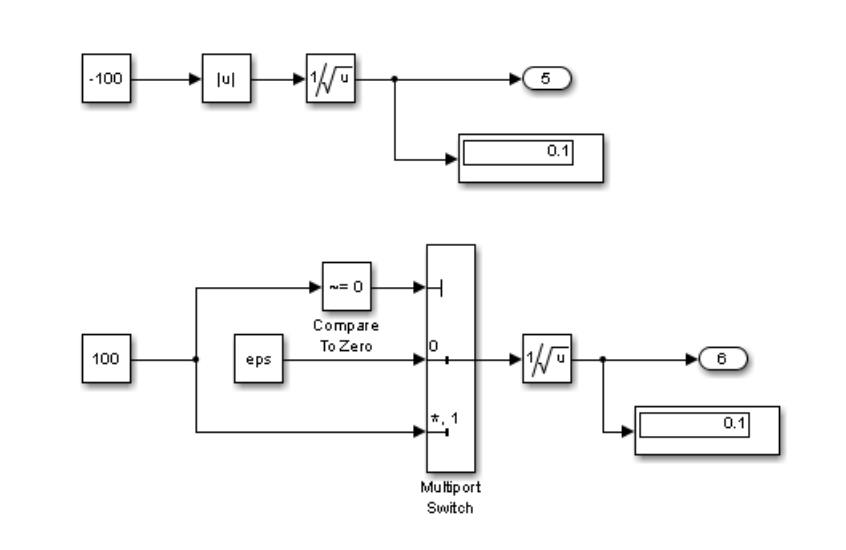
ID: Title	hisl_0002: Usage of remainder and reciprocal operations
Note	You can get a divide-by-zero operation, resulting in an infinite (`Inf`) output value for the `reciprocal` function, or a Not-a-Number (`NaN`) output value for the `rem` function. To avoid overflows or undefined values, protect the corresponding input from going to zero.
Rationale	Protect against overflows and undefined numerical results.
Model Advisor Checks	“Check usage of remainder and reciprocal operations” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Dir 4.1 • INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors
Last Changed	R2021b
Examples	In the following example, when the input signal oscillates around zero, the output exhibits a large change in value. You need further protection against the large change in value. 

hisl_0003: Usage of square root operations

ID: Title	hisl_0003: Usage of square root operations	
Description	To support robustness of generated code, when using the Square Root operations, do one of the following:	
	A	Account for complex numbers as the output.
	B	Protect the input from going negative.
Rationale	Avoid undesirable results in generated code.	
Model Advisor Checks	“Check usage of square root operations” (Simulink Check)	
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Dir 4.1	
Last Changed	R2021b	
Examples		

hisl_0028: Usage of Reciprocal Square Root blocks

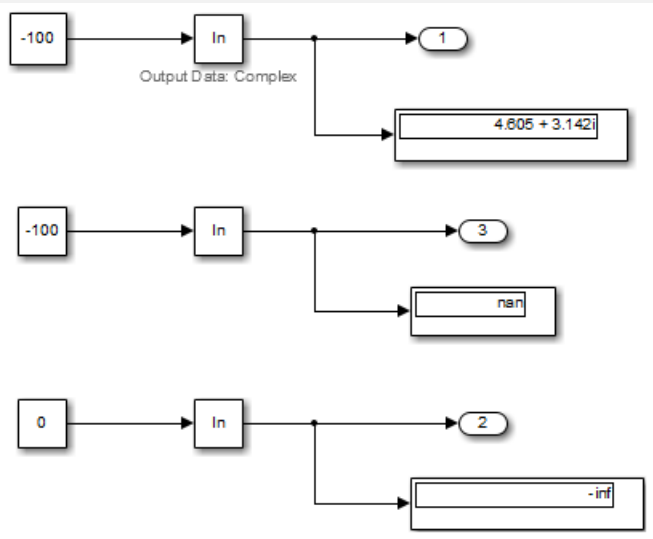
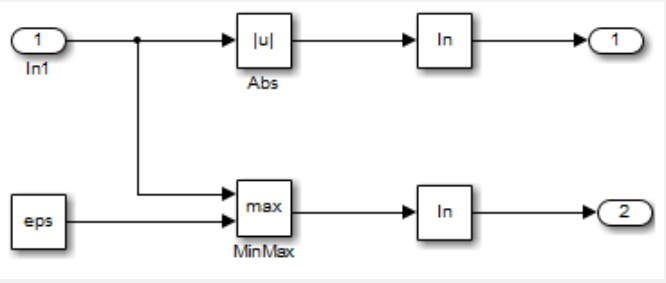
ID: Title	hisl_0028: Usage of Reciprocal Square Root blocks	
Description	To support robustness of generated code, when using the Reciprocal Square Root block, do one of the following:	
	A	Protect the input from going negative.
	B	Protect the input from going to zero.

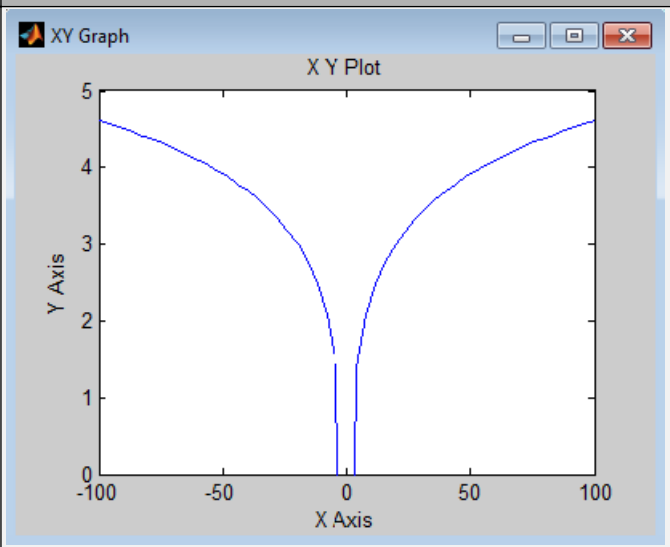
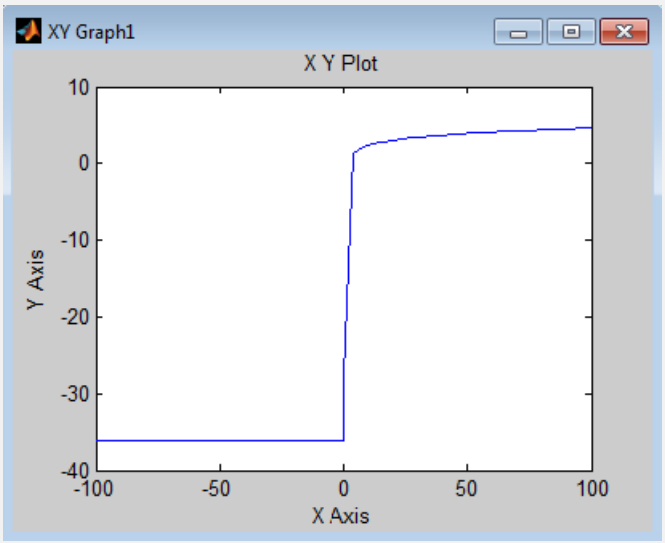
ID: Title	hisl_0028: Usage of Reciprocal Square Root blocks	
Note	You can get a divide-by-zero operation, resulting in an `(Inf)` output value for the reciprocal function. To avoid overflows or undefined values, protect the corresponding input from going to zero.	
Rationale	A, B	Avoid undesirable results in generated code.
Model Advisor Checks	“Check usage of Reciprocal Sqrt blocks” (Simulink Check)	
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Dir 4.1 • INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors	
Last Changed	R2021b	
Examples		

hisl_0004: Usage of natural logarithm and base 10 logarithm operations

ID: Title	hisl_0004: Usage of natural logarithm and base 10 logarithm operations	
Description	To support robustness of generated code, when using the math operations like natural logarithm (`log`) or base 10 logarithm (`log10`) :	
	A	Protect the input from going negative.

ID: Title	hisl_0004: Usage of natural logarithm and base 10 logarithm operations	
	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	“Check usage of log and log10 operations” (Simulink Check)	
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Dir 4.1 • INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors	
Last Changed	R2021b	

ID: Title	hisl_0004: Usage of natural logarithm and base 10 logarithm operations
Examples	 You can protect against: • Negative numbers using an Abs block. • Zero values using a combination of the MinMax block and a Constant block, with **Constant value** set to `eps` (epsilon). The following example displays the resulting output for input values ranging from `-100` to `100`. 

ID: Title	hisl_0004: Usage of natural logarithm and base 10 logarithm operations
	 

hisl_0005: Usage of Product blocks

ID: Title	hisl_0005: Usage of Product blocks
Description	When the Product block parameter **Multiplication** is set to `Matrix(*)`, protect divisor inputs from becoming singular input matrices.
Notes	When using Product blocks to compute the inverse of a matrix, or a matrix division, you might get a divide by a singular matrix. This division results in a `NaN` output. To avoid overflows, protect divisor inputs from becoming singular input matrices.
Rationale	Protect against overflows and support robustness of generated code.
Model Advisor Checks	Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

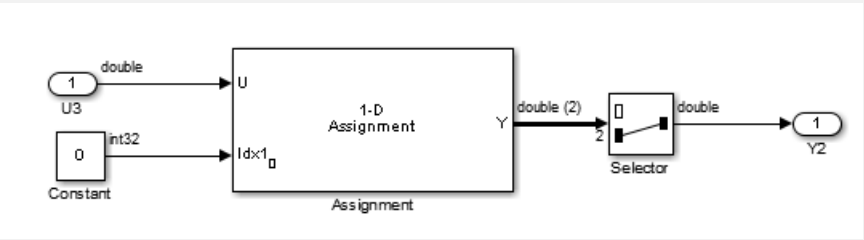
ID: Title	hisl_0005: Usage of Product blocks
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate • MISRA C:2012, Dir 4.1
Prerequisites	hisl_0314: Configuration Parameters > Diagnostics > Data Validity > Signals
Last Changed	R2021a

hisl_0029: Usage of Assignment blocks

ID: Title	hisl_0029: Usage of Assignment blocks
Description	To support robustness of generated code, when using the Assignment block, initialize array fields before their first use.
Notes	If the output vector of the Assignment block is not initialized with an input to the block, elements of the vector might not be initialized in the generated code. When the Assignment block is used iteratively and array fields 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. For a partial write operations, maintain a persistent output buffer (for example, see cgsl_0408, Partial data send for component deployment).
Rationale	Avoid undesirable results in generated code.
Model Advisor Checks	“Check usage of Assignment blocks” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ IEC 61508-3, Table A.3 (2) ‘Strongly typed programming language’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.2.g – 'Algorithms are accurate' • MISRA C:2012, Rule 9.1 • EXP33-C. Do not read uninitialized memory

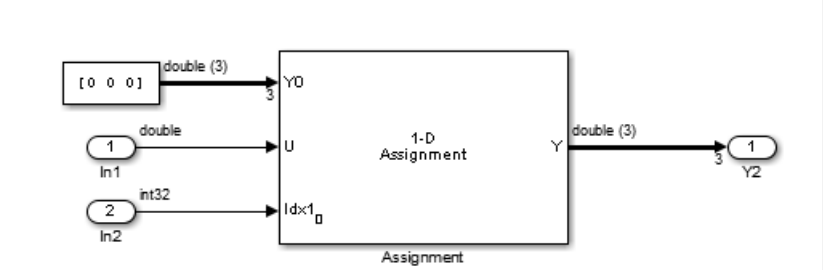
ID: Title	hisl_0029: Usage of Assignment blocks
Last Changed	R2023a

ID: Title	hisl_0029: Usage of Assignment blocks
Examples	




```
31   /* Model step function */
32   void Assignment1_step(void)
33   {
34     real_T rtb_Assignment[2];
35
36     /* Assignment: '<Root>/Assignment' incorporates:
37      *  Constant: '<Root>/Constant'
38      *  Inport: '<Root>/U3'
39      */
40     rtb_Assignment[0] = Assignment1_U.U3;
41
42     /* Outport: '<Root>/Y2' */
43     Assignment1_Y.Y2 = rtb_Assignment[1];
44   }
```

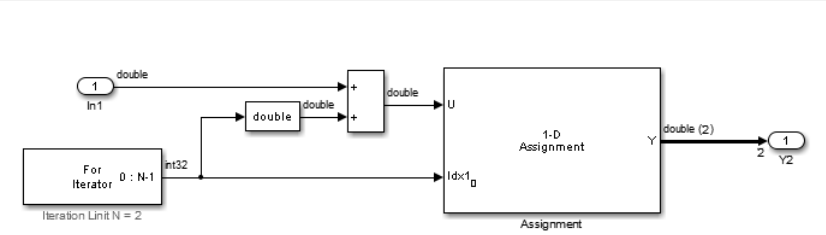
Not Recommended: No initialization input Y0 when block is not used iteratively




```
/* Model step function */
32   void Assignment2_step(void)
33   {
34     /* Assignment: '<Root>/Assignment' incorporates:
35      *  Constant: '<Root>/Constant'
36      *  Inport: '<Root>/In1'
37      *  Inport: '<Root>/In2'
38      */
39     Assignment2_Y.Y2[0] = 0.0;
40     Assignment2_Y.Y2[1] = 0.0;
41     Assignment2_Y.Y2[2] = 0.0;
42     Assignment2_Y.Y2[Assignment2_U.In2] = Assignment2_U.In1;
43   }
```

Recommended: Initialization input Y0 when block is not used iteratively

ID: Title	hisl_0029: Usage of Assignment blocks




```
/* Model step function */
  void Assignment3_step(void)
  {
    int32_T s1_iter;

    /* Outputs for Iterator SubSystem: '<Root>/For Iterator Subsystem' incorporates:
     *  ForIterator: '<S1>/For Iterator'
     */
    for (s1_iter = 0; s1_iter < 2; s1_iter++) {
      /* Assignment: '<S1>/Assignment' incorporates:
       *  DataTypeConversion: '<S1>/Data Type Conversion'
       *  Inport: '<Root>/In1'
       *  Sum: '<S1>/Add'
       */
      Assignment3_Y.Out1[s1_iter] = Assignment3_U.In1 + ((real_T)s1_iter);
    }

    /* End of Outputs for SubSystem: '<Root>/For Iterator Subsystem' */
  }
```

Recommended: Initialize array fields when block is used iteratively

hisl_0066: Usage of Gain blocks

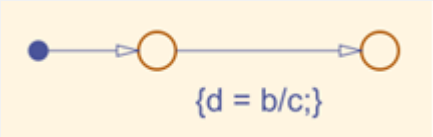
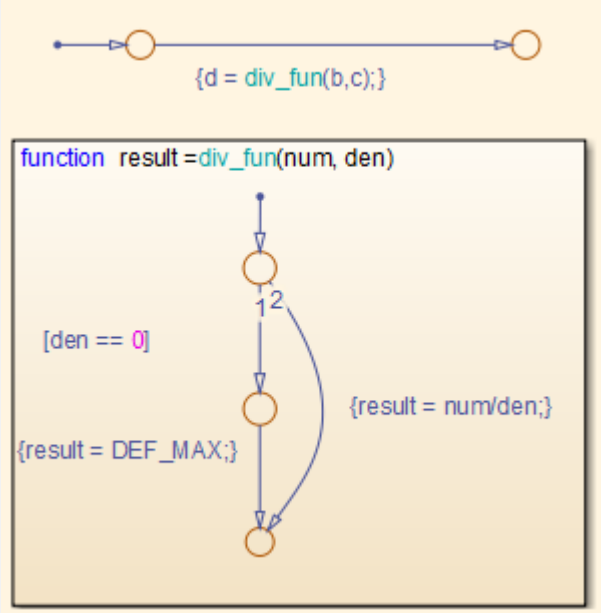
ID: Title	hisl_0066: Usage of Gain blocks
Description	To support traceability of generated code, the value of the Gain block must not resolve to 1.
Notes	The code generation process can remove Gain values equal to 1 during optimization, resulting in model elements with no traceable code. An exception to this rule is setting the Gain value to a named parameter data object with a non-auto storage class.
Rationale	Support the generation of traceable code.
Model Advisor Checks	“Check usage of Gain blocks” (Simulink Check)

ID: Title	hisl_0066: Usage of Gain blocks
References	• DO-331, Section MB 6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' IEC 61508-3, Table B.8 (3) 'Control Flow Analysis' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 7 (1f) 'Control flow analysis' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.19 (3) 'Control Flow Analysis'
Last Changed	R2018a

hisl_0067: Protect against divide-by-zero calculations

ID: Title	hisl_0067: Protect against divide-by-zero calculations
Description	To support robustness of generated code, when performing divide operations, protect the divisor from going to zero.
Note	To prove that division-by-zero is not possible, perform a static analysis of the model. If division-by-zero is possible, implement one of the following. Using more than one option can result in redundant protection operations: • Execute the divide-by-zero Model Advisor check • Modify the code generation process to use Code Replacement Libraries (CRLs) • For integer-based operations, clear configuration parameter **Remove code that protects against division arithmetic exceptions** Using CRLs or clearing configuration parameter **Remove code that protects against division arithmetic exceptions** protects division operations against divide-by-zero operations. However, this action does introduce additional computational and memory overhead, as well as the potential to introduce unreachable code.
Rationale	Improve code compliance of generated code
Model Advisor Checks	“Check for divide-by-zero calculations” (Simulink Check)

ID: Title	hisl_0067: Protect against divide-by-zero calculations
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Dir 4.1
See Also	• “What Is Code Replacement?” (Simulink Coder) • “Code Replacement Libraries” (Simulink Coder) • “hisl_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions” on page 5-24
Last Changed	R2021a

ID: Title	hisl_0067: Protect against divide-by-zero calculations
Example	**Incorrect** Division operation can result in a divide-by-zero scenario.  **Correct** Graphical function to model divide-by-zero check. 

Ports & Subsystems

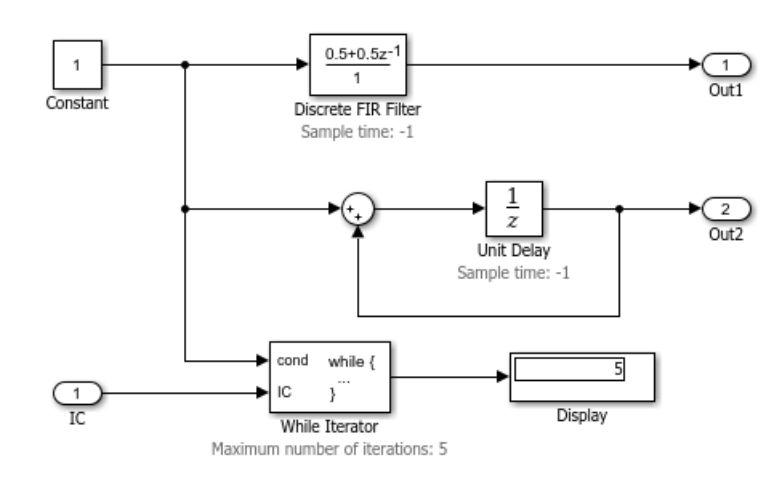
In this section...

hisl_0006: Usage of While Iterator blocks

ID: Title	hisl_0006: Usage of While Iterator blocks
Description	To support bounded iterative behavior in the generated code when using the While Iterator block, set block parameter **Maximum number of iterations** to a positive integer value.
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	Support bounded iterative in the generated code.
Model Advisor Checks	“Check usage of While Iterator blocks” (Simulink Check)
References	• DO-331, Section MB.6.3.2.g – 'Algorithms are accurate' • IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1 • INT32-C. Ensure that operations on signed integers do not result in overflow

ID: Title	hisl_0006: Usage of While Iterator blocks
Last Changed	R2021b

hisl_0007: Usage of For Iterator or While Iterator subsystems

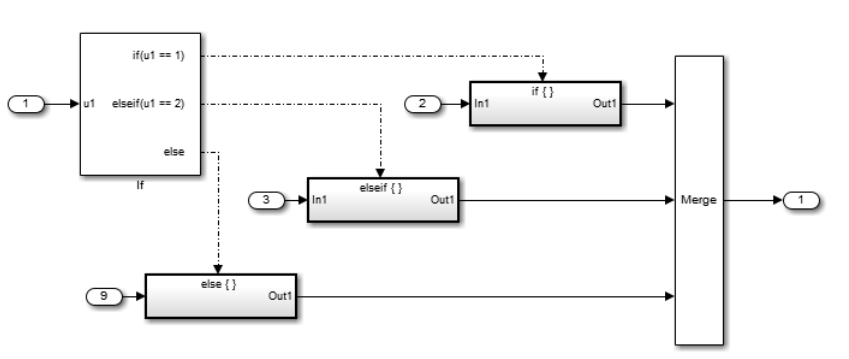
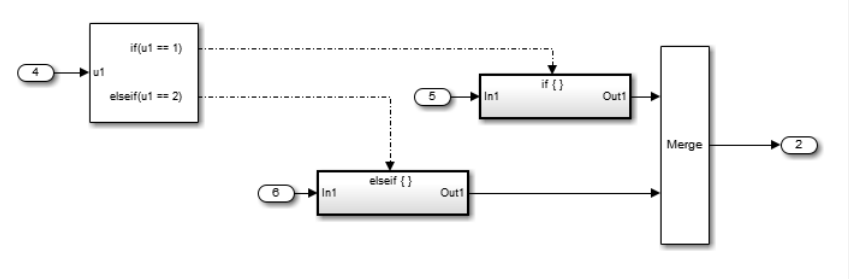
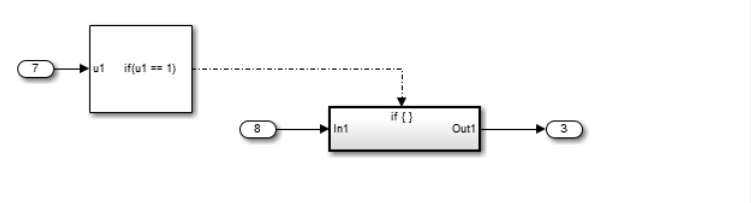
ID: Title	hisl_0007: Usage of For Iterator or While Iterator subsystems
Description	To support unambiguous behavior, when using For Iterator Subsystem or While Iterator Subsystem, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions within the subsystems.
Rationale	Avoid ambiguous behavior from the subsystem.
Model Advisor Checks	“Check usage of For and While Iterator subsystems” (Simulink Check)
References	• DO-331, Section MB.6.3.2.g 'Algorithms are accurate • IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1
Last Changed	R2018b
Examples	The following example causes a warning: the Discrete FIR Filter block is time-dependent and is in a For or While Iterator subsystem. 

hisl_0008: Usage of For Iterator Blocks

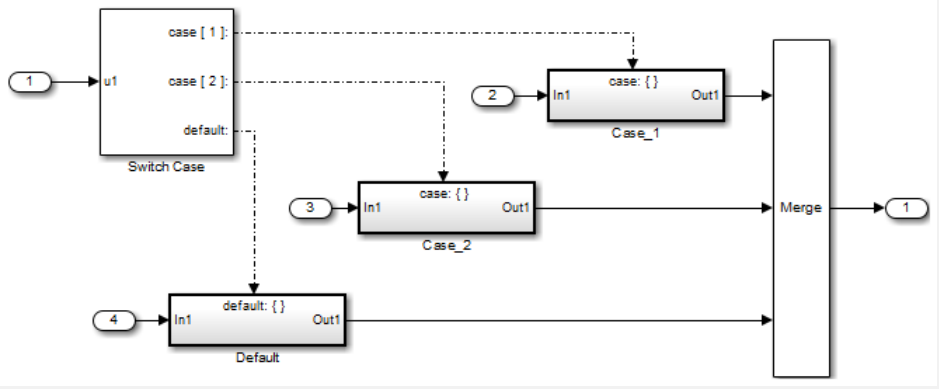
ID: Title	hisl_0008: Usage of For Iterator blocks	
Description	To support bounded iterative behavior in the generated code when using the For Iterator block, do one of the following:	
	A	Set block parameter **Iteration limit source** to `internal`.
	B	When **Iteration limit source** must be `external`, use a block that has a constant value. Options include Width, Probe, or Constant.
	C	Clear block parameters **Set next i (iteration variable) externally**.
	D	To observe the iteration value during simulation, select block parameter **Show iteration variable**.
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	“Check usage of For Iterator blocks” (Simulink Check)	
References	• DO-331, Section MB.6.3.2.g – 'Algorithms are accurate' • IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1	
Last Changed	R2016a	

hisl_0010: Usage of If blocks and If Action Subsystem blocks

ID: Title	hisl_0010: Usage of If blocks and If Action Subsystem blocks	
Description	To support verifiable generated code, when using the If block with nonempty `Elseif` expressions,	
	A	Select block parameter **Show else condition**.
	B	Connect the outports of the If block to If Action Subsystem blocks.
Prerequisites	“hisl_0016: Usage of blocks that compute relational operators” on page 2-35	
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	“Check usage of If blocks and If Action Subsystem blocks” (Simulink Check)	

ID: Title	hisl_0010: Usage of If blocks and If Action Subsystem blocks
References	• DO-331, Section MB.6.3.2.d – 'Low-level requirements are verifiable' DO-331 Section MB.6.3.2.b – Low-level requirements are accurate and consistent • IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1
Last Changed	R2016b
Examples	 **Recommended: Elseif with Else**
	 **Not Recommended: No Else Path**
	 **Recommended: Only an If, no Else required**

hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks

ID: Title	hisl_0011: Usage of Switch Case blocks and Action Subsystem blocks	
Description	To support verifiable generated code, when using the Switch Case block:	
	A	Select block parameter **Show default case**.
	B	Connect the outports of the Switch Case block to a Switch Case Action Subsystem block.
	C	Use an integer data type or an enumeration value for the inputs to Switch Case blocks.
Prerequisites	“hisl_0016: Usage of blocks that compute relational operators” on page 2-35	
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	“Check usage of Switch Case blocks and Switch Case Action Subsystem blocks” (Simulink Check)	
References	• DO-331, Section MB.6.3.2.d – ‘Low-level requirements are verifiable DO-331 Section MB.6.3.2.b – Low-level requirements are accurate and consistent • IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1	
Last Changed	R2016b	
Examples	The following graphic displays an example of providing a default path of execution using a “Default” block. 	

hisl_0012: Usage of conditionally executed subsystems

ID: Title	hisl_0012: Usage of conditionally executed subsystems	
Description	To support unambiguous behavior, when using conditionally executed subsystems:	
	A	Specify inherited (`-1`) sample times for all blocks in the subsystem, except Constant. Constant blocks can use infinite (`inf`) sample time.
	B	If the subsystem is called asynchronously, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystem.
Rationale	A, B	Support unambiguous behavior.
Model Advisor Checks	“Check usage of conditionally executed subsystems” (Simulink Check)	
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262–6, Table 1(b) 'Use of language subsets' ISO 26262–6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Sections MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB 6.3.2.b 'Low-level requirements are accurate and consistent'	
Last Changed	R2018b	
Examples	When using discrete blocks, the behavior depends on the operation across multiple contiguous time steps. When the blocks are called intermittently, the results may not conform to your expectations.	

hisl_0024: Inport interface definition

ID: Title	hisl_0024: Inport interface definition
Description	To support strong data typing and unambiguous behavior of the model and the generated code, set parameters **Data type**, **Port dimensions**, and **Sample time** for each: • Model root-level inport • Signal object that explicitly resolves to a connected signal line • Architecture model root-level inport port For export-function models, you can set **Sample time** to `-1`.
Note	Using root-level Inport blocks without fully defined dimensions, sample times, or data type can lead to ambiguous simulation results.
Rationale	• Avoid ambiguous behavior. • Support full specification of the software interface.

ID: Title	hisl_0024: Inport interface definition
Model Advisor Checks	“Check for root Inports with missing properties” (Simulink Check)
References	• DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table B.9 (6) ‘Fully defined interface’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1a) – Enforcement of low complexity ISO 26262-6, Table 1 (1c) – Enforcement of strong typing ISO 26262-6, Table 1 (1f) – Use of unambiguous graphical representation ISO 26262-6, Table 3 (1c) – Restricted size of interfaces ISO 26262-6, Table 7 (1k) – Interface test • EN 50128, Table A.3 (19) ‘Fully Defined Interface‘
Last Changed	R2023a

hisl_0025: Design min/max specification of input interfaces

ID: Title	hisl_0025: Design min/max specification of input interfaces
Description	Provide design minimum and maximum interface ranges for each • root-level Inport block in Simulink model • root-level Input port of Architecture model

Notes	• Specifying the range of root level Input ports enables additional capabilities.[a]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 are used by Embedded Coder to optimize the generated code. To use these design ranges for optimization, select configuration parameter **Optimize using the specified minimum and maximum values**. This configuration parameter is applicable only when the **System target file** is an ERT-based target.
	• 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	“Check for root Inports with missing range definitions” (Simulink Check)

ID: Title	hisl_0025: Design min/max specification of input interfaces
References	• DO-331, Section MB.6.3.2.d – ‘Low-level requirements are verifiable’ DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table B.9 (6) ‘Fully defined interface’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1c) – Enforcement of strong typing ISO 26262-6, Table 7 (1e) – Formal verification ISO 26262-6, Table 7 (1k) – Interface test ISO 26262-6, Table 8 (1c) – Analysis of boundary values ISO 26262-6, Table 3 (1c) – Restricted size of interfaces • EN 50128, Table A.1(11) – Software Interface Specifications EN 50128 Table A.3 (19) ‘Fully Defined Interface‘
Last Changed	R2022b

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

hisl_0026: Design min/max specification of output interfaces

ID: Title	hisl_0026: Design min/max specification of output interfaces
Description	Provide minimum and maximum interface ranges for each • root-level Outport block in Simulink model • root-level Outport ports of Architecture model

Notes	• Specifying the range of root level Output ports enables additional capabilities.[a]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 are used by Embedded Coder to optimize the generated code. To set these design ranges, select configuration parameter **Optimize using the specified minimum and maximum values**. This configuration parameters is applicable only when the **System target file** is an ERT-based target.
	• 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	“Check for root Outports with missing range definitions” (Simulink Check)

ID: Title	hisl_0026: Design min/max specification of output interfaces
References	• DO-331, Section MB.6.3.2.d – ‘Low-level requirements are verifiable’ DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table B.9 (6) ‘Fully defined interface’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1c) – Enforcement of strong typing ISO 26262-6, Table 7 (1e) – Formal verification ISO 26262-6, Table 7 (1k) – Interface test ISO 26262-6, Table 8 (1c) – Analysis of boundary values ISO 26262-6, Table 3 (1c) – Restricted size of interfaces • EN 50128, Table A.1(11) – Software Interface Specifications EN 50128 Table A.3 (19) ‘Fully Defined Interface‘
Last Changed	R2022b

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

hisl_0077: Outport interface definition

ID: Title	hisl_0077: Outport interface definition
Description	To support strong data typing and unambiguous behavior of the model and the generated code, set the parameters **Data type**, **Port dimensions**, and **Sample time** for each: • Model root-level outport • Signal object that explicitly resolves to a connected signal line • Architecture model root-level outport port For export-function models, you can set **Sample time** to `-1`.
Note	Using root-level Outport 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 ambiguous behavior. • Support full specification of software interface.
Model Advisor Checks	“Check for root Outports with missing properties” (Simulink Check)
References	• DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table B.9 (6) ‘Fully defined interface’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1a) – Enforcement of low complexity ISO 26262-6, Table 1 (1c) – Enforcement of strong typing ISO 26262-6, Table 1 (1f) – Use of unambiguous graphical representation ISO 26262-6, Table 3 (1c) – Restricted size of interfaces ISO 26262-6, Table 7 (1k) – Interface test • EN 50128, Table A.3 (19) ‘Fully Defined Interface‘

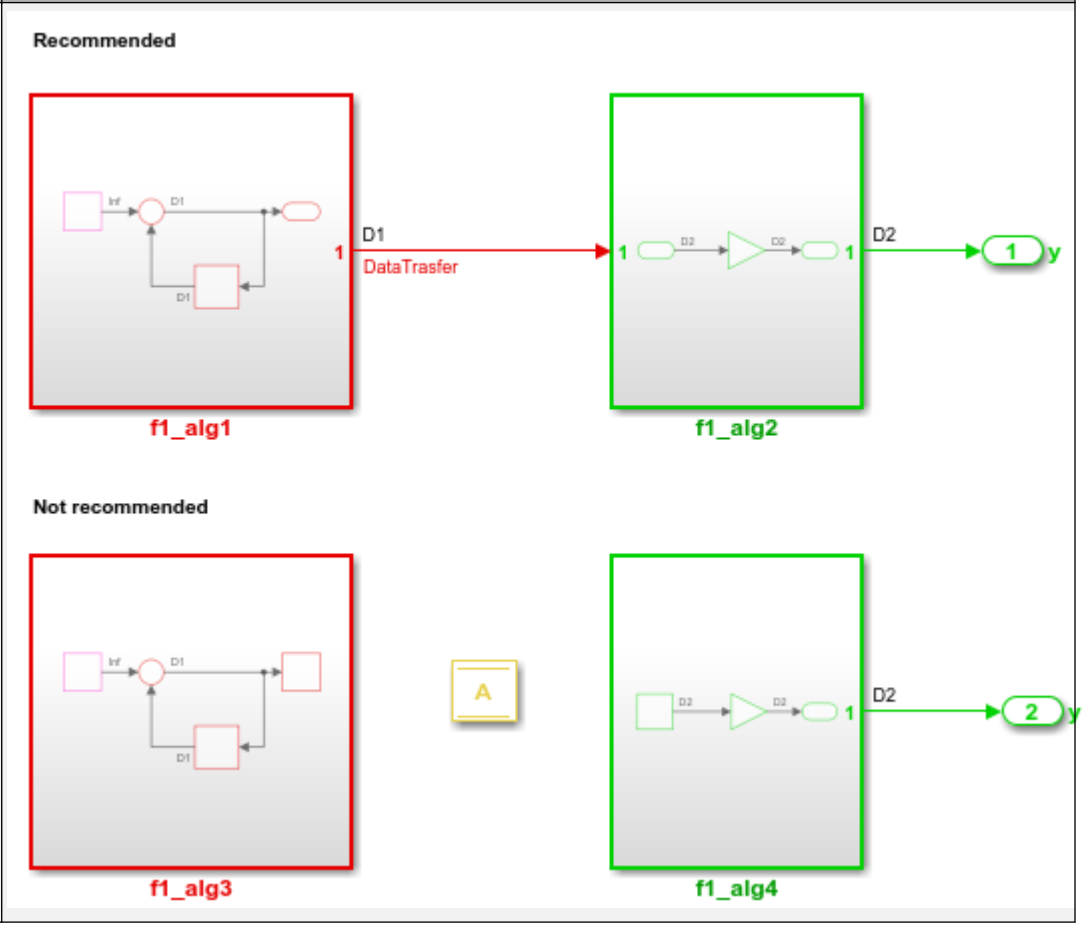
ID: Title	hisl_0077: Outport interface definition
Last Changed	R2023a

Signal Routing

In this section...

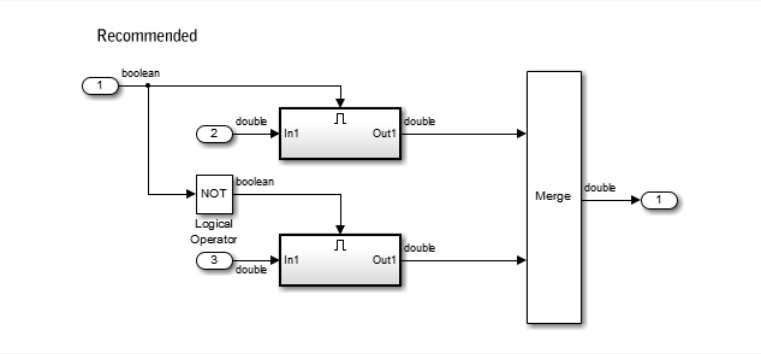
hisl_0013: Usage of data store memory

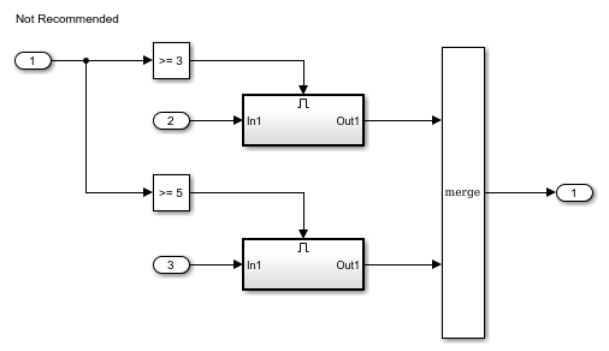
ID: Title	hisl_0013: Usage of data store memory
Description	When using data store memory, set these data store memory model configuration parameters to `error`: • **Detect read before write** • **Detect write after read** • **Detect write after write** • **Multitask data store** • **Duplicate data store names**
Notes	Use input and output signals instead of data store memory to communicate data whenever possible. Using data store memory blocks to communicate data across system boundaries can lead to unexpected results. If you have a Simulink Design Verifier license, you can detect data store memory access violations in a model statically. Select the Design Verifier model configuration parameter **Data store access violations**. For more information, see “Detect Data Store Access Violations in a Model” (Simulink Design Verifier).
Rationale	Simulation diagnostics help identify places where data store memory communication invalidates the Simulink signal semantic.
Model Advisor Checks	“Check safety-related diagnostic settings for data store memory” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.3.b 'Software architecture is consistent’
Last Changed	R2023a

ID: Title	hisl_0013: Usage of data store memory
Examples	

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 block parameter **Allow unequal port widths**.
	D	Set the Outport block parameter **Output when disabled** to `held` for each conditionally executed subsystem being merged.

ID: Title	hisl_0015: Usage of Merge blocks	
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. 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 hisl_0304: Configuration Parameters > Diagnostics > Model initialization	
Rationale	A, B, C, D	Avoid ambiguous behavior.
Model Advisor Checks	“Check usage of Merge blocks” (Simulink Check)	
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.3.b 'Software architecture is consistent’	
See Also	Merge block in the Simulink documentation	
Last Changed	R2018b	
Examples	 **Recommended**	

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

hisl_0021: Consistent vector indexing method

ID: Title	hisl_0021: Consistent vector indexing method	
Description	Within a model, use:	
	A	Consistent vector indexing method. Supports configurable indexing: • Assignment • For Iterator • Index Vector • Multiport Switch • Selector Support only one-based indexing: • Fcn (deprecated) • MATLAB Function • MATLAB System • State Transition Table with MATLAB action language • Test Sequence • Stateflow chart with MATLAB action language • Truth Table function with MATLAB action language Supports only zero-based indexing: • Stateflow chart with C action language • Truth Table function with C action language • State Transition Table with C action language
Rationale	A	Reduce the risk of introducing errors due to inconsistent indexing.
Model Advisor Checks	“Check for inconsistent vector indexing methods” (Simulink Check)	

ID: Title	hisl_0021: Consistent vector indexing method
References	• IEC 61508–3, Table A.3 (3) 'Language subset' IEC 61508–3, Table A.4 (5) 'Design and coding standards' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guide' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.12 (1) 'Coding Standard' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
See Also	“cgsl_0101: Zero-based indexing”
Last Changed	R2019a

hisl_0022: Data type selection for index signals

ID: Title	hisl_0022: Data type selection for index signals	
Description	For index signals, use:	
	A	An integer or enumerated data type
	B	A data type that covers the range of indexed values.
	Blocks that use a signal index include: • Assignment • Direct Lookup Table (n-D) • Index Vector • Interpolation Using Prelookup • MATLAB® Function • Multiport Switch • Selector • Stateflow® Chart	
Rationale	A	Prevent unexpected results that can occur with rounding operations for floating-point data types.
	B	Enable access to data in a vector.
Model Advisor Checks	“Check data types for blocks with index signals” (Simulink Check)	

ID: Title	hisl_0022: Data type selection for index signals
References	• IEC 61508–3, Table A.3 (2) 'Strongly typed programming language' IEC 61508–3, Table A.4 (3) 'Defensive programming' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.2.g – 'Algorithms are accurate' • FLP30-C. Do not use floating-point variables as loop counters
Last Changed	R2021b

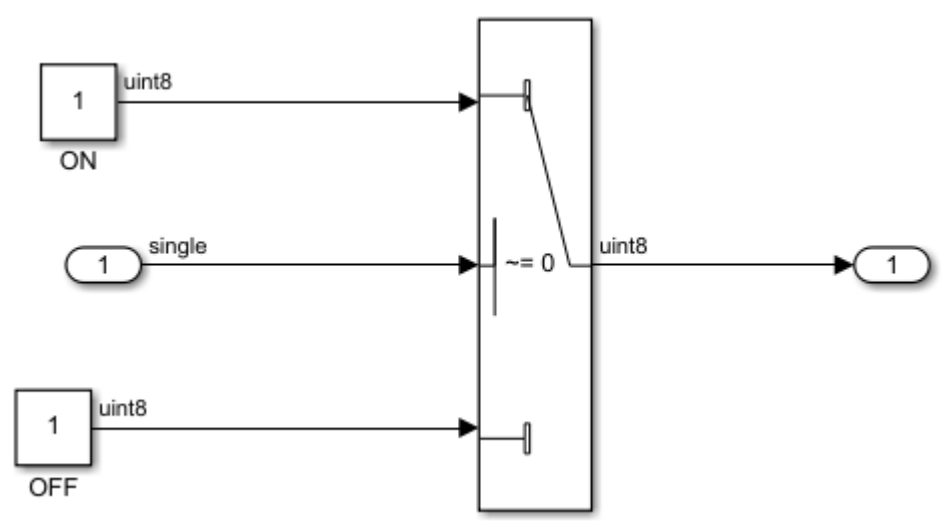
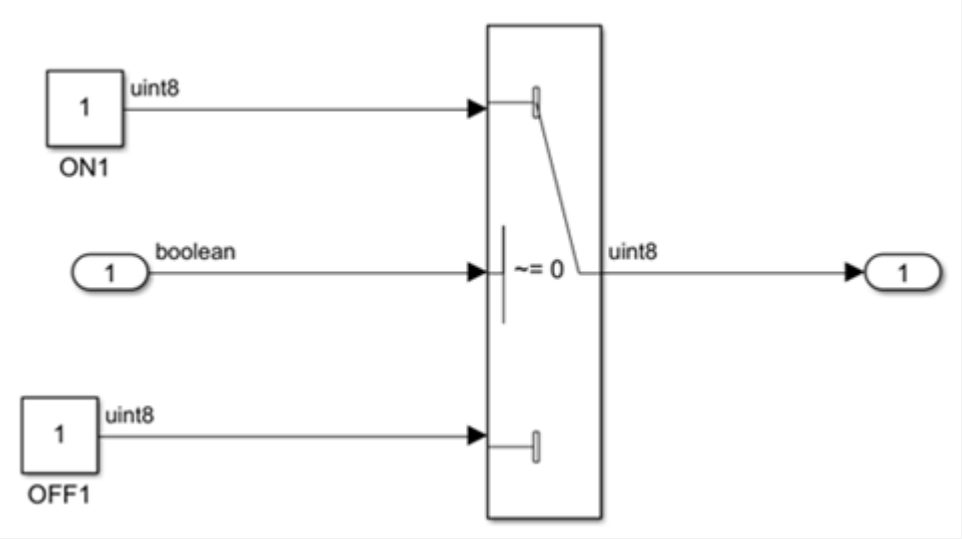
hisl_0023: Verification of variant blocks

ID: Title	hisl_0023: Verification of variant blocks	
Description	When verifying that a model is consistent with generated code, do the following:	
	A	For each Variant block, set the **Variant activation time** to `update diagram` or `update diagram analyze all choices`.
Rationale	A	Simplify consistency testing between the model and generated code by restricting the code base to a single variant.
Model Advisor Checks	“Check usage of variant blocks” (Simulink Check)	
References	• DO-331, Section MB.6.3.3.b – Software architecture is consistent • IEC 61508–3, Table A.4 (7) 'Use of trusted / verified software modules and components'	
Last Changed	R2021b	
See Also	Variant Subsystem, Variant Model, Variant Assembly Subsystem	

hisl_0034: Usage of Signal Routing blocks

ID: Title	hisl_0034: Usage of Signal Routing blocks
Description	When using Switch blocks, avoid comparisons using the ~= operator on floating-point data types.
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	Improve model robustness.
Model Advisor Checks	“Check usage of Signal Routing blocks” (Simulink Check)

ID: Title	hisl_0034: Usage of Signal Routing blocks
References	• DO-331, Sections MB.6.3.2.g 'Algorithms are accurate' • IEC 61508-3, Table A.3 (3) – 'Language subset' Table A.4 (3) – 'Defensive programming' • IEC 62304, 5.5.3 - 'Software Unit acceptance criteria' • ISO 26262-6, Table 1 (1b) - 'Use of language subsets' Table 1 (1d) - 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) - 'Language Subset' Table A.3 (1) - 'Defensive Programming' • MISRA C:2012, Dir 1.1
Last Changed	R2021a

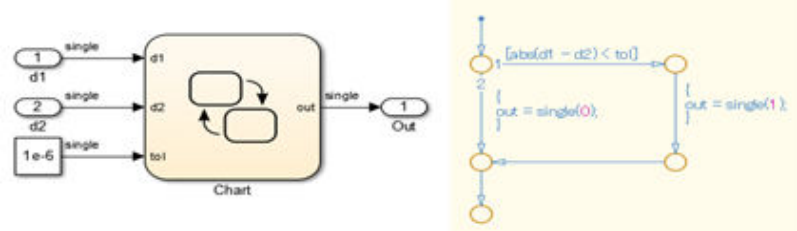
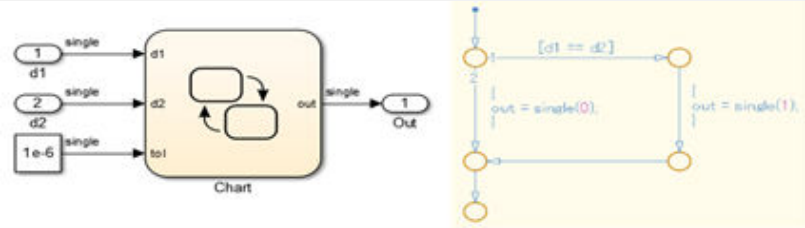
ID: Title	hisl_0034: Usage of Signal Routing blocks
Examples	**Not Recommended**  **Recommended** 

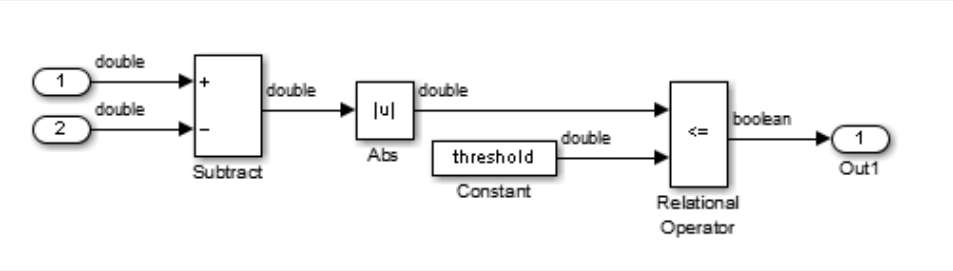
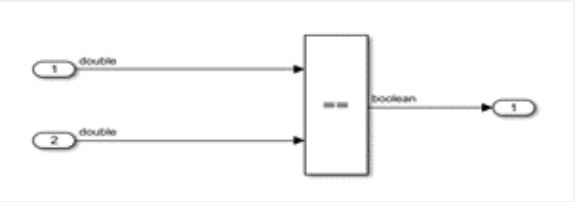
Logic and Bit Operations

In this section...

hisl_0016: Usage of blocks that compute relational operators

ID: Title	hisl_0016: Usage of blocks that compute relational operators
Description	To support the robustness of the operations, avoid using the equality and inequality operators on floating-point data types.
Notes	Due to floating-point precision issues, do not test floating-point expressions for equality (==) or inequality (~=, !=).
Rationale	Improve model robustness and prevent unexpected results.
Model Advisor Checks	“Check relational comparisons on floating-point signals” (Simulink Check)
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Dir 1.1
See Also	“Relational Operations”
Last Changed	R2021b

ID: Title	hisl_0016: Usage of blocks that compute relational operators
Examples	**Ex: 1** **Example — Correct** • myDouble > 0.99 && myDouble < 1.01; % test range **Example — Incorrect** • myDouble == 1.0 • mySingle ~= 15.0 **Ex: 2** **Example — Correct** Equality comparison operators are not used in floating-point operands.  **Example — Incorrect** Equality comparison operator == is used in floating-point operands.  **Example — Correct** To test whether two floating-point variables or expressions are equal, compare the difference of the two variables against a threshold that takes into account the floating-point relative accuracy (eps) and the magnitude of the numbers. The following pattern shows how to test two double-precision input signals, In1 and In2, for equality.

ID: Title	hisl_0016: Usage of blocks that compute relational operators
	 **Example — InCorrect** Equality comparison operator == is used in floating-point operands 

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

ID: Title	hisl_0017: Usage of blocks that compute relational operators (2)	
Description	To support unambiguous behavior in the generated code, when using blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare to Zero, and Detect Change	
	A	Set block parameter **Output data type** to `Boolean`.
	B	For Relational Operator blocks, verify that input signals are of the same data type.
Rationale	A, B	Support generation of code that produces unambiguous behavior.
Model Advisor Checks	“Check usage of Relational Operator blocks” (Simulink Check)	
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.3 (1) 'Defensive Programming' • 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” on page 2-35	

ID: Title	hisl_0017: Usage of blocks that compute relational operators (2)
Last Changed	R2018a

hisl_0018: Usage of Logical Operator block

ID: Title	hisl_0018: Usage of Logical Operator block	
Description	To support unambiguous behavior of generated code, when using the Logical Operator block,	
	A	Set block parameter **Output data type** to `Boolean`.
	B	Ensure input signals are of type `Boolean`.
Prerequisites	“hisl_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)” on page 5-5	
Rationale	A, B	Avoid ambiguous behavior of generated code.
Model Advisor Checks	“Check usage of Logical Operator blocks” (Simulink Check)	
References	• DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Directive 10.1	
Last Changed	R2017b	

hisl_0019: Usage of bitwise operations

ID: Title	hisl_0019: Usage of bitwise operations	
Description	To support unambiguous behaviour, when using bitwise operations,	
	A	Avoid bitwise operations on signed integer data types.
Notes	Bitwise operations are not meaningful on signed integers due to unpredictable behaviour. For example, a shift operation might move the sign bit into the number, or a numeric bit into the sign bit.	
Rationale	A	Support unambiguous behavior of generated code.
Model Advisor Checks	“Check usage of bit operation blocks” (Simulink Check)	

ID: Title	hisl_0019: Usage of bitwise operations
References	• DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.3 (2) 'Strongly typed programming language' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • EN 50128, Table A.3 (1) 'Defensive Programming' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • MISRA C:2012, Rule 10.1
See Also	“hisl_0073: Usage of bit-shift operations” on page 2-44
Last Changed	R2021b

Lookup Table Blocks

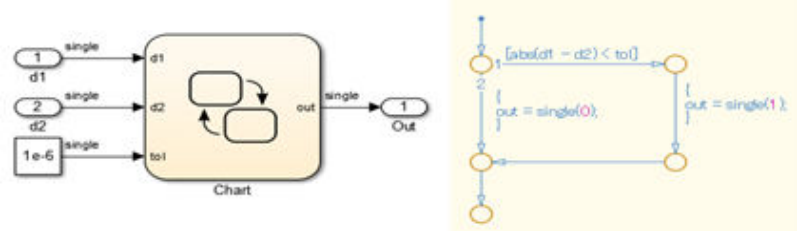
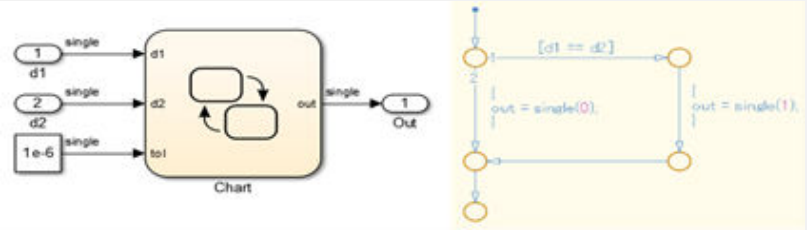
In this section...

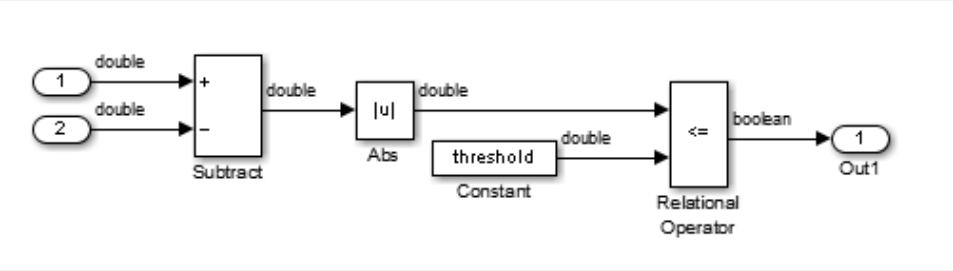
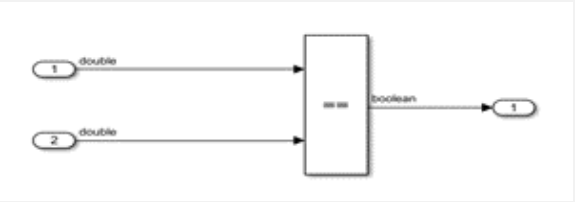
hisl_0033: Usage of Lookup Table blocks

ID: Title	hisl_0033: Usage of Lookup Table blocks	
Description	To support robustness of generated code, when using the 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, Prelookup, and Interpolation Using Prelookup blocks:	
	A	Clear block parameter **Remove protection against out-of-range input in generated code** in each 1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table, or Prelookup block.
	B	Clear block parameter **Remove protection against out-of-range index in generated code** in each Interpolation Using Prelookup block.
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	“Check usage of lookup table blocks” (Simulink Check)	
References	• DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming'	
See Also	1-D Lookup Table 2-D Lookup Table n-D Lookup Table Prelookup	
Last Changed	R2021a	

hisl_0016: Usage of blocks that compute relational operators

ID: Title	hisl_0016: Usage of blocks that compute relational operators
Description	To support the robustness of the operations, avoid using the equality and inequality operators on floating-point data types.
Notes	Due to floating-point precision issues, do not test floating-point expressions for equality (==) or inequality (~=, !=).
Rationale	Improve model robustness and prevent unexpected results.
Model Advisor Checks	“Check relational comparisons on floating-point signals” (Simulink Check)
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Dir 1.1
See Also	“Relational Operations”
Last Changed	R2021b

ID: Title	hisl_0016: Usage of blocks that compute relational operators
Examples	**Ex: 1** **Example — Correct** • myDouble > 0.99 && myDouble < 1.01; % test range **Example — Incorrect** • myDouble == 1.0 • mySingle ~= 15.0 **Ex: 2** **Example — Correct** Equality comparison operators are not used in floating-point operands.  **Example — Incorrect** Equality comparison operator == is used in floating-point operands.  **Example — Correct** To test whether two floating-point variables or expressions are equal, compare the difference of the two variables against a threshold that takes into account the floating-point relative accuracy (eps) and the magnitude of the numbers. The following pattern shows how to test two double-precision input signals, In1 and In2, for equality.

ID: Title	hisl_0016: Usage of blocks that compute relational operators
	 **Example — InCorrect** Equality comparison operator == is used in floating-point operands 

hisl_0072: Usage of tunable parameters for referenced models

ID: Title	hisl_0072: Usage of tunable parameters for referenced models
Description	Use the `Simulink.Parameter` object to define tunable parameters. This applies to all tunable parameters that are meant to be shared via either the base workspace or Simulink data dictionaries. It does not apply to model arguments.
Notes	Simulink ignores the storage class settings of parameters that are configured by using the **Model Parameter Configuration** dialog box for referenced models. This guideline is applicable only when configuration parameter **Default parameter behavior** is set to `Inlined`.
Rationale	Prevent unintended loss of parameter tunability.
Model Advisor Checks	“Check for parameter tunability information ignored for referenced models”
References	• DO-331, Section MB.6.3.1.g – Algorithms are accurate DO-331, Section MB.6.3.2.g – Algorithms are accurate • IEC 61508-3, Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming'
See Also	“Create Tunable Calibration Parameter in the Generated Code” (Simulink Coder)
Last Changed	R2021b

hisl_0073: Usage of bit-shift operations

ID: Title	hisl_0073: Usage of bit-shift operations
Description	For bit-shifting operations (e.g. a >> b or a << b), do not perform: Shift operations that are greater than or equal to the bit-width (b must not be equal or greater than the bit width of a).
Rationale	Generation of code with shift operations can result in violation of coding standards
Model Advisor Checks	“Check usage of bit-shift operations” (Simulink Check)
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' • IEC 61508–3, Table A.3 (2) Strongly typed programming language IEC 61508–3, Table A.4 (3) Defensive programming • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) Use of language subsets ISO 26262-6, Table 1 (1c) Enforcement of strong typing ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques • EN 50128, Table A.3 (1) Defensive Programming EN 50128, Table A.4 (8) Strongly Typed Programming Language • MISRA C:2012, Rule 12.2 • INT34-C. Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand
See Also	“Create Tunable Calibration Parameter in the Generated Code” (Simulink Coder)
Last Changed	R2021b

Stateflow Chart Considerations

Chart Properties

In this section...

hisf_0001: State Machine Type

ID: Title	hisf_0001: State Machine Type
Description	To create Stateflow charts that implement consistent Stateflow semantics, use the same **State Machine Type** (`Classic`, `Mealy`, or `Moore`) for all charts in the model.
Note	In Mealy charts, actions are associated with transitions. In the Moore charts, actions are associated with states. In Classic charts, actions can be associated with both transition and states. At compile time, Stateflow verifies that the chart semantics comply with the formal definitions and rules of the selected type of state machine. If the chart semantics are not in compliance, the software provides a diagnostic message.
Rationale	Promote a clear modeling style.
Model Advisor Checks	“Check state machine type of Stateflow charts” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
See Also	• “Specify Properties for Stateflow Charts” (Stateflow) • “Create Mealy and Moore Charts” (Stateflow)
Last Changed	R2018b

hisf_0002: User-specified state/transition execution order

ID: Title	hisf_0002: User-specified state/transition execution order	
Description	Do the following to explicitly set the execution order for active states and valid transitions in Stateflow charts:	
	A	In the Chart Properties dialog box, select **User specified state/transition execution order**.
Prerequisites	hisl_0311: Configuration Parameters > Diagnostics > Stateflow	

ID: Title	hisf_0002: User-specified state/transition execution order	
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.	
Rationale	A	Promote an unambiguous modeling style.
Model Advisor Checks	“Check Stateflow charts for ordering of states and transitions” (Simulink Check)	
References	This guideline supports adhering to: • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508–3, Table A.3 (3) 'Language subset' IEC 61508–3, Table A.4 (5) 'Design and coding standards' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.12 (1) 'Coding Standard' EN 50128, Table A.12 (2) 'Coding Style Guide'	
See Also	• “Specify Properties for Stateflow Charts” (Stateflow) • “Evaluate Transitions” (Stateflow) • “Execution Order for Parallel States” (Stateflow)	
Last Changed	R2018b	

hisf_0011: Stateflow debugging settings

ID: Title	hisf_0011: Stateflow debugging settings
Description	To protect against unreachable code and indeterminate execution time, • Set configuration parameters **Wrap on overflow** and **Simulation range checking** to `error`. In the model, open the **Debug** tab and select **Diagnostics > Detect Cyclical Behavior** • Right-click on each truth table in the model and select **Properties**. Set these parameters to `Error`: • **Underspecification** • **Overspecification**

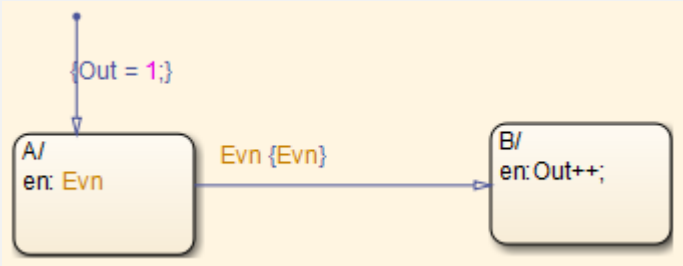
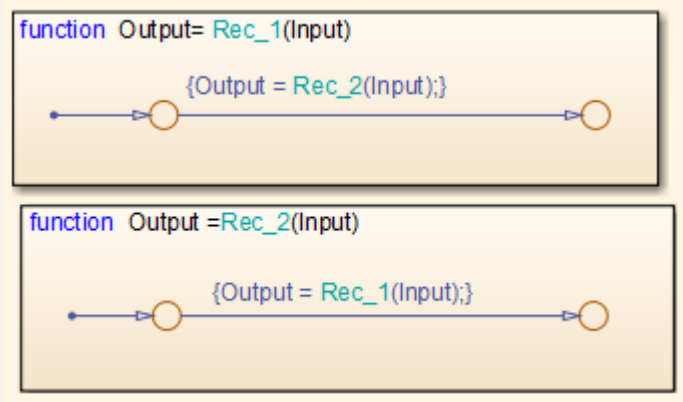
ID: Title	hisf_0011: Stateflow debugging settings
Notes	Run-time diagnostics are only triggered during simulation. If the error condition is not reached during simulation, the error message is not triggered for code generation.
Rationale	Protect against unreachable code and unpredictable execution time.
Model Advisor Checks	“Check Stateflow debugging options” (Simulink Check)
References	• DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.3.d 'Software architecture is verifiable' • IEC 61508-3, Table A.3 (2) ‘Strongly typed programming language’ IEC 61508-3, Table A.3 (3) - Language subset IEC 61508-3, Table A.4 (5) - Design and coding standards • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) - 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1e) - 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) - 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) - 'Use of style guides' • EN 50128, Table A.3 (1) - Defensive Programming EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.4 (11) - Language Subset
See Also	“Specify Properties of Truth Table Functions” (Stateflow)
Last Changed	R2017b

Chart Architecture

In this section...

hisf_0004: Protect against recursive function calls to improve code compliance

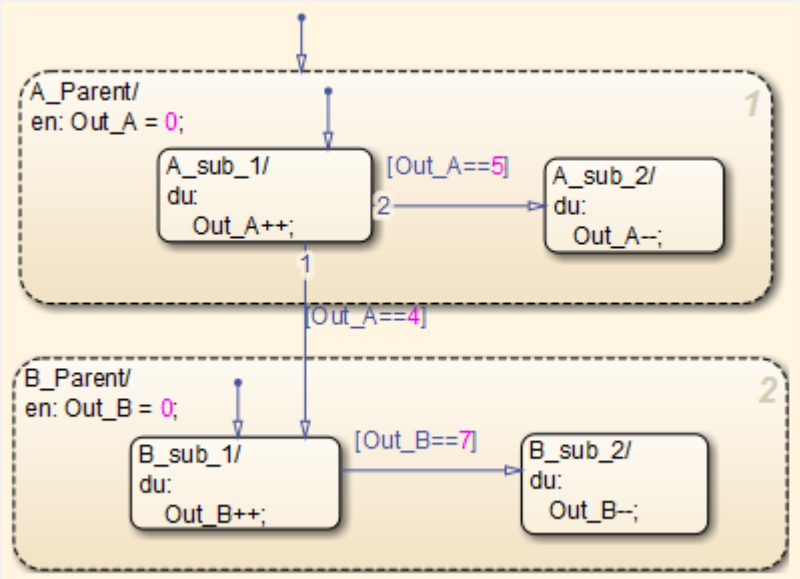
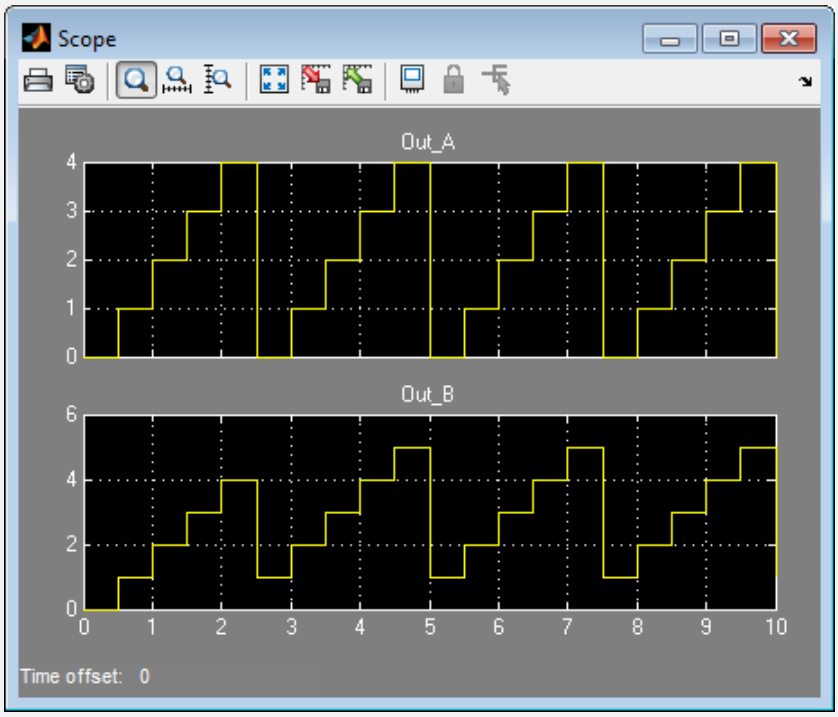
ID: Title	hisf_0004: Protect against recursive function calls to improve code compliance
Description	To improve compliance of generated code, do not call functions recursively. This includes any combination of graphical functions, truth table functions, MATLAB functions, or Simulink functions.
Prerequisites	• “hisf_0011: Stateflow debugging settings” on page 3-3 • “hisl_0311: Configuration Parameters > Diagnostics > Stateflow” on page 5-16 • “hisl_0060: Configuration parameters that improve MISRA C:2012 compliance” on page 7-15
Notes	A recursion exists when a function calls itself directly or indirectly through another function call.
Rationale	Promote bounded function call behavior.
Model Advisor Checks	“Check usage of recursions” (Simulink Check)
References	• IEC 61508-3, Table B.1 (6) 'Limited use of recursion' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 6 (1j) 'No recursions' • EN 50128, Table A.12 (6) 'Limited Use of Recursion' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' DO-331, Section MB.6.3.3.d 'Software architecture is verifiable' • MISRA C:2012, Rule 17.2
Last Changed	R2021a

ID: Title	hisf_0004: Protect against recursive function calls to improve code compliance
Examples	There are multiple patterns in Stateflow that can result in recursion.  **Recursive Function Calls**
	When the default state A is entered, event Evn is broadcast in the entry action of A. Evn results in a recursive call of the interpretation algorithm. Since A is active, the outgoing transition of A is tested. Since the current event Evn matches the transition event (and because of the absence of condition) the condition action is executed, broadcasting Evn again. This results in a new call of the interpretation algorithm which repeats the same sequence of steps until stack overflow.  **Recursive Function Calls**

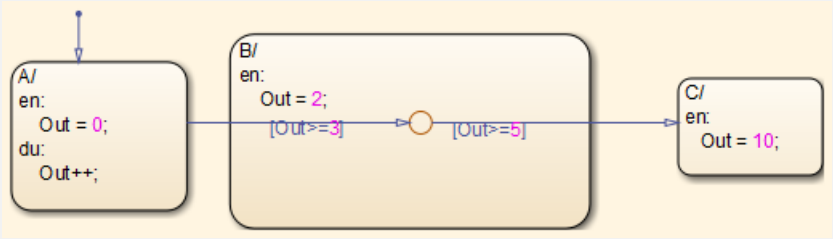
hisf_0013: Usage of transition paths (crossing parallel state boundaries)

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	“Check Stateflow charts for transition paths that cross parallel state boundaries” (Simulink Check)	

ID: Title	hisf_0013: Usage of transition paths (crossing parallel state boundaries)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'
Last Changed	R2017b

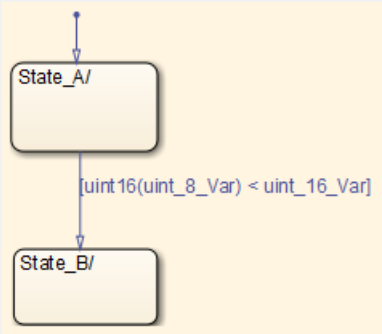
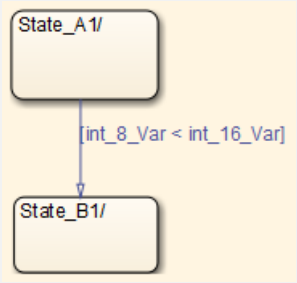
ID: Title	hisf_0013: Usage of transition paths (crossing parallel state boundaries)
Example	In the following example, when `Out_A` is 4, both parent states (`A_Parent` and `B_Parent`) are reentered. Reentering the parent states resets the values of `Out_A` and `Out_B` to zero.  

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	“Check for inappropriate use of transition paths” (Simulink Check)	
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • DDO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent'	
Last Changed	R2018b	
Examples		

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

ID: Title	hisf_0015: Strong data typing (casting variables and parameters in expressions)	
Description	To facilitate strong data typing,	
	A	Explicitly type cast variables and parameters of different data types in: • Transition conditions • Transition actions • State actions
Notes	The Stateflow software automatically casts variables of different type into the same data type. This guideline helps clarify data types of the intermediate variables.	
Rationale	A	Apply strong data typing.
Model Advisor Checks	“Check Stateflow charts for strong data typing” (Simulink Check)	

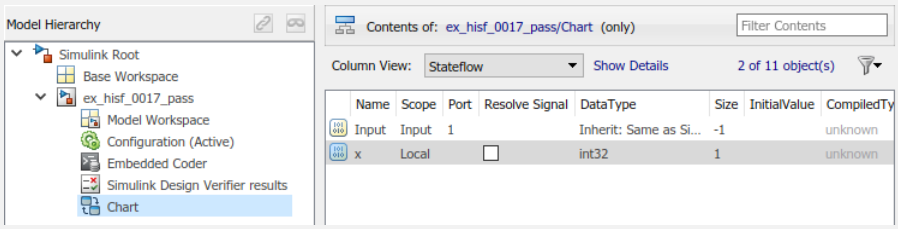
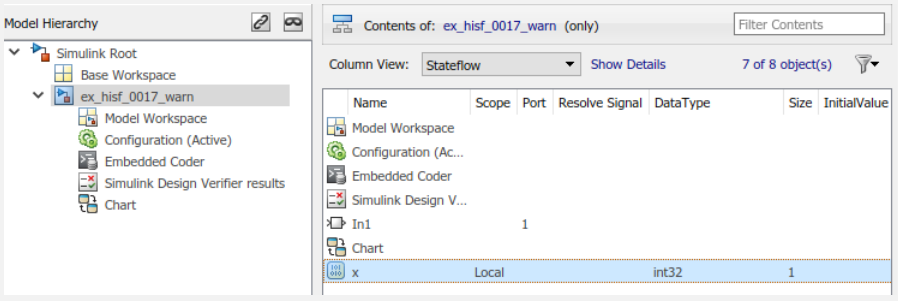
ID: Title	hisf_0015: Strong data typing (casting variables and parameters in expressions)
References	• IEC 61508-3, Table A.3 (2) ‘Strongly typed programming language’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Rule 10.1 MISRA C:2012, Rule 12.2
Last Changed	R2021a
Examples	 **Recommended**  **Not Recommended**

hisf_0016: Stateflow port names

ID: Title	hisf_0016: Stateflow port names
Description	The name of a Stateflow input or output must be the same as the corresponding signal. An exception to the guideline is that reusable Stateflow blocks can have different port names.
Rationale	Support generation of traceable code.
Model Advisor Checks	“Check naming of ports in Stateflow charts” (Simulink Check)

ID: Title	hisf_0016: Stateflow port names
References	• DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'
Last Changed	2018a

hisf_0017: Stateflow data object scoping

ID: Title	hisf_0017: Stateflow data object scoping
Description	Stateflow data objects with local scope must be defined at the chart level or below.
Rationale	Support generation of traceable code.
Model Advisor Checks	“Check scoping of Stateflow data objects” (Simulink Check)
References	• DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'
Last Changed	2018a
Examples	 **Recommended**  **Not Recommended**

MATLAB Function and MATLAB Code Considerations

MATLAB Functions

In this section...

himl_0001: Usage of standardized MATLAB function headers

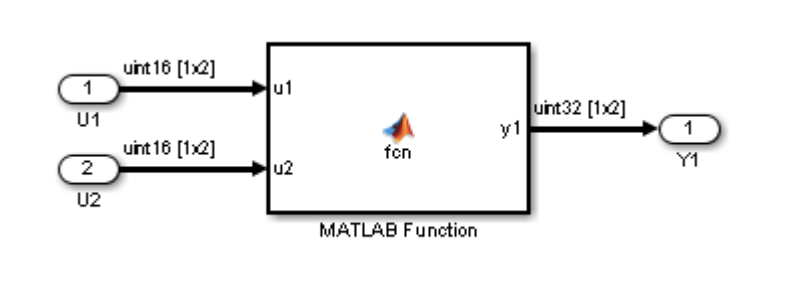
ID: Title	himl_0001: Usage of standardized MATLAB function headers
Description	When using MATLAB functions, use a standardized header to provide information about the purpose and use of the function.
Rationale	A standardized header improves the readability and documentation of MATLAB functions. The header should provide a function description and usage information.
Model Advisor Checks	“Check usage of standardized MATLAB function headers” (Simulink Check)
References	• DO-331, Section MB.6.3.4.e – Source code is traceable to low-level requirements • ISO 26262-6, Table 1 (1g) – Use of style guides
See Also	• MathWorks Advisory Board (MAB) guideline na_0025: MATLAB Function header • Orion GN&C: MATLAB and Simulink Standards, jh_0073: eML Header • **MATLAB Function Block Editor**
Last Changed	R2018b

ID: Title	himl_0001: Usage of standardized MATLAB function headers

Examples	A typical standardized function header includes:
	• Function name
	• Description
	• Inputs and outputs (if possible, include size and type)
	• Assumptions and limitations
	• Revision history
	Example:
	`% FUNCTION NAME:`
	`%   avg`
	`%`
	`% DESCRIPTION:`
	`%   Compute the average of three inputs`
	`%`
	`% INPUT:`
	`%   in1 - (double) Input one`
	`%   in2 - (double) Input two`
	`%   in3 - (double) Input three`
	`%`
	`% OUTPUT:`
	`%   out - (double) Calculated average of the three inputs`
	`%`
	`% ASSUMPTIONS AND LIMITATIONS:`
	`%   None`
	`%`
	`% REVISION HISTORY:`
	`%   05/02/2018 - mmyers`
	`%       * Initial implementation`
	`%`

himl_0002: Strong data typing at MATLAB function boundaries

ID: Title	himl_0002: Strong data typing at MATLAB function boundaries
Description	To support strong data typing at the interfaces of MATLAB functions, explicitly define the interface for input signals, output signals, and parameters, by setting: • Complexity • Type
Rationale	Defined interfaces: • Allow consistency checking of interfaces. • Prevent unintended generation of different functions for different input and output types. • Simplify testing of functions by limiting the number of test cases.
Model Advisor Checks	“Check for MATLAB Function interfaces with inherited properties” (Simulink Check)

ID: Title	himl_0002: Strong data typing at MATLAB function boundaries
References	• IEC 61508-3, Table B.9 (6) - Fully defined interface • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1c) - Enforcement of strong typing ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation • EN 50128, Table A.1 (11) - Software Interface Specifications • DO-331, Section MB.6.3.2.b - Low-level requirements are accurate and consistent
See Also	• MathWorks Advisory Board (MAB) guideline na_0034: MATLAB Function block input/output settings • Orion GN&C: MATLAB and Simulink Standards, jh_0063: eML block input / output settings • **MATLAB Function Block Editor**
Last Changed	R2016a
Examples	**Recommended:** Specify the complexity and type of input `u1` as follows: • **Complexity** to `Off` • **Type** to `uint16`  **Not Recommended:** Do *not* specify the complexity and type of input `u1` as follows: • **Complexity** to `Inherited` • **Type** to `Inherit: Same as Simulink`. **Note** To modify the input, from the toolbar of the **MATLAB Function Block Editor**, select **Edit Data**.

himl_0003: Complexity of user-defined MATLAB Functions

<table>
<tr><th>ID: Title</th><th>himl_0003: Complexity of user-defined MATLAB Functions</th></tr>
<tr><td>Description</td><td>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)</td></tr>
<tr><td>Note</td><td>Size and complexity limits can vary across projects. Typical limits might be as described in this table:<table><tr><th>Metric</th><th>Limit</th></tr><tr><td>Lines of code</td><td>60 per MATLAB function</td></tr><tr><td>Nested function levels</td><td>3[1,2]</td></tr><tr><td>Cyclomatic complexity</td><td>15</td></tr><tr><td>Density of comments</td><td>0.2 comment lines per line of code</td></tr><tr><td colspan="2">[1]Pure Wrappers to external functions are not counted as separate levels.
[2]Standard MATLAB library functions do not count as separate levels.</td></tr></table></td></tr>
<tr><td>Rationale</td><td>Readability
Comprehension
Traceability
Maintainability
Testability</td></tr>
<tr><td>Model Advisor Checks</td><td>“Check MATLAB Function metrics” (Simulink Check)</td></tr>
<tr><td>References</td><td>IEC 61508-3, Table B.9 (6) - Fully defined interface
IEC 62304, 5.5.3 - Software Unit acceptance criteria
ISO 26262-6, Table 1 (1a) - Enforcement of low complexity
ISO 26262-6, Table 1 (1f) - Use of unambiguous graphical representation
EN 50128, Table A.1(11) - Software Interface Specifications
DO-331, Sections MB.6.3.1.e - High-level requirements conform to standards
DO-331, Sections MB.6.3.2.e - Low-level requirements conform to standards</td></tr>
<tr><td>See Also</td><td>MathWorks Advisory Board (MAB) guidelines:
na_0016: Source lines of MATLAB Functions
na_0017: Number of called function levels
na_0018: Number of nested if/else and case statement
Orion GN&C: MATLAB and Simulink Standards, jh_0084: eML Comments
MATLAB Function Block Editor</td></tr>
<tr><td>Last Changed</td><td>R2021b</td></tr>
</table>

MATLAB Code

In this section...

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	“Check MATLAB Code Analyzer messages” (Simulink Check)	

ID: Title	himl_0004: MATLAB Code Analyzer recommendations for code generation
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ IEC 61508-3, Table A.4 (5) 'Design and coding standards' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) 'Use of style guides' ISO 26262-6, Table 1 (1h) 'Use of naming conventions' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' EN 50128, Table A.12 (1) 'Coding Standard' EN 50128, Table A.12 (2) 'Coding Style Guide' • 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 Using the Code Analyzer”
Last Changed	R2016a

ID: Title	himl_0004: MATLAB Code Analyzer recommendations for code generation

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`.

ID: Title	himl_0006: MATLAB code if / elseif / else patterns
Rationale	• Defensive programming • Readability • Traceability
Model Advisor Checks	“Check if/elseif/else patterns in MATLAB Function blocks” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.e 'Conformance to standards' DO-331, Section MB.6.3.2.e 'Conformance to standards' DO-331, Section MB.6.3.3.e 'Conformance to standards'
See Also	• “hisl_0010: Usage of If blocks and If Action Subsystem blocks” on page 2-19
Last Changed	R2018b

ID: Title	himl_0006: MATLAB code if / elseif / else patterns

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.

ID: Title	himl_0007: MATLAB code switch / case / otherwise patterns
Rationale	• Defensive programming • Readability • Traceability
Model Advisor Checks	“Check switch statements in MATLAB Function blocks” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • DO-331, Section MB.6.3.1.e 'Conformance to standards' DO-331, Section MB.6.3.2.e 'Conformance to standards' DO-331, Section MB.6.3.3.e 'Conformance to standards' • MISRA C: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” on page 2-21
Last Changed	R2018b

ID: Title	himl_0007: MATLAB code switch / case / otherwise patterns

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

ID: Title	himl_0008: MATLAB code relational operator data types
Model Advisor Checks	“Check usage of relational operators in MATLAB Function blocks” (Simulink Check)
References	• DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.4 (11) 'Language Subset'
See Also	• “hisl_0016: Usage of blocks that compute relational operators” on page 2-35 • “hisl_0017: Usage of blocks that compute relational operators (2)” on page 2-37
Last Changed	R2018b

Examples	**Recommended**
	• `myBool == true`
	`myInt8 == int8(1)`
	Not Recommended
	• `myBool == 1`
	`myInt8 == true`
	`myInt8 == 1`
	`myInt8 == int16(1)`
	`myEnum1.EnumVal == int32(1)`

himl_0010: MATLAB code with logical operators and functions

ID: Title	himl_0010: MATLAB code with logical operators and functions
Description	For logical operators and logical functions in MATLAB code, use logical data types
Notes	Logical operators: &&, `||`, ~ Logical functions: `and`, `or`, `not`, `xor`
Rationale	• Prevent unexpected results
Model Advisor Checks	“Check usage of logical operators and functions in MATLAB Function blocks” (Simulink Check)

ID: Title	himl_0010: MATLAB code with logical operators and functions
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate'
Last Changed	R2018b

Examples	**Recommended**
	• `~myLogical`
	`(myInt8 > int8(4)) && myLogical`
	`xor(myLogical1,myLogical2)`
	Not Recommended
	• `~myInt8`
	`myInt8 && myDouble`

himl_0012: Usage of MATLAB functions for code generation

ID: Title	himl_0012: Usage of MATLAB functions for code generation
Description	Use only MATLAB functions that support code generation.
Rationale	To detect and avoid the usage of MATLAB functions which are not supported by code generation at earliest possible stages of development.
Model Advisor Checks	“Check MATLAB functions not supported for code generation” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' DO-331, Section MB.6.3.1.b 'Accuracy and consistency’ DO-331, Section MB.6.3.2.b 'Accuracy and consistency’
See Also	`coder.screener`“Functions”
Last Changed	R2021b

himl_0013: Limitation of built-in MATLAB Function complexity

ID: Title	himl_0013: Limitation of built-in MATLAB Function complexity
Description	When authoring MATLAB code, limit the usage of built-in MATLAB functions that may result in generated code that exceeds complexity limits established for your project.

<table>
<tr><th>ID: Title</th><th>himl_0013: Limitation of built-in MATLAB Function complexity</th></tr>
<tr><td>Notes</td><td>Complexity limits can vary across projects. Typical limits might be as described in this table:<table><tr><td>Metric</td><td>Limit</td></tr><tr><td>Cyclomatic Complexity (Generated Code)</td><td>10</td></tr></table></td></tr>
<tr><td>Rationale</td><td>Improve testability and maintainability.</td></tr>
<tr><td>Model Advisor Checks</td><td>“Metrics for generated code complexity” (Simulink Check)</td></tr>
<tr><td>References</td><td>• ISO 26262-6, Table 2 (1a) - 'Natural language'
ISO 26262-6, Table 3 (1b) – ‘Restricted size and complexity of software components’
ISO 26262-6, Table 1 (1a) - Enforcement of low complexity</td></tr>
<tr><td>Last Changed</td><td>R2021b</td></tr>
</table>

himl_0011: Data type and size of condition expressions

ID: Title	himl_0011: Data type and size of condition expressions
Description	Logical scalars should be used for condition expressions. Condition expressions include: • `if` expressions • `elseif` expressions • `while` expressions • Condition expressions of Stateflow transitions
Rationale	Prevent execution of unexpected code paths
Model Advisor Checks	“Check type and size of condition expressions” (Simulink Check)
References	• IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.4 (11) 'Language Subset' • DO-331, Section MB.6.3.1.g 'Algorithms are accurate' DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012 Rule 14.4 - The controlling expression of an `if` statement and the controlling expression of an `iteration-statement` shall have `essential Boolean` type.
Last Changed	R2019b

ID: Title	himl_0011: Data type and size of condition expressions

Examples	**Recommended**
	
	Assume variable `var` is a scalar of type double with value `-1`.
	
	MATLAB Code:
	
	`if var > 0 % expression is a logical scalar`
	`    … % will not be executed`
	`elseif var < 0 % expression is a logical scalar`
	`    … % will be executed`
	`else`
	`    … % will not be executed`
	`end`
	`while var < 5 % expression is a logical scalar`
	`    var = var + 1; % executed 5 times`
	`end`
	
	Stateflow Transition Condition:
	
	`[var > 0]{…} % condition action will not be executed`
	
	Not Recommended
	
	Assume variable `var` is a scalar of type double with value `-1`.
	
	MATLAB Code:
	
	`if var % expression is a double scalar`
	`    … % will be executed because var is non-zero`
	`elseif ~var`
	`    … % will not be executed`
	`else`
	`    … % will not be executed`
	`end`
	`while var % expression is a double scalar`
	`    var = var + 1; % executed 1 time`
	`end`
	
	Stateflow Transition Condition:
	
	`[var]{…} % condition action will be executed because var is non-zero`

Configuration Parameter Considerations

Solver

In this section...

hisl_0040: Configuration Parameters > Solver > Simulation time

ID: Title	hisl_0040: Configuration Parameters > Solver > Simulation time	
Description	Set these simulation time configuration parameters as follows:	
	A	**Start time** to `0.0`.
	B	**Stop time** to a positive value that is less than the value of **Application lifespan (days)**.
Note	Simulink allows nonzero start times for simulation. However, production code generation requires a zero start time. **Stop time** in seconds and **Application lifespan (days)** is in days. When configuration parameter **Application lifespan (days)** is set to `auto` (default), any positive value for **Stop time** is valid.	
Rationale	A	Generate code that is valid for production code generation.
Model Advisor Checks	“Check safety-related solver settings for simulation time” (Simulink Check)	
References	• DO-331 Section MB.6.3.1.g—Algorithms are accurate DO-331 Section MB.6.3.2.g—Algorithms are accurate • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'	
See Also	• “hisl_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)” on page 5-5 • “Solver Pane” in 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)`.

ID: Title	hisl_0041: Configuration Parameters > Solver > Solver options	
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	“Check safety-related solver settings for solver options” (Simulink Check)	
References	• DO-331 Section MB.6.3.1.g—Algorithms are accurate DO-331 Section MB.6.3.2.g—Algorithms are accurate • IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'	
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 multitasking models that are deployed with a preemptive (interruptable) operating system: • Clear model configuration parameter **Automatically handle rate transition for data transfer**. • Explicitly model rate transitions by using a Rate Transition block and selecting block parameter **Ensure data integrity during data transfer**. This guideline applies to rate-based models only.
Notes	Selecting the **Automatically handle rate transition for data transfer** parameter can insert rate transition code without a corresponding model construct. This can impede establishing full traceability or showing that only intended functions are included in generated code. You can select or clear the **Higher priority value indicates higher task priority** model configuration parameter. Selecting this parameter determines whether the **Sample time priorities** parameter assumes lower values indicate higher priorities or higher values indicate higher priorities.
Rationale	Support fully specified models and unambiguous code.
Model Advisor Checks	“Check safety-related solver settings for tasking and sample-time” (Simulink Check)
References	• DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements’ • IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'

ID: Title	hisl_0042: Configuration Parameters > Solver > Tasking and sample time options
See Also	“Solver Pane”
Last Changed	R2023a

Math and Data Types

hisl_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)

ID: Title	hisl_0045: Configuration Parameters > Math and Data Types > Implement logic signals as Boolean data (vs. double)
Description	To support unambiguous behavior when using logical operators, relational operators, and the Combinatorial Logic block, select configuration parameter **Implement logic signals as Boolean data (vs. double)**.
Notes	Selecting **Implement logic signals as Boolean data (vs. double)** enables Boolean type checking, which produces an error when blocks that prefer Boolean inputs connect to double signals. This checking results in generating code that requires less memory.
Rationale	Avoid ambiguous model behavior and optimize memory for generated code.
Model Advisor Checks	“Check safety-related optimization settings for logic signals” (Simulink Check)
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' • IEC 61508-3, Table A.3 (2) 'Strongly typed programming language’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' • EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' • MISRA C:2012, Rule 10.1
See Also	**Implement logic signals as Boolean data (vs. double)** in the Simulink documentation.
Last Changed	R2018b

hisl_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)

ID: Title	hisl_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)
Description	To support the robustness of systems that run continuously, set configuration parameter **Application lifespan (days)** to `inf`.
Notes	Embedded applications might run continuously. Do not assume a limited lifespan for timers and counters. When you set **Application lifespan (days)** to `inf`, the simulation time is less than the application lifespan.
Rationale	Support robustness of systems that run continuously.
Model Advisor Checks	“Check safety-related optimization settings for application lifespan” (Simulink Check)

ID: Title	hisl_0048: Configuration Parameters > Math and Data Types > Application lifespan (days)
References	• DO-331, Section MB.6.3.1.g 'Algorithms are accurate’ DO-331, Section MB.6.3.2.g 'Algorithms are accurate’ • IEC 61508-3, Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming'
See Also	• “Application lifespan (days)” in the Simulink documentation • “hisl_0040: Configuration Parameters > Solver > Simulation time” on page 5-2
Last Changed	R2018b

Diagnostics

In this section...

hisl_0036: Configuration Parameters > Diagnostics > Saving

ID: Title	hisl_0036: Configuration Parameters > Diagnostics > Saving
Description	Set these configuration parameters to `error`: • **Block diagram contains disabled library links** • **Block diagram contains parameterized library links**
Rationale	Prevent unexpected results.
Model Advisor Checks	“Check safety-related diagnostic settings for saving” (Simulink Check)
References	• DO-331, Section MB.6.3.3.b ‘Software architecture is consistent’ • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' • EN 50128, Table A.4 (11) 'Language Subset'
See Also	“Model Configuration Parameters: Diagnostics”
Last Changed	R2021a

hisl_0043: Configuration Parameters > Diagnostics > Solver

<table>
<tr><th>ID: Title</th><th>hisl_0043: Configuration Parameters > Diagnostics > Solver</th></tr>
<tr><td>Description</td><td>In the Configuration Parameters dialog box, on the Diagnostics pane, set the Solver parameters as follows:
Algebraic loop to <code>error</code>.
Minimize algebraic loop to <code>error</code>.
Block priority violation to <code>error</code> if you are using block priorities.
Automatic solver parameter selection to <code>error</code>.
State name clash to <code>warning</code>.</td></tr>
<tr><td>Note</td><td>Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.

This table clarifies the result of not specifying the configuration parameter as indicated above.<table><tr><th>Configuration Parameter</th><th>Result</th></tr><tr><td>Algebraic loop</td><td>Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.</td></tr><tr><td>Minimize algebraic loop</td><td>Automatic breakage of algebraic loops can go undetected and might result in unpredictable block order execution.</td></tr><tr><td>Block priority violation</td><td>Block execution order can include undetected conflicts that might result in unpredictable block order execution.</td></tr><tr><td>Automatic solver parameter selection</td><td>An automatic change to the solver, step size, or simulation stop time can go undetected and might impact the operation of generated code.</td></tr><tr><td>State name clash</td><td>A name being used for more than one state might go undetected.</td></tr></table></td></tr>
<tr><td>Rationale</td><td>Support generation of robust and unambiguous code.</td></tr>
<tr><td>Model Advisor Checks</td><td>“Check safety-related diagnostic settings for solvers” (Simulink Check)</td></tr>
<tr><td>References</td><td>DO-331, Section MB.6.3.3.b – Software architecture is consistent.
DO-331, MB.6.3.3.e 'Software architecture conforms to standards’
IEC 61508-3, Table A.3 (3) 'Language subset’
IEC 62304, 5.5.3 - Software Unit acceptance criteria
ISO 26262-6, Table 1 (1b) 'Use of language subsets'
EN 50128, Table A.4 (11) 'Language Subset'</td></tr>
<tr><td>See Also</td><td>“Model Configuration Parameters: Diagnostics” in the Simulink documentation
jc_0021: Model diagnostic settings in the Simulink documentation</td></tr>
<tr><td>Last Changed</td><td>R2018b</td></tr>
</table>

hisl_0044: Configuration Parameters > Diagnostics > Sample Time

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > Sample Time
Description	In the Configuration Parameters dialog box, on the **Diagnostics** > **Sample Time** pane, set these parameters to `error`: • “Source block specifies -1 sample time” • “Multitask data transfer” • “Single task data transfer” • “Multitask conditionally executed subsystem” • “Tasks with equal priority” • “Enforce sample times specified by Signal Specification blocks” • “Unspecified inheritability of sample time” If the target system does not allow preemption between tasks that have equal priority, set “Tasks with equal priority” to `none`.

<table>
<tr><th>ID: Title</th><th colspan="2">hisl_0044: Configuration Parameters > Diagnostics > Sample Time</th></tr>
<tr><td>Note</td><td colspan="2">Enabling diagnostics pertaining to the solver provides information to detect violations of other guidelines.

This table clarifies the result of not specifying the configuration parameter as indicated above.

<table>
<tr><th>Configuration Parameter</th><th>Result</th></tr>
<tr><td>Source block specifies -1 sample time</td><td>Use of inherited sample times for a source block, such as Sine Wave, can go undetected and result in unpredictable execution rates for source and downstream blocks.</td></tr>
<tr><td>Multitask data transfer</td><td>Invalid transfer of data between two blocks operating in multitasking mode can go undetected. You cannot use invalid data transfer for embedded real-time software applications.</td></tr>
<tr><td>Single task data transfer</td><td>The transfer of data between two blocks operating in single-tasking mode can go undetected. You cannot use single-tasking data transfer for embedded real-time software applications.</td></tr>
<tr><td>Multitask conditionally executed subsystems</td><td>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.</td></tr>
<tr><td>Tasks with equal priority</td><td>Two asynchronous tasks with equal priority might go undetected and show unexpected behavior in target systems that allow preemption.</td></tr>
<tr><td>Enforce sample times specified by Signal Specification blocks</td><td>Inconsistent sample times for a Signal Specification block and the connected destination block might go undetected and result in unpredictable execution rates.</td></tr>
<tr><td>Unspecified inheritability of sample times</td><td>An S-function that is not explicitly set to inherit sample time can go undetected and result in unpredictable behavior.</td></tr>
</table>
</td></tr>
<tr><td>Rationale</td><td>A</td><td>Support generation of robust and unambiguous code.</td></tr>
<tr><td>Model Advisor Checks</td><td colspan="2">“Check safety-related diagnostic settings for sample time” (Simulink Check)</td></tr>
</table>

ID: Title	hisl_0044: Configuration Parameters > Diagnostics > 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’ DO-331, Section MB.6.3.3.e – Software architecture conforms to standards. • IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' • EN 50128, Table A.4 (11) 'Language Subset'
See Also	“Model Configuration Parameters: Sample Time Diagnostics”
Last Changed	R2017b

hisl_0301: Configuration Parameters > Diagnostics > Compatibility

ID: Title	hisl_0301: Configuration Parameters > Diagnostics > Compatibility
Description	Set configuration parameter **S-function upgrades needed** to `error`.
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for compatibility” (Simulink Check)
References	• DO-331, Section MB.6.3.3.b – Software architecture is consistent • IEC 61508-3, Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming'
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	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set the **Parameters** parameters as follows: • **Detect downcast** to `error` • **Detect underflow** to `error` • **Detect loss of tunability** to `error` • **Detect overflow** to `error` • **Detect precision loss** to `error`
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for parameters” (Simulink Check)
References	• DO-331, Section MB.6.3.1.g – Algorithms are accurate DO-331, Section MB.6.3.2.g – Algorithms are accurate. • IEC 61508-3, Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming'
See Also	“Model Configuration Parameters: Data Validity Diagnostics” in the Simulink documentation
Last Changed	R2018b

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

ID: Title	hisl_0303: Configuration Parameters > Diagnostics > Data Validity > Merge blocks
Description	Set configuration parameter **Detect multiple driving blocks executing at the same time step** to `error`.
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for Merge blocks” (Simulink Check)
References	• DO-331 MB.6.3.2 (b) Accuracy and Consistency • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset
See Also	“Detect multiple driving blocks executing at the same time step” in the Simulink documentation
Last Changed	R2017b

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

ID: Title	hisl_0304: Configuration Parameters > Diagnostics > Data Validity > Model initialization
Description	Set configuration parameter **Underspecified initialization** to `Simplified`.
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for model initialization” (Simulink Check)
References	• DO-331, Section MB.6.3.3.b – Software architecture is consistent • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset • MISRA C:2012, Rule 9.1
See Also	“Underspecified initialization detection” in the Simulink documentation
Last Changed	R2017b

hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging

ID: Title	hisl_0305: Configuration Parameters > Diagnostics > Data Validity > Debugging
Description	Set configuration parameter **Model Verification block enabling** to `Disable all`.
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for data used for debugging” (Simulink Check)
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 • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset
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	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	“Check safety-related diagnostic settings for signal connectivity” (Simulink Check)
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' • IEC 61508-3, Table A.3 (3) - 'Language subset' • IEC 62304, 5.5.3 - 'Software Unit acceptance criteria' • ISO 26262-6, Table 1 (1b) - 'Use of language subsets' ISO 26262-6, Table 1 (1f) - 'Use of unambiguous graphical representation' • EN 50128, Table A.4 (11) - 'Language Subset'
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	In the Configuration Parameters dialog box, on the **Diagnostics** > **Connectivity** pane, set the **Buses** parameters as follows: • **Unspecified bus object at root Outport block** to `error` • **Element name mismatch** to `error` • **Bus signal treated as vector** to `error` • **Non-bus signals treated as bus signals** to `error`
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for bus connectivity” (Simulink Check)
References	• DO-331, Section MB.6.3.3.b – Software architecture is consistent • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset
See Also	“Model Configuration Parameters: Connectivity Diagnostics” in the Simulink documentation

ID: Title	hisl_0307: Configuration Parameters > Diagnostics > Connectivity > Buses
Last Changed	R2020a

hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls

ID: Title	hisl_0308: Configuration Parameters > Diagnostics > Connectivity > Function calls
Description	Set configuration parameter **Context-dependent inputs** to `error`.
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings that apply to function-call connectivity” (Simulink Check)
References	• DO-331, Section MB.6.3.3.b – Software architecture is consistent • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset
See Also	“Model Configuration Parameters: Connectivity Diagnostics” in the Simulink documentation
Last Changed	R2017b

hisl_0309: Configuration Parameters > Diagnostics > Type Conversion

ID: Title	hisl_0309: Configuration Parameters > Diagnostics > Type Conversion
Description	In the Configuration Parameters dialog box, on the **Diagnostics** > **Type Conversion** pane, set these parameters as follows: • **Unnecessary type conversion** to `warning` • **Vector/matrix block input conversion** to `error` • **32-bit integer to single precision float conversion** to `warning`
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for type conversions” (Simulink Check)

ID: Title	hisl_0309: Configuration Parameters > Diagnostics > Type Conversion
References	• DO-331, Section MB.6.3.1.g – Algorithms are accurate DO-331, Section MB.6.3.2.g – Algorithms are accurate • IEC 61508–3, Table A.3 (2) Strongly typed programming language IEC 61508–3, Table A.4 (3) Defensive programming • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) Use of language subsets ISO 26262-6, Table 1 (1c) Enforcement of strong typing ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques • EN 50128, Table A.4 (8) Strongly Typed Programming Language EN 50128, Table A.3 (1) Defensive Programming
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	In the Configuration Parameters dialog box, on the **Diagnostics** > **Model Referencing** pane, set these parameters as follows: • **Port and parameter mismatch** to `error` • **Invalid root Inport/Outport block connection** to `error` • **Unsupported data logging** to `error`
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for model referencing” (Simulink Check)
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 • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset
See Also	“Model Configuration Parameters: Model Referencing Diagnostics” in the Simulink documentation
Last Changed	R2020a

hisl_0311: Configuration Parameters > Diagnostics > Stateflow

ID: Title	hisl_0311: Configuration Parameters > Diagnostics > Stateflow
Description	On the **Diagnostics** > **Stateflow** pane, set these configuration parameters to `error`: • “Unexpected backtracking” • “Invalid input data access in chart initialization” • “No unconditional default transitions” • “Transition outside natural parent” • “Undirected event broadcasts” • “Transition action specified before condition action” • “Read-before-write to output in Moore chart” • “Absolute time temporal value shorter than sampling period” • “Self transition on leaf state” • “Execute-at-Initialization disabled in presence of input events” • “Unreachable execution path”
Rationale	Improve robustness of design and promote a clear modeling style.
Model Advisor Checks	“Check safety-related diagnostic settings for Stateflow” (Simulink Check)
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.d 'Low-level requirements are verifiable' 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' • EN 50128, Table A.4 (11) - 'Language Subset' EN 50128, Table A.12 (6) - 'Limited Use of Recursion' • IEC 62304, 5.5.3 - 'Software Unit acceptance criteria' • ISO 26262-6, Table 1 (1b) - 'Use of language subsets' ISO 26262-6, Table 8 (1j) - 'No recursions' • IEC 61508-3, Table A.3 (3) - 'Language subset' • MISRA C:2012, Rule 17.2
See Also	“Model Configuration Parameters: Stateflow Diagnostics” in the Simulink documentation
Last Changed	R2023a

hisl_0314: Configuration Parameters > Diagnostics > Data Validity > Signals

ID: Title	hisl_0314: Configuration Parameters > Diagnostics > Data Validity > Signals
Description	In the Configuration Parameters dialog box, on the **Diagnostics** > **Data Validity** pane, set the **Signals** parameters as follows: • **Signal resolution** to `Explicit only` • **Division by singular matrix** to `error` • **Underspecified data types** to `error` • **Inf or NaN block output** to `error` • **“rt” prefix for identifiers** to `error` • **Wrap on overflow** to `error` • **Saturate on overflow** to `error` • **Simulation range checking** to `error`
Rationale	Improve robustness of design.
Model Advisor Checks	“Check safety-related diagnostic settings for signal data” (Simulink Check)
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' • IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Dir 4.1
See Also	“Model Configuration Parameters: Data Validity Diagnostics”
Last Changed	R2018a

Hardware Implementation

hisl_0071: Configuration Parameters > Hardware Implementation >Inconsistent hardware implementation settings

ID: Title	hisl_0071: Configuration Parameters > Hardware Implementation >Inconsistent hardware implementation settings
Description	Inconsistencies or under-specification of hardware attributes can result in incompatible code generation for production hardware. For compatible code generation, these configuration parameters must be the same between production hardware and test hardware: • **Byte ordering** • **Signed integer division rounds to**
Notes	Simulink and Simulink Coder™ require two sets of target specifications. The first set describes the final intended production target. The second set describes the currently selected target. If the configuration parameters do not match, the code generator creates extra code to emulate the behavior of the production target. Inconsistent hardware parameters between production hardware and test hardware can be avoided by selecting configuration parameter **Test hardware is the same as production hardware**.
Rationale	Efficient code generation
Model Advisor Check	“Check safety-related settings for hardware implementation” (Simulink Check)
References	• ISO 26262-6, Table 4 (1a) ‘Walk-through of the design’ ISO 26262-6, Table 4 (1b) ‘Inspection of the design’ ISO 26262-6, Table 7 (1a) ‘Walk-through' ISO 26262-6, Table 7 (1c) 'Inspection' ISO 26262-6, Table 7 (1n) ‘Back-to-back comparison test between model and code, if applicable ' ISO 26262-6, Table 10 (1e) 'Back-to-back comparison test between model and code, if applicable' • DO-331 MB.6.3.2.c ‘Compatibility with Target Computer’ DO-331 MB.6.3.3.c ‘Compatibility with Target Computer’
See Also	“Set Byte Ordering for Device” (Simulink Coder)
Last Changed	R2021a

Model Referencing

hisl_0037: Configuration Parameters > Model Referencing

ID: Title	hisl_0037: Configuration Parameters > Model Referencing	
Description	Set these Configuration Parameters as follows:	
	A	**Rebuild** to `Never` or `If any changes detected`.
	B	**Never rebuild diagnostic** to `Error if rebuild required`.
	C	Clear **Pass fixed-size scalar root inputs by value for code generation**.
	D	Clear **Minimize algebraic loop occurrences**.
Rationale	A	To prevent unnecessary regeneration of the code, resulting in changing only the date of the file and slowing down the build process when using model references.
	B	For safety-related applications, an error should alert model developers that the parent and referenced models are inconsistent.
	C	To prevent unpredictable data because scalar values can change during a time step.
	D	To be compatible with the recommended setting of **Single output / update function** for embedded systems code.
Model Advisor Checks	“Check safety-related model referencing settings” (Simulink Check)	
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’ • IEC 61508-3, Table A.3 (3) 'Language subset’ IEC 61508-3, Table A.4 (3) 'Defensive programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming'	
See Also	“Model Configuration Parameters: Model Referencing”	
Last Changed	R2021a	

Simulation Target

hisl_0046: Configuration Parameters > Simulation Target > Block reduction

ID: Title	hisl_0046: Configuration Parameters > Simulation Target > Block reduction
Description	To support unambiguous presentation of the generated code and support traceability between a model and generated code, clear configuration parameter **Block reduction**.
Notes	Selecting **Block reduction** might optimize blocks out of the code generated for a model. This results in requirements without associated code and violates traceability objectives.
Rationale	Supports: • Unambiguous presentation of generated code • Traceability between a model and generated code
Model Advisor Checks	“Check safety-related block reduction optimization settings” (Simulink Check)
References	• DO-331, Section MB.6.3.4.e ‘Source code is traceable to low-level requirements’ • IEC 61508-3, Clauses 7.4.7.2, 7.4.8.3, and 7.7.2.8 which require to demonstrate that no unintended functionality has been introduced • ISO 26262-6, Table 1 (1f) – Use of unambiguous graphical representation
See Also	“Block reduction” in the Simulink documentation
Last Changed	R2018b

Code Generation

In this section...

hisl_0052: Configuration Parameters > Code Generation > Optimization > Data initialization

ID: Title	hisl_0052: Configuration Parameters > Code Generation > Optimization > Data initialization
Description	Explicitly initialize variables for models that are configured with these model configuration parameter settings: • **System target file** set to an ERT-based system target file other than `autosar.tlc` • **Code interface packaging** is set to `Nonreusable function` or `Reusable function`. For these models, you can configure code generation that completely defines data and initializes internal and external data to zero by clearing these model configuration parameters: • **Remove root level I/O zero initialization** • **Remove internal data zero initialization**

ID: Title	hisl_0052: Configuration Parameters > Code Generation > Optimization > Data initialization

Notes	• If the target environment provides mechanisms to initialize I/O and state variables, consider using the target environment initialization mechanism instead of clearing the **Remove root level I/O zero initialization** and **Remove root level I/O zero initialization** parameters.
	• If a model is configured with a data code interface:
	  • For data that is mapped to a storage class that has imported data scope, Simulink selects **Remove root level I/O zero initialization** parameter. Imported data is not defined by the generated code and zero initialization of such data is the responsibility of the target environment.
	  • For data that is mapped to a storage class with exported data scope, clear the **Remove root level I/O zero initialization** parameter.
	• If a model is configured to use a service code interface, Simulink selects the **Remove root level I/O zero initialization** parameter for root-level inports and outports that are mapped to a service interface configured with the outside-execution or during-execution data communication method. The target environment defines the data for communication and is responsible for initialization. If a service interface is configured to use direct access the data communication method, the code generator applies storage classes, which results in data interface behavior.

Rationale	Support fully defined data in generated code.
Model Advisor Checks	“Check safety-related optimization settings for data initialization” (Simulink Check)
References	• DO-331, Section MB.6.3.3.b 'Software architecture is consistent’ • IEC 61508-3, Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming'
See Also	Information about these parameters in the Simulink documentation: • Remove root level I/O zero initialization (Embedded Coder) • Remove internal data zero initialization (Embedded Coder)
Last Changed	R2023a

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

ID: Title	hisl_0053: Configuration Parameters > Code Generation > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values
Description	To support verifiable code, select configuration parameter **Remove code from floating-point to integer conversions that wraps out-of-range values**

ID: Title	hisl_0053: Configuration Parameters > Code Generation > Optimization > 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 whose **Saturate on integer overflow** configuration parameter is cleared, deselecting **Remove code from floating-point to integer conversions that wraps out-of-range values** can add code that wraps out of range values, resulting in unreachable code that cannot be tested.
Rationale	Support generation of code that can be verified.
Model Advisor Checks	“Check safety-related optimization settings for data type conversions” (Simulink Check)
References	• DO-331, Section MB.6.3.1.g 'Algorithms are accurate’ DO-331, Section MB.6.3.2.g 'Algorithms are accurate’ • IEC 61508-3, Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Rule 2.1 • INT32-C. Ensure that operations on signed integers do not result in overflow
See Also	Remove code from floating-point to integer conversions that wraps out-of-range values (Simulink Coder) in the Simulink documentation
Last Changed	R2021b

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

ID: Title	hisl_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions
Description	To support the robustness of the operations, clear configuration parameter **Remove code that protects against division arithmetic exceptions**.
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. This configuration parameter is applicable only when the **System target file** is an ERT-based target.
Rationale	Protect against divide-by-zero exceptions for fixed-point code.
Model Advisor Checks	“Check safety-related optimization settings for division arithmetic exceptions” (Simulink Check)

ID: Title	hisl_0054: Configuration Parameters > Code Generation > Optimization > Remove code that protects against division arithmetic exceptions
References	• DO-331, Section MB.6.3.1.g 'Algorithms are accurate’ DO-331, Section MB.6.3.2.g 'Algorithms are accurate’ • IEC 61508-3, Table A.3 (3) 'Language Subset’ IEC 61508-3 Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1(b) 'Use of language subsets' ISO 26262-6, Table 1(d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Dir 4.1 • INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors
See Also	Remove code that protects against division arithmetic exceptions (Embedded Coder) in the Simulink documentation
Last Changed	R2021b

hisl_0056: Configuration Parameters > Code Generation > Optimization > Optimize using the specified minimum and maximum values

ID: Title	hisl_0056: Configuration Parameters > Code Generation > Optimization > Optimize using the specified minimum and maximum values
Description	To support verifiable code, clear configuration parameter **Optimize using the specified minimum and maximum values**.
Notes	Selecting **Optimize using the specified minimum and maximum values** can result in requirements without associated code and violates traceability objectives.
Rationale	Support traceability between a model and generated code.
Model Advisor Checks	“Check safety-related optimization settings for specified minimum and maximum values” (Simulink Check)
References	• DO-331 Section MB.MB.6.3.4.e 'Source code is traceable to low-level requirements’' • IEC 61508-3, Table A.4 (3) 'Defensive Programming’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques • EN 50128, Table A.3 (1) 'Defensive Programming'
See also	• Optimize using the specified minimum and maximum values (Embedded Coder) • Radio Technical Commission for Aeronautics (RTCA) for information on the DO-178C Software Considerations in Airborne Systems and Equipment Certification and related standards
Last Changed	R2018b

hisl_0038: Configuration Parameters > Code Generation > Comments

ID: Title	hisl_0038: Configuration Parameters > Code Generation > Comments	
Description	In the Configuration Parameters dialog box, on the **Code Generation > Comments** pane, select these parameters:	
	A	**Include comments**.
	B	**Simulink block comments**.
	C	**Show eliminated blocks**.
	D	**Verbose comments for 'Model default' storage class**.
	E	**Requirements in block comments**.
Rationale	A	Including comments provides good traceability between the code and the model.
	B	Including comments that describe the code for blocks provides good traceability between the code and the model.
	C	Including comments that describe the code for blocks eliminated from a model provides 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 provides good traceability between the code and the model.
Model Advisor Checks	“Check safety-related code generation settings for comments” (Simulink Check)	
References	• DO-331, Section MB.6.3.4.e 'Source code is traceable to low-level requirements‘ • IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles' • EN 50128, Table A.4 (11) 'Language Subset'	
See Also	“Model Configuration Parameters: Comments” (Embedded Coder)	
Last Changed	R2021a	

hisl_0039: Configuration Parameters > Code Generation > Interface

ID: Title	hisl_0039: Configuration Parameters > Code Generation > Interface	
Description	For models used to develop high-integrity systems, in the Configuration Parameters dialog box, on the **Code Generation > Interface** pane, set the **Software environment**, **Code interface**, and **Data exchange interface** parameters as follows:	
	A	Clear **Support: non-finite numbers**.
	B	Clear **Support: absolute time**.
	C	Clear **Support: continuous time**.
	D	Clear **Support: non-inlined S-functions**.

ID: Title	hisl_0039: Configuration Parameters > Code Generation > Interface	
	E	Clear **Classic call interface**.
	F	Select **Single output / update function**.
	G	Clear **Terminate function required**.
	H	Select **Remove error status field in real-time model data structure**.
	I	Clear **MAT-file logging**.
Rationale	A	Support for non-finite numbers is not recommended for real-time safety-related systems.
	B	Support for absolute time is not recommended for real-time safety-related systems.
	C	Support for continuous time is not recommended for real-time safety-related systems.
	D	Support for non-inlined S-functions requires support of non-finite numbers, which is not recommended 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 not recommended for real-time safety-related systems; use an ERT based target instead.
	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, which is not recommended 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	“Check safety-related code generation interface settings” (Simulink Check)	
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 • IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'	
See Also	“Model Configuration Parameters: Code Generation Interface” (Embedded Coder)	
Last Changed	R2021a	

hisl_0047: Configuration Parameters > Code Generation > Code Style

ID: Title	hisl_0047: Configuration Parameters > Code Generation > Code Style	
Description	In the Configuration Parameters dialog box, on the **Code Generation > Code Style** pane, set these parameters:	
	A	Set Parentheses level (Embedded Coder) to `Standards(Parentheses for Standards Compliance)` or `Maximum(Specify precedence with parentheses)`.
	B	Select Preserve operand order in expression (Embedded Coder).
Note	These configuration parameters are available when configuration parameter System target file (Simulink Coder) is set to `ert.tlc`.	
Rationale	A	To prevent unexpected results.
	B	To improve traceability of the generated code.
Model Advisor Checks	“Check safety-related code generation settings for code style” (Simulink Check)	
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’ • IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • MISRA C:2012, Rule 12.1	
See Also	“Model Configuration Parameters: Code Style” (Embedded Coder)	
Last Changed	R2019b	

hisl_0049: Configuration Parameters > Code Generation > Identifiers

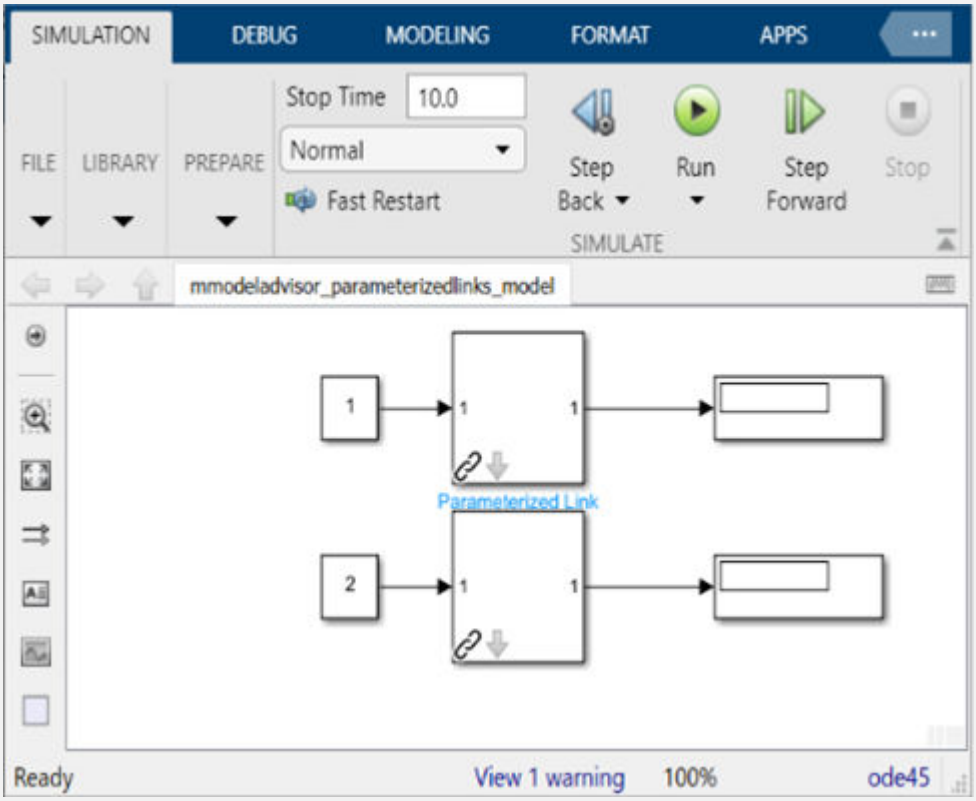
ID: Title	hisl_0049: Configuration Parameters > Code Generation > Identifiers
Description	To minimize the likelihood that parameter and signal names will change during code generation when the model changes, set configuration parameter **Minimum mangle length** to 4 or greater.
Rationale	Decrease the effort to perform code review.
Model Advisor Checks	“Check safety-related code generation identifier settings” (Simulink Check)
References	• DO-331, Section MB.6.3.4.e ‘Source code is traceable to low-level requirements’ • IEC 61508-3, Table A.3 (3) 'Language subset’ • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset'
See Also	“Model Configuration Parameters: Code Generation Identifiers” (Embedded Coder)
Last Changed	R2021a

hisl_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants

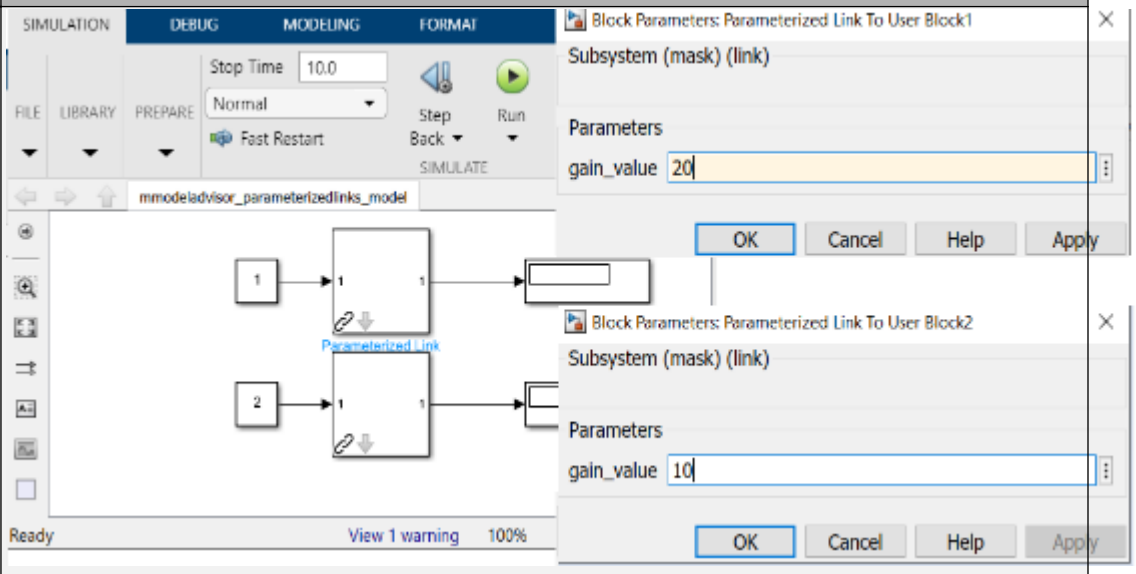
ID: Title	hisl_0074: Configuration Parameters > Diagnostics > Modeling issues related to variants
Description	Set these configuration parameters to error: • Arithmetic operations in variant conditions • Variant condition mismatch at signal source and destination
Rationale	To maintain a consistent behavior between the simulation and generated code and to prevent the creation of unused variables in generated code.
Model Advisor Checks	“Check safety-related diagnostic settings for variants” (Simulink Check)
References	• DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508–3, Table A.4 (7) 'Use of trusted / verified software modules and components' • MISRA C:2012, Rule 2.2 • ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1e) 'Use of well-trusted design principles'
See Also	• “Arithmetic operations in variant conditions” • “Prevent Creation of Unused Variables for Unconditional and Conditional Variant Choices” • “Variant condition mismatch at signal source and destination”
Last Changed	R2021b

hisl_0075: Usage of library links

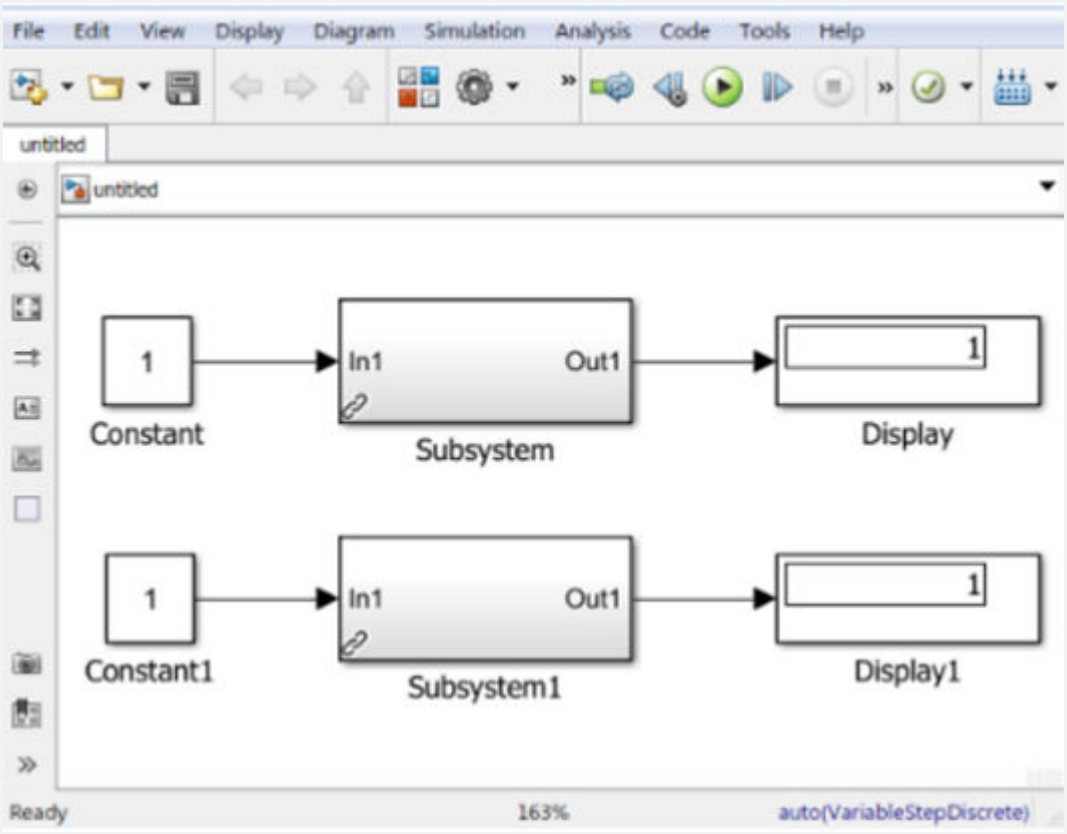
ID: Title	hisl_0075: Usage of library links
Description	To maintain a consistent behavior between the simulation and generated code, avoid Disabled library links and Parameterized library links in the model.
Rationale	To avoid unintended simulation results.
Model Advisor Checks	“Check for disabled and parameterized library links” (Simulink Check)
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’ • IEC 61508-3, Table A.3 (3) 'Language subset' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' • EN 50128, Table A.4 (11) 'Language Subset'
Last Changed	R2022a

ID: Title	hisl_0075: Usage of library links
Examples	**Recommended:** It is recommended to use masked library links to change the parameter values inside the library blocks instead of creating the parameterized library links.  Changing the gain value on masked parameter in the model doesn't create the parameterized links.

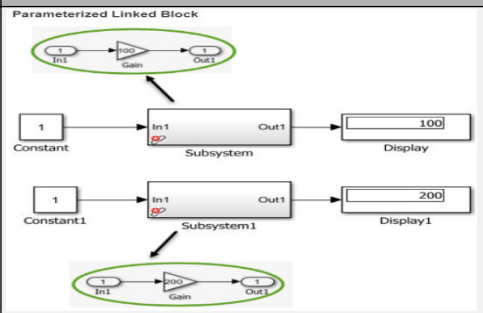
ID: Title	hisl_0075: Usage of library links



Not Recommended:



Changing the gain value using set_param in the command window cause parameterized library link. This would cause unintended behavior.

ID: Title	hisl_0075: Usage of library links
	

Naming Considerations

Naming Considerations

In this section...

hisl_0031: Model file names

ID: Title	hisl_0031: Model file names
Description	For model file names: • Use these characters: `a-z`, `A-Z`, `0-9`, and the underscore `(_)`. • Use strings that are more than 2 and less than 64 characters. (*Not including the dot and file extension*). Do not: • Start the name with a number. • Use underscores at the beginning or end of a string. • Use more than one consecutive underscore. • Use underscores in file extensions. • Use reserved identifiers.
Rationale	• Readability • Compiler limitations • Model-to-generated code traceability
Model Advisor Checks	“Check model file name” (Simulink Check)
See Also	• MAB guideline ar_0001: Usable characters for file names • MAB guideline ar_0002: Usable characters for folder names • “Reserved Keywords” (Embedded Coder)
References	• ISO 26262-6, Table 1 (1h) 'Use of naming conventions' • DO-331, Section MB.6.3.2.e – ‘Low-level requirements conform to standards’ DO-331, MB.6.3.3.e 'Software architecture conforms to standards’ • DCL37-C. Do not declare or define a reserved identifier
Last Changed	R2021b

ID: Title	hisl_0031: Model file names
Examples	**Recommended** • `My_model.slx` **Not Recommended** • `_My__model.slx` • `2018_01_11_model.slx` • `New.slx`

hisl_0032: Model element names

ID: Title	hisl_0032: Model element names
Description	For these types of model elements: • Signal labels • Parameters • Blocks • Named Stateflow objects (states, boxes, Simulink functions, graphical functions, truth tables) And, these types of architecture model objects: • Components • Ports • Connectors • Interfaces • Stereotypes 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.

ID: Title	hisl_0032: Model element names
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` • `int64`, `uint64` • `inf`, `Inf` • `NaN`, `nan` • `eps` • `intmin`, `intmax` • `realmin`, `realmax` • `pi` • `infinity` • `Nil`
Rationale	The code generator might remove or mangle invalid characters when producing an identifier in generated code.
Model Advisor Checks	“Check model object names” (Simulink Check)
See Also	MAB guidelines: • jc_0201: Usable characters for subsystem names • jc_0211: Usable characters for Inport blocks and Outport block • jc_0231: Usable characters for block names • na_0019: Restricted variable names
References	• MISRA C:2012, Rule 5.1 • MISRA C:2012, Rule 21.2 • ISO 26262-6, Table 1 (1h) 'Use of naming conventions' • DO-331, Section MB.6.3.2.e – ‘Low-level requirements conform to standards’ DO-331, MB.6.3.3.e 'Software architecture conforms to standards’ • DCL37-C. Do not declare or define a reserved identifier
Last Changed	R2023a

ID: Title	hisl_0032: Model element names
Examples	**Recommended** • Block name: `My_Controller` • Signal name: `a_b` **Not Recommended** • Block name: `My Controller` • Signal name: `12a__b`

MISRA C:2012 Compliance Considerations

Modeling Style

In this section...

hisl_0032: Model element names

ID: Title	hisl_0032: Model element names
Description	For these types of model elements: • Signal labels • Parameters • Blocks • Named Stateflow objects (states, boxes, Simulink functions, graphical functions, truth tables) And, these types of architecture model objects: • Components • Ports • Connectors • Interfaces • Stereotypes 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.

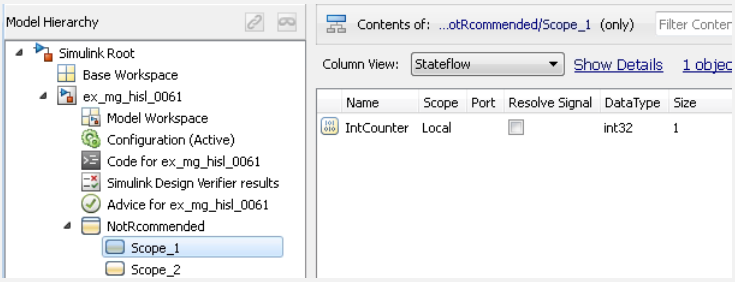
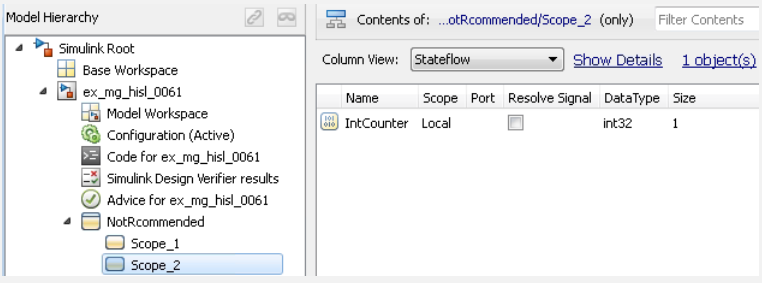
ID: Title	hisl_0032: Model element names
Notes	Reserved names: • MATLAB keywords • Reserved keywords for C, C++, and code generation. For complete list, see “Reserved Keywords” (Simulink Coder). • `int8` , `uint8` • `int16`, `uint16` • `int32`, `uint32` • `int64`, `uint64` • `inf`, `Inf` • `NaN`, `nan` • `eps` • `intmin`, `intmax` • `realmin`, `realmax` • `pi` • `infinity` • `Nil`
Rationale	The code generator might remove or mangle invalid characters when producing an identifier in generated code.
Model Advisor Checks	“Check model object names” (Simulink Check)
See Also	MAB guidelines: • jc_0201: Usable characters for subsystem names • jc_0211: Usable characters for Inport blocks and Outport block • jc_0231: Usable characters for block names • na_0019: Restricted variable names
References	• MISRA C:2012, Rule 5.1 • MISRA C:2012, Rule 21.2 • ISO 26262-6, Table 1 (1h) 'Use of naming conventions' • DO-331, Section MB.6.3.2.e – ‘Low-level requirements conform to standards’ DO-331, MB.6.3.3.e 'Software architecture conforms to standards’ • DCL37-C. Do not declare or define a reserved identifier
Last Changed	R2023a

ID: Title	hisl_0032: Model element names
Examples	**Recommended** • Block name: `My_Controller` • Signal name: `a_b` **Not Recommended** • Block name: `My Controller` • Signal name: `12a__b`

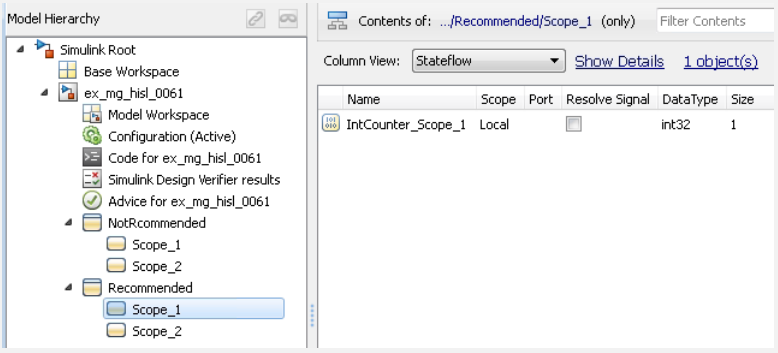
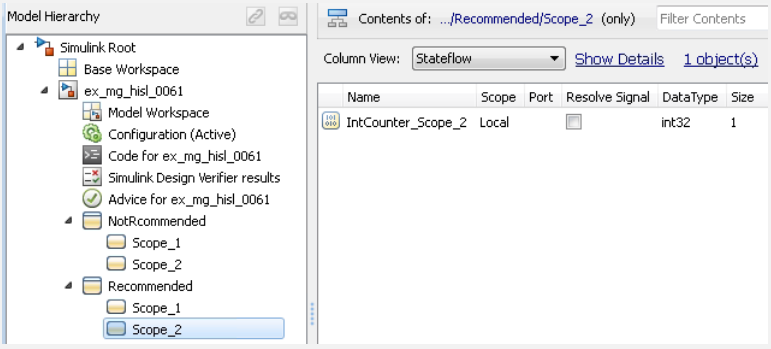
hisl_0061: Unique identifiers for clarity

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	“Check Stateflow charts for uniquely defined data objects” (Simulink Check)	
References	• DO-331, Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508-3, Table A.3 (2) ‘Strongly typed programming language’ IEC 61508-3, Table A.3 (3) - Language subset IEC 61508-3, Table A.4 (5) - Design and coding standards • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - 'Use of language subsets' ISO 26262-6, Table 1 (1c) 'Enforcement of strong typing' ISO 26262-6, Table 1 (1d) - 'Use of defensive implementation techniques' ISO 26262-6, Table 1 (1e) - 'Use of well-trusted design principles' ISO 26262-6, Table 1 (1f) - 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1g) - 'Use of style guides' ISO 26262-6, Table 1 (1h) - 'Use of naming conventions' • EN 50128, Table A.3 (1) - Defensive Programming EN 50128, Table A.4 (8) 'Strongly Typed Programming Language' EN 50128, Table A.4 (11) - 'Language Subset' EN 50128, Table A.12 (1) 'Coding Standard' EN 50128, Table A.12 (2) 'Coding Style Guide'	
See Also	“Model Configuration Set Customization” (Embedded Coder)	
Last Changed	R2017b	

ID: Title	hisl_0061: Unique identifiers for clarity

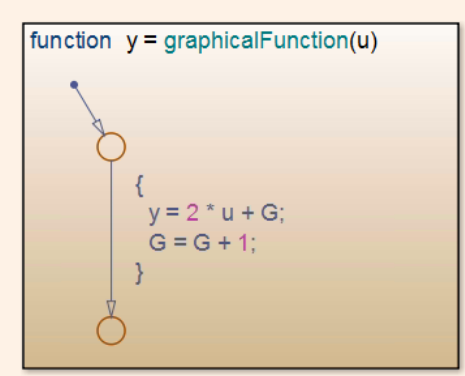
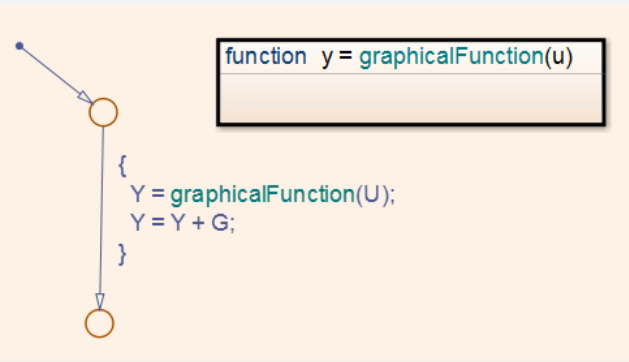
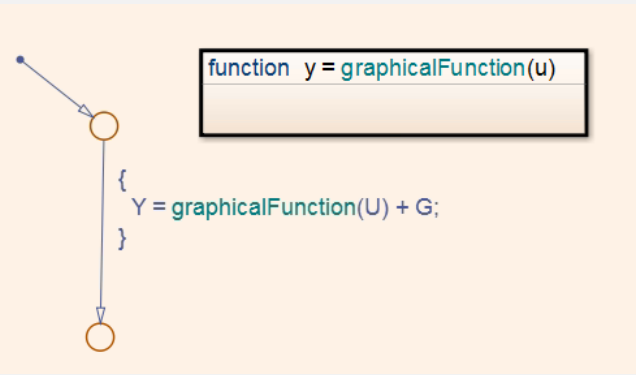
Examples	**Not Recommended**
	
	In the following example, two states `Scope_1` and `Scope_2` use local identifier `IntCounter`.
	
	`Scope_1`
	`% IntCounter is defined at this scope`
	`entry:`
	`IntCounter = int32(0);`
	`during:`
	`Chart_Level_Output_S1 = Chart_Level_Input + IntCounter;`
	`IntCounter = IntCounter + int32(1);`
	
	`Scope_2`
	`% IntCounter is defined at this scope`
	`entry:`
	`IntCounter = int32(0);`
	`during:`
	`Chart_Level_Output_S2 = Chart_Level_Input + IntCounter;`
	`IntCounter = IntCounter + int32(1);`
	
	The identifier `IntCounter` is defined for two states, `Scope_1` and `Scope_2`.
	
	 
	
	 

ID: Title	hisl_0061: Unique identifiers for clarity

	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`.
	
	`Scope_1`
	`% IntCounter_Scope_1 is defined at this scope`
	`entry:`
	`IntCounter_Scope_1 = int32(0);`
	`during:`
	`Chart_Level_Output_S1 = Chart_Level_Input + IntCounter_Scope_1;`
	`IntCounter_Scope_1 = IntCounter_Scope_1 + int32(1);`
	
	`Scope_2`
	`% IntCounter_Scope_2 is defined at this scope`
	`entry:`
	`IntCounter_Scope_2 = int32(0);`
	`during:`
	`Chart_Level_Output_S2 = Chart_Level_Input + IntCounter_Scope_2;`
	`IntCounter_Scope_2 = IntCounter_Scope_2 + int32(1);`
	
	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

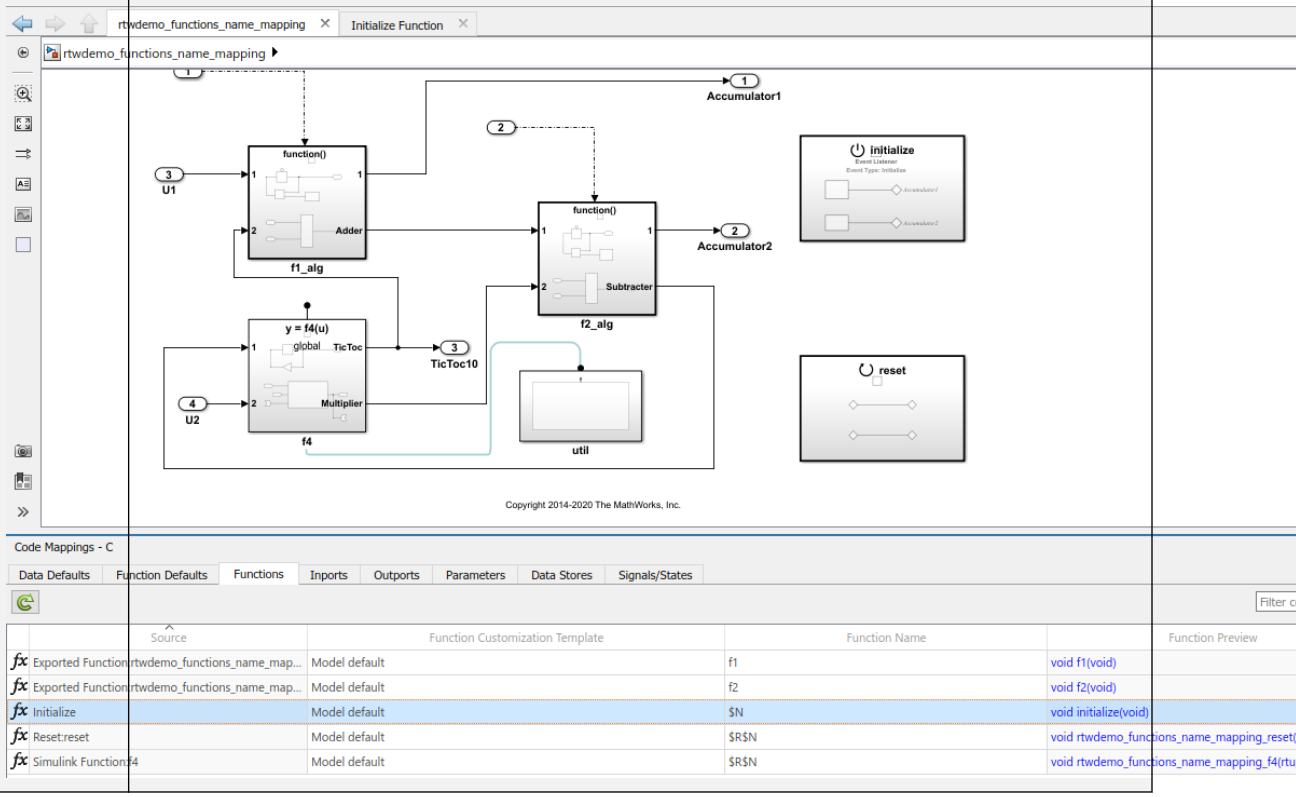
ID: Title	hisl_0062: Global variables in graphical functions
Description	For data with a global scope used in a function, do not use the data in the calling expression if a value is assigned to the data in that function.
Rationale	Enhance readability of a model by removing ambiguity in the values of global variables.
Model Advisor Checks	“Check global variables in graphical functions” (Simulink Check)
References	• IEC 61508–3, Table A.3 (3) 'Language subset' IEC 61508–3, Table A.4 (4) 'Modular approach' IEC 61508–3, A.4 (5) 'Design and coding standards' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1f) 'Use of unambiguous graphical representation' ISO 26262-6, Table 1 (1h) 'Use of naming conventions' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.12 (1) 'Coding Standard' EN 50128, Table A.12 (2) 'Coding Style Guide' • DO-331, Section MB.6.3.2.g 'Algorithms are accurate' • MISRA C:2012, Rule 13.2 MISRA C:2012, Rule 13.5 • EXP30-C. Do not depend on the order of evaluation for side effects
Last Changed	R2021b

ID: Title	hisl_0062: Global variables in graphical functions
Examples	Consider a graphical function `graphicalFunction` that modifies the global data `G`.  **Recommended**  **Not Recommended** 

hisl_0063: Length of user-defined object names to improve MISRA C:2012 compliance

ID: Title	hisl_0063: Length of user-defined object names to improve MISRA C:2012 compliance	
Description	To improve MISRA C:2012 compliance of generated code, use configuration parameter **Maximum identifier length** (`MaxIdLength`) to limit the length of user defined names. **Note** The default of Maximum identifier length is 31.	
	A	For Subsystem blocks with parameter **Function name options** set to `User specified`, limit the length of function names to be equal to or less than the value specified in **Maximum identifier length**.
	B	Limit the length of data object names to be equal to or less than the value specified in **Maximum identifier length**: • `Simulink.AliasType` • `Simulink.NumericType` • `Simulink.Variant` • `Simulink.Bus` • `Simulink.BusElement` • `Simulink.IntEnumType`
	C	When using these storage classes, limit the length of signal and parameter names to be equal to or less than the value specified in **Maximum identifier length**: • Exported Global • Imported Extern • Imported Extern Pointer • Custom storage class **Note** If specified, this includes the length of the **Identifier** name.
Rationale	Length in the generated code can result in a MISRAC:2012 violation.	
Model Advisor Checks	“Check for length of user-defined object names” (Simulink Check)	

ID: Title	hisl_0063: Length of user-defined object names to improve MISRA C:2012 compliance
References	• ISO 26262-6, Table 6 (1d) – No multiple use of variable names • DO-331, Section MB.6.3.2.e – 'Low-level requirements conform to standards' DO-331, MB.6.3.3.e 'Software architecture 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" on page 7-15
Last Changed	R2021a
Examples	You can limit the function name to the length equal to or less than the specified value using **Code mappings**. This can be used to avoid violation of MISRA rules.



Block Usage

In this section...

hisl_0020: Blocks not recommended for MISRA C:2012 compliance

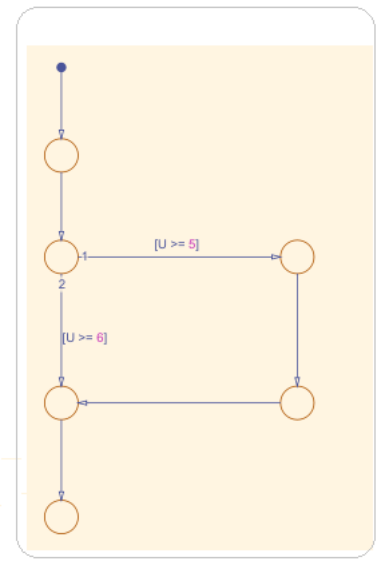
ID: Title	hisl_0020: Blocks not recommended for MISRA C:2012 compliance	
Description	To improve MISRA C:2012 compliance of the generated code:	
	A	Use only blocks that support code generation, as documented in the Simulink Block Support Table.
	B	Do not use blocks that are listed as “Not recommended for production code” in the Simulink Block Support Table.
	C	Do not use Lookup Table blocks using cubic spline interpolation or extrapolation methods. Specific blocks are: • 1-D Lookup Table • 2-D Lookup Table • n-D Lookup Table
	D	Do not use deprecated Lookup Table blocks. The deprecated Lookup Table blocks are Lookup and Lookup2D.
	E	Do not use S-Function Builder blocks in the model or subsystem.
	F	Do not use From Workspace blocks in the model or subsystem.
	G	Do not use these String blocks in the model or subsystem: • Compose String • Scan String • String to Single • String to Double • To String
Notes	If you follow this and other modeling guidelines, you can eliminate model constructs that are not suitable for C/C++ production code generation, at the same time, increase the likelihood of generating code that complies with the MISRA C:2012 standard. Use the Block Support Table block to view the Block Support Table. Blocks with the footnote (4) in the Block Support Table are classified as "Not recommended for production code".	
Rationale	A, B, C, D, E, F, G	Improve quality and MISRA C:2012 compliance of the generated code.

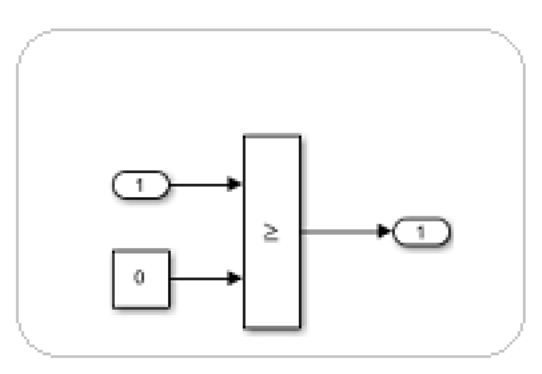
ID: Title	hisl_0020: Blocks not recommended for MISRA C:2012 compliance
Model Advisor Checks	For A,B,C, D, E, F, and G: “Check for blocks not recommended for MISRA C:2012” (Simulink Check) For A and B: “Check for blocks not recommended for C/C++ production code deployment” (Simulink Check)
References	• 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’ • IEC 61508-3, Table A.3 (3) - Language subset • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) - Use of language subsets • EN 50128, Table A.4 (11) - Language Subset • MISRA C: 2012
Last Changed	R2018b

hisl_0101: Avoid operations that result in dead logic to improve code compliance

ID: Title	hisl_0101: Avoid operations that result in dead logic to improve code compliance
Description	To improve the compliance of generated code, avoid operations that result in dead code or unreachable code.
Rationale	Enhance clarity and prevention of dead code.
Model Advisor Checks	“Check for unreachable and dead code” (Simulink Check)
References	• 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’ • ISO 26262-6, Table 1 (1e) – Use of well-trusted design principles • ISO 26262-6, Table 6 (1h) – No hidden data flow or control flow • MISRA C:2012, Rule 14.3 MISRA C:2012, Rule 2.1
Last Changed	R2022a

ID: Title	hisl_0101: Avoid operations that result in dead logic to improve code compliance

Example	**Not Recommended:**
	
	The following examples displays operations that result in dead code or unreachable code.
	
	Stateflow:
	
	 
	
	MATLAB function:
	
	Matlab function blocks and Embedded Matlab fcns
	`function y = fcn(u)`
	`if u >= 0`
	`y = u;`
	`else`
	`y = uint8(10);`
	`end`
	`end`
	
	Simulink:

ID: Title	hisl_0101: Avoid operations that result in dead logic to improve code compliance
	

hisl_0102: Data type of loop control variables to improve MISRA C:2012 compliance

ID: Title	hisl_0102: Data type of loop control variables to improve MISRA C:2012 compliance
Description	To improve MISRA C:2012 compliance of generated code, use integer data type for variables that are used as loop control counter variables in: • For loops constructed in Stateflow and MATLAB. • For Iterator blocks.
Rationale	Improve MISRA C:2012 compliance of the generated code.
Model Advisor Checks	“Check data type of loop control variables” (Simulink Check)
References	• ISO 26262-6, Table 1 (1c) - Enforcement of strong typing • DO-331, Section MB.6.3.2.g – 'Algorithms are accurate' • MISRA C:2012, Rule 14.1
Last Changed	R2018a

Configuration Settings

hisl_0060: Configuration parameters that improve MISRA C:2012 compliance

ID: Title	hisl_0060: Configuration parameters that improve MISRA C:2012 compliance

Description	Set these model configuration parameters as specified:
	• **System target file** as an ERT-based target
	• **Use division for fixed-point net slope computation** to `On` or `Use division for reciprocals of integers only`.
	• **Inf or NaN block output** to `warning` or `error`.
	• **Model Verification block enabling** to `Disable All`
	• **Undirected event broadcasts** to `error`.
	• **Wrap on overflow** to `warning` or `error`.
	• **Production hardware signed integer division rounds to** to `Zero` or `Floor`
	• **Compile-time recursion limit for MATLAB functions** to `0`.
	• **Casting Modes** to `Standards Compliant`.
	• **Code replacement library** to `None` or `AUTOSAR 4.0`
	• **Maximum identifier length** to the implementation dependent limit. The default is `31`.
	• **Parentheses level** to `Standards(Parentheses for Standards Compliance)` or `Maximum(Specify precedence with parentheses)`.
	• **Shared code placement** to `Shared location`.
	• **Language standard** to `C89/C90 (ANSI)` or `C99 (ISO)`, depending on the toolchain.
	• **Bitfield declarator type specifier** to `uint_T` when any of these parameters are selected:
	  • **Pack Boolean data into bitfields**
	  • **Use bitsets for storing state configuration**
	  • **Use bitsets for storing Boolean data**
	Select (`on`) these configuration parameters:
	• **Include Comments**
	• **MATLAB user comments**
	• **Preserve static keyword in function declarations** (Select only when configuration parameter **File packaging format** is set to `Compact` or `CompactWithDataFile`.)
	Deselect (`off`) these configuration parameters:
	• **Shift right on a signed integer as arithmetic shift**
	• **Dynamic memory allocation in MATLAB functions**
	• **Enable run-time recursion for MATLAB functions**
	• **External mode**
	• **Generate shared constants**
	• **MAT-file logging**

ID: Title	hisl_0060: Configuration parameters that improve MISRA C:2012 compliance
	• **Replace multiplications by powers of two with signed bitwise shifts** • **Support complex numbers** (Only if you do not need complex number support) • **Support continuous time** • **Support non-finite numbers** • **Support non-inlined S-functions** • **Suppress generation of default cases for switch statements if unreachable** • **Use dynamic memory allocation for model initialization** (Keep this parameter selected only when configuration parameter **Code Interface Packaging** is set to `Reusable Function`.
Rationale	Improve MISRA C:2012 compliance of the generated code.
Model Advisor Checks	For High-Integrity System Modeling, see “Check configuration parameters for MISRA C:2012” (Simulink Check). For Modeling Guidelines for MISRA C:2012, see “Check configuration parameters for MISRA C:2012” (Simulink Check)
References	• IEC 61508-3, Table A.3 (3) 'Language subset’ • ISO 26262-6, Table 1 (1b) 'Use of language subsets' • EN 50128, Table A.4 (11) 'Language Subset' • MISRA C:2012 • EXP33-C. Do not read uninitialized memory • DO-331, Section MB 6.3.2.b 'Low-level requirements are accurate and consistent' DO-331, Section MB.6.3.2.c ‘Low-level requirements are compatible with target computer 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' DO-331, Section MB.6.3.3.b – Software architecture is consistent DO-331 MB.6.3.3.c ‘Compatibility with Target Computer’ DO-331, Section MB.6.3.3.d 'Software architecture is verifiable' DO-331, MB.6.3.3.e 'Software architecture conforms to standards’
Last Changed	R2023a

Stateflow Chart Considerations

In this section...

hisf_0065: Type cast operations in Stateflow to improve code compliance

ID: Title	hisf_0065: Type cast operations in Stateflow to improve code compliance
Description	In Stateflow charts that use the C action language, use the `:=` notation to protect against Stateflow casting integer and fixed-point calculations to wider data types than the input data types.
Note	If you follow this and other modeling guidelines, you increase the likelihood of generating code that complies with the coding standards.
Rationale	To avoid implicit casts in the generated code that might violate coding standards.
Model Advisor Checks	“Check assignment operations in Stateflow Charts” (Simulink Check)
References	• DO-331 Section MB.6.3.2.b 'Low-level requirements are accurate and consistent' • IEC 61508–3, Table A.3 (2) Strongly typed programming language IEC 61508–3, Table A.4 (3) Defensive programming • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) Use of language subsets ISO 26262-6, Table 1 (1c) Enforcement of strong typing ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques • EN 50128, Table A.4 (8) Strongly Typed Programming Language EN 50128, Table A.3 (1) Defensive Programming • MISRA C:2012, Rule 10.1 MISRA C:2012, Rule 12.2
Prerequisites	“hisl_0060: Configuration parameters that improve MISRA C:2012 compliance” on page 7-15
Last Changed	R2021a

hisf_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance

ID: Title	hisf_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance	
Description	To improve code compliance of the generated code:	
	A	Do not use unary minus operators on unsigned data types.

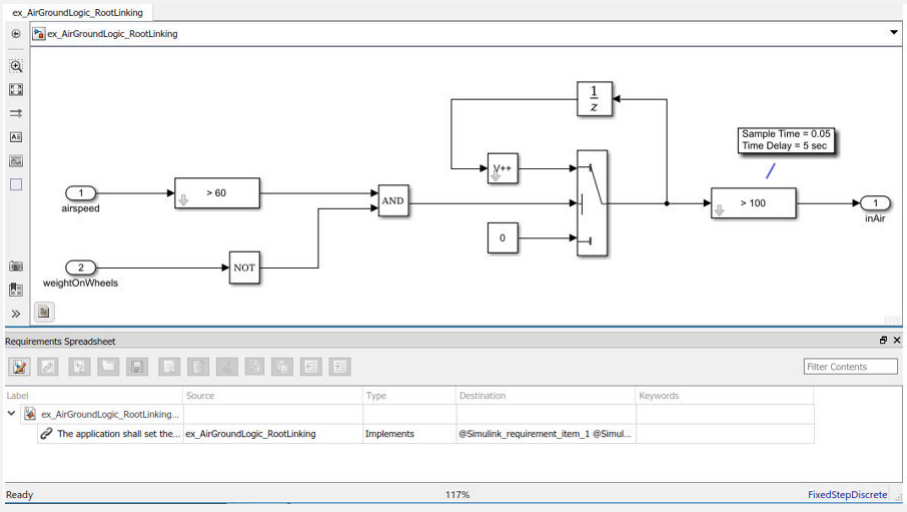
ID: Title	hisf_0211: Protect against use of unary operators in Stateflow Charts to improve code compliance
Note	The MATLAB and C action languages do not restrict the use of unary minus operators on unsigned expressions.
Rationale	Improve code compliance of the generated code.
Model Advisor Checks	“Check Stateflow charts for unary operators” (Simulink Check)
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' • IEC 61508–3, Table A.3 (2) Strongly typed programming language IEC 61508–3, Table A.4 (3) Defensive programming • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) Use of language subsets ISO 26262-6, Table 1 (1c) Enforcement of strong typing ISO 26262-6, Table 1 (1d) Use of defensive implementation techniques • EN 50128, Table A.4 (8) Strongly Typed Programming Language EN 50128, Table A.3 (1) Defensive Programming • MISRA C:2012, Rule 10.1
Last Changed	R2017b

Requirements Considerations

Requirement Considerations

hisl_0070: Placement of requirement links in a model

ID: Title	hisl_0070: Placement of requirement links in a model	
Description	Establish bidirectional traceability between model requirements and the model elements that are used to implement the requirement. A single element or combination of elements can link to requirements. When linking requirements, follow these guidelines.	
	A	Apply requirement links to the lowest level component of model elements. Model elements that do not impact the model's behavior or the generated code are exempt from requirement linking. See Notes for additional information.
	B	At the project level, define the maximum number of unique requirement links associated with each component. A minimum of one requirement link is required.
	C	At the project level, define the maximum number of child model elements for each linked component.
Notes	Use Requirements Toolbox™ to trace between the model and the requirements from which the model was developed. Apply user tags (Requirements Toolbox) to define model elements as derived and/or safety requirements. To reduce the number of requirements that are linked to a model, apply requirements at the component-level. A component contains a group of model elements, for example: • In Simulink, a component is a top-level block diagram, subsystem, MATLAB function, or area annotation. • In Stateflow, a component is a chart, superstate, box, Simulink function, graphical function, Simulink State, MATLAB Function, or Truth Table. • In MATLAB, a component is a function. • In System Composer, a Component is an Adapter or a Component block. Components that contain *only* these model elements are exempt from requirement linking: • Model Info, DocBlock, or System Requirements blocks • Area annotations • Model element with requirement links • Commented out model elements When a linked component contains a nonexempt child model element, the child implements the associated requirement either in part or whole.	
Rationale	A	Establishing requirement links at the component level captures the relationship of model elements. In addition, maintainability improves because the need to update requirement links for minor logic changes is reduced.
	B, C	Support requirement change impact analysis.

ID: Title	hisl_0070: Placement of requirement links in a model
Model Advisor Check	“Check for model elements that do not link to requirements” (Simulink Check)
References	• DO-331, Section MB.6.3.2.f - 'Low-level requirements trace to high-level requirements' • IEC 61508-3, Table A.2 (12) - 'Computer-aided specification and design tools' IEC 61508-3, Table A.2 (9) - 'Forward traceability between the software safety requirements specification and software architecture' IEC 61508-3, Table A.2 (10) - 'Backward traceability between the software safety requirements specification and software architecture' IEC 61508-3, Table A.4 (8) - 'Forward traceability between the software safety requirements specification and software design' IEC 61508-3, Table A.8 (1) - 'Impact analysis' • IEC 62304, 5.2 - 'Software requirements analysis' IEC 62304, 7.4.2 - 'Analyze impact of software changes on existing risk control measures' • ISO 26262-6, Table 2 (1a) - 'Natural language' ISO 26262-6, Table 3 (1b) – ‘Restricted size and complexity of software components’ ISO 26262-6, Table 5 (1a) – Natural language ISO 26262-6: 7.4.2.a - The verifiability of the software architectural design ISO 26262-8: 8.4.3 Change request analysis • EN 50128, Table A.3 (23) - 'Modeling supported by computer aided design and specification tools' EN 50128, Table D.58 - Traceability EN 50128, Table A.10 (1) - 'Impact Analysis'
See Also	“Requirements Traceability” (Requirements Toolbox)
Last Changed	R2021a
Examples	**Recommended: Requirement links on parent component** Requirement link placed at the top level model with no subsystems. 

ID: Title	hisl_0070: Placement of requirement links in a model
	Recommended: Requirement links placed on area annotation Requirement link placed on an area annotation.