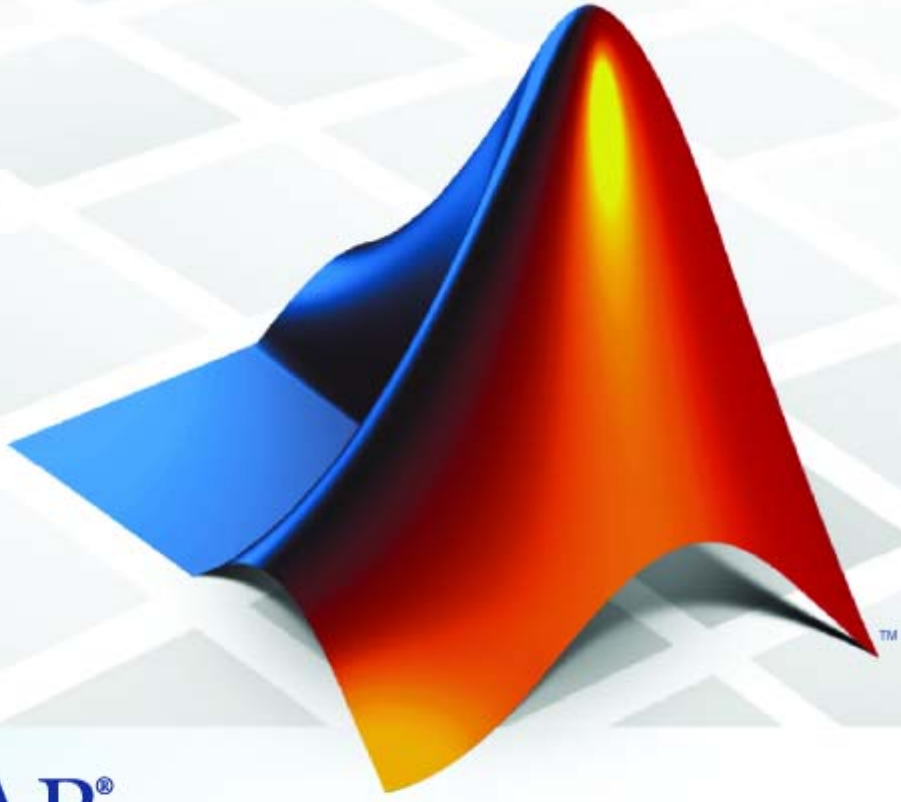
Modeling Guidelines for High-Integrity Systems

Block and Configuration Parameter Considerations



MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Modeling Guidelines for High-Integrity Systems

Trademarks

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

Patents

The MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

September 2009	Online only	New for version 1.0 (Release 2009b)

Contents

1

Introduction

Motivation

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

These guidelines do not assume that you use a particular safety or certification standard. The guidelines reference some safety standards where applicable, including DO-178B, IEC 61508, and MISRA C®.

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

This document does not address model style or development processes. For more information about creating models in a way that improves consistency, clarity, and readability, see the *MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB®, Simulink, and Stateflow® (Version 2.0).* Development process guidance and additional information for specific standards is available with the IEC Certification Kit (for IEC 61508) and DO Qualification Kit (for DO-178B) products.

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

2

Block-Specific Considerations

Math Operations

hisl_0001: Usage of Abs Block

hisl_0002: Usage of Math Function Blocks (Remainder and Reciprocal)

hisl_0003: Usage of Math Function Blocks (Square Root)

hisl_0004: Usage of Math Function Blocks (Natural Logarithm and Base 10 Logarithm)

hisl_0005: Usage of Product Blocks

hisl_0001: Usage of Abs Block

ID: Title hisl_0001: Usage of Abs block

Priority Strongly recommended

Prerequisites Not applicable

Description To support the robustness of the generated code when using Abs blocks:

- Avoid Boolean and unsigned integer data types as inputs to the Abs block.
- In the Abs block parameter dialog box, select **Saturate on integer overflow**.

Note 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. This results in an untraceable block.

For signed data types, Simulink does not represent the absolute value of the most negative value. . When you select **Saturate on integer overflow**, the absolute value of the data type saturates to the most positive representable value. When you clear **Saturate on integer overflow**, the absolute value of the most negative value represented by the data type has no effect.

Rationale

- Code Generation
- Verification and Validation
- High Integrity Systems

Notes This guideline supports adhering to:

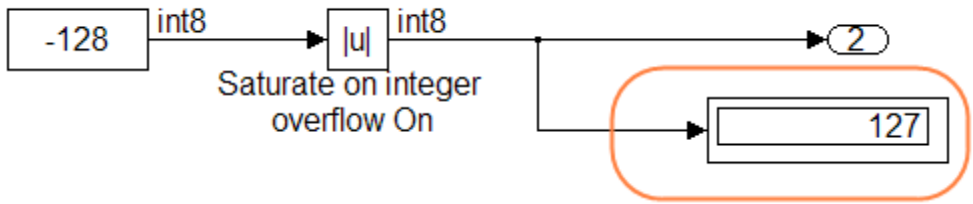
- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming';
  IEC 61508-3, Table A.3 (2) 'Strongly typed programming language';
  IEC 61508-3, Table B.8 (3) 'Control Flow Analysis'

- DO-178B, Section 6.4.4.3c 'Structural Coverage Analysis Resolution (Dead Code)'
- MISRA-C:2004, Rule 14.1;
  MISRA-C:2004, Rule 21.1

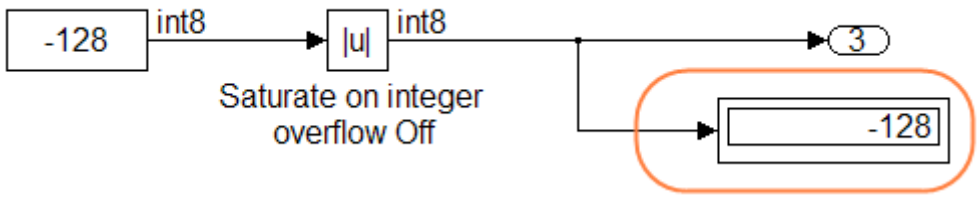
Model Advisor Checks

- **By Task > Modeling Standards for DO-178B > "Check for proper usage of blocks that compute absolute values"**
- **By Task > Modeling Standards for IEC-61508 > "Check usage of Simulink constructs"**

Example



Correct



Incorrect

hisl_0002: Usage of Math Function Blocks (Remainder and Reciprocal)

ID: Title

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

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support the robustness of the generated code, when using Math Function blocks with remainder after division (`rem`) or array reciprocal (`reciprocal`) functions:

- Protect the input of the `reciprocal` function from going to zero.
- Protect the second input of the `rem` function from going to zero.

Note When using the array reciprocal or remainder after division functions, you might get a divide by zero operation, resulting in an infinite (`Inf`) output. To avoid overflows, protect the corresponding inputs from going to zero.

Rationale

- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- MISRA-C:2004, Rule 21.1
- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.4.2.2 'Robustness Test Cases'
  DO-178B, Section 6.4.3 'Requirements-Based Testing Methods'

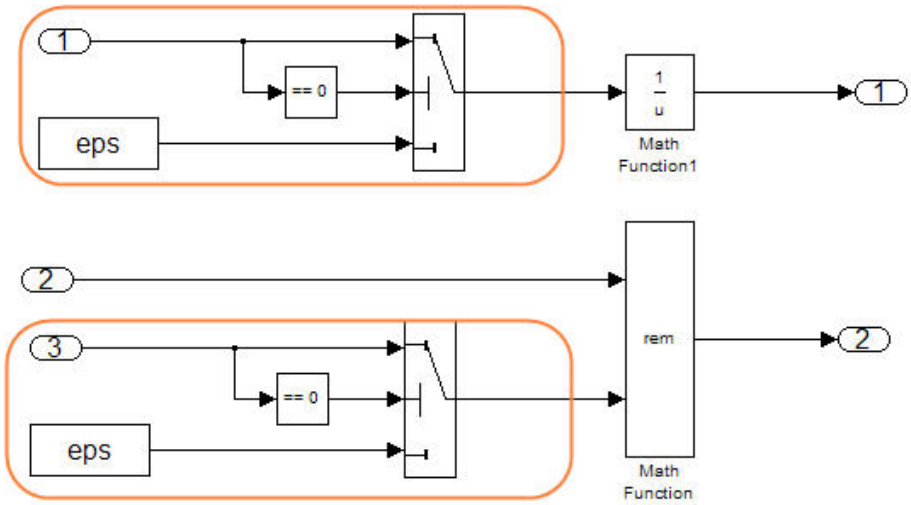
Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check for proper usage of Math blocks"

hisl_0002: Usage of Math Function Blocks (Remainder and Reciprocal)

Example

The following is a basic example of protection from zero division. When the input signal oscillates around zero, the output exhibits a large change in value. The MathWorks recommends further protection against the large change in value.



hisl_0003: Usage of Math Function Blocks (Square Root)

ID: Title

hisl_0003: Usage of Math Function blocks (square root)

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support the robustness of the generated code, when using Math Function blocks with the square root (`sqrt`) function parameter, do one of the following:

- Account for complex numbers as the output.
- Account for negative values as the block output.
- Protect the input from going negative.

Note For negative inputs, the square root function takes the absolute value of the input and performs the square root operation. The square root function sets the sign of the output to negative, which might lead to undesirable results in the generated code.

Rationale

- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

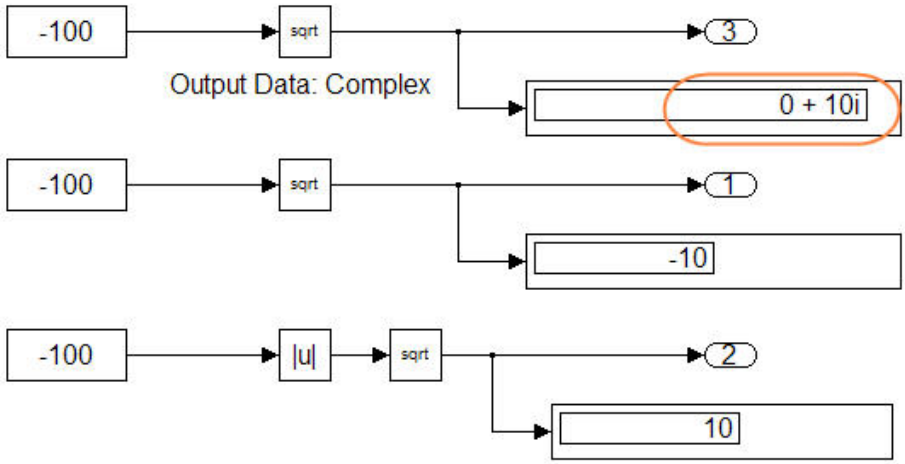
- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.4.2.2a 'Robustness Test Cases'

Model Advisor Checks

Not applicable

hisl_0003: Usage of Math Function Blocks (Square Root)

Example



hisl_0004: Usage of Math Function Blocks (Natural Logarithm and Base 10 Logarithm)

ID: Title

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

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support the robustness of the generated code, when using Math Function blocks with natural logarithm (`log`) or base 10 logarithm (`log10`) function parameters, do one of the following:

- Protect the input from going negative.
- Protect the input from equaling zero.
- Account for complex numbers as the output value.

Note 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

- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

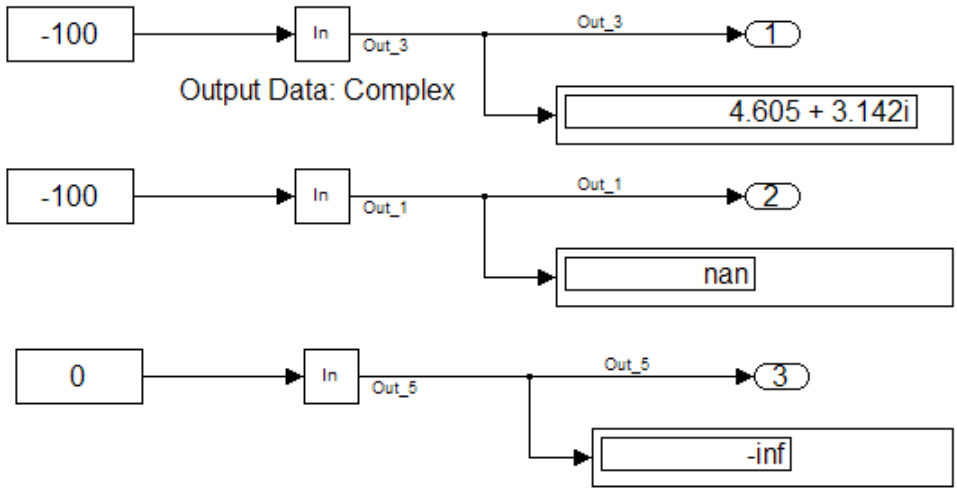
- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.4.2.2a 'Robustness Test Cases'

Model Advisor Checks

By Task > Modeling Standards for IEC-61508 > "Check usage of Simulink constructs"

hisl_0004: Usage of Math Function Blocks (Natural Logarithm and Base 10 Logarithm)

Example

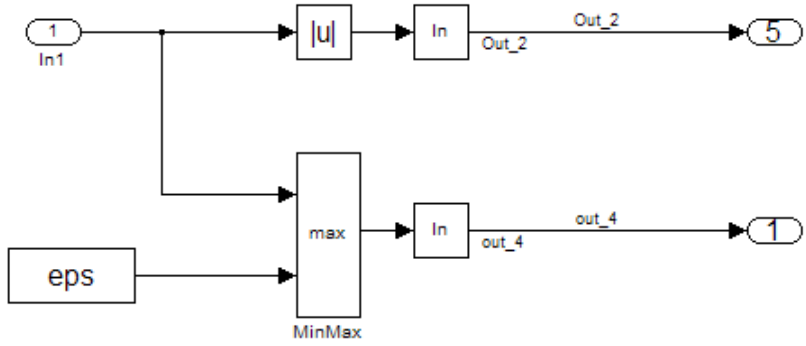


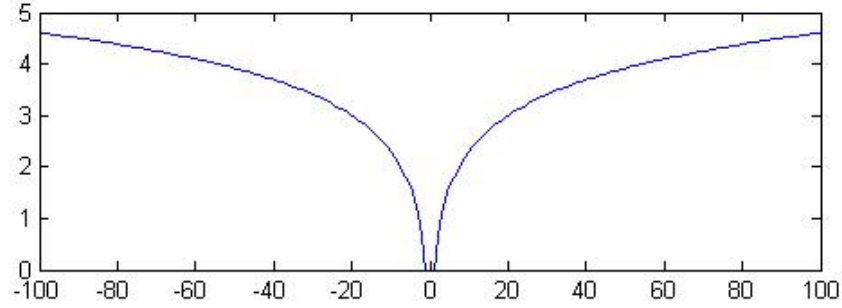
You can protect against:

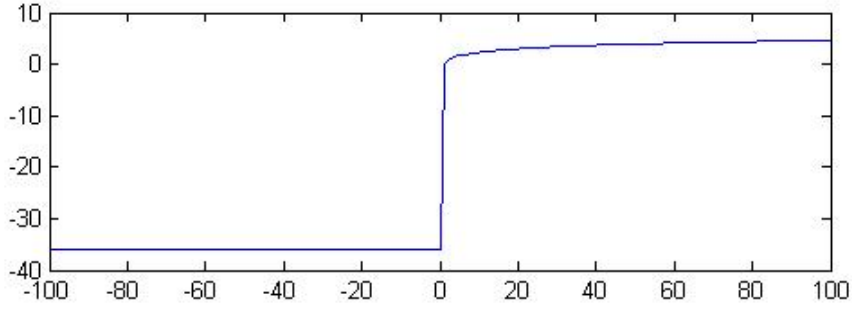
- Negative numbers using an Abs block.
- Zero values using a combination of the MinMax block and a Constant block, with **Constant value** set to `eps` (epsilon).

The following example displays the resulting output for input values ranging from `-100` to `100`.

hisl_0004: Usage of Math Function Blocks (Natural Logarithm and Base 10 Logarithm)







hisl_0005: Usage of Product Blocks

ID: Title hisl_0005: Usage of Product blocks

Priority Strongly recommended

Prerequisites Not applicable

Description To support the robustness of the generated code, when using Product blocks with divisor inputs:

- In `Element-wise(.*)` mode, protect all divisor inputs from going to zero.
- In `Matrix(*)` mode, protect all divisor inputs from becoming singular input matrices.
- In the Configuration Parameters dialog box, set **Diagnostics > Data Validity > Signals > Division by singular matrix** to `error`.

Note When using Product blocks for element-wise divisions, you might get a divide by zero, resulting in a `NaN` output. To avoid overflows, protect all divisor inputs from going to zero.

When using Product blocks to compute the inverse of a matrix, or a matrix divide, you might get a divide by a singular matrix. This division results in a `NaN` output. To avoid overflows, protect all divisor inputs from becoming singular input matrices.

During simulation, while the software inverts one of the inputs of a Product block that is in matrix multiplication mode, the **Division by singular matrix** diagnostic can detect a singular matrix.

Rationale

- Simulation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.4.2.2 'Robustness Test Cases'
  DO-178B, Section 6.4.3 'Requirements-Based Testing Methods'
- MISRA-C:2004, Rule 21.1

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related diagnostic settings for signal data"

Example

Not applicable

hisl_0005: Usage of Product Blocks

Ports & Subsystems

hisl_0006: Usage of While Iterator Blocks

hisl_0007: Usage of While Iterator Subsystems

hisl_0008: Usage of For Iterator Blocks

hisl_0009: Usage of For Iterator Subsystem Blocks

hisl_0010: Usage of If Blocks and If Action Subsystem Blocks

hisl_0011: Usage of Switch Case Blocks and Action Subsystem Blocks

hisl_0012: Usage of Triggered Subsystems

hisl_0012_b: Usage of Function-Call Subsystems

hisl_0006: Usage of While Iterator Blocks

ID: Title hisl_0006: Usage of While Iterator blocks

Priority Strongly recommended

Prerequisites Not applicable

Description To support statistically deterministic generated code when using While Iterator blocks, in the While Iterator block parameters dialog box:

- Set **Maximum number of iterations** to a positive integer value.
- Consider selecting **Show iteration number port** to observe the iteration value during simulation.

Note When you use While Iterator subsystems, The MathWorks recommends setting the maximum number of iterations. If you use an unlimited number of iterations, you might get infinite loops in the generated code, which leads to execution-time overruns.

To observe the iteration value during simulation and determine whether the loop reaches the maximum number of iterations, select **Show iteration number port** of the While Iterator block. If the loop reaches the maximum number of iterations, verify whether the output values of the While Iterator block are correct.

Rationale

- Simulation
- Code Generation
- Verification and Validation
- High Integrity Systems

Notes This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset'
  IEC 61508-3, Table A.4 (3) 'Defensive programming'

- DO-178B, Section 6.3.1e 'Review and Analyses of the High-Level Requirements: Conformance to standards'
  DO-178B, Section 6.3.2e 'Review and Analyses of the Low-Level Requirements: Conformance to standards'
- MISRA-C:2004, Rule 21.1

Model Advisor Checks

- **By Task > Modeling Standards for IEC 61508 > "Check usage of Simulink constructs"**
- **By Task > Modeling Standards for DO-178B > "Check for proper usage of While Iterator blocks"**

Example

Not applicable

hisl_0007: Usage of While Iterator Subsystems

ID: Title

hisl_0007: Usage of While Iterator subsystems

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support unambiguous behavior when you use While Iterator subsystems:

- Use inherited (`-1`) or constant (`inf`) sample times for all blocks within the subsystems.
- Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems.

Rationale

- Simulation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset'
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.4.3c 'Requirements-Based Testing Methods: Requirements-Based Low-Level Testing'
- MISRA-C:2004, Rule 21.1

Model Advisor Checks

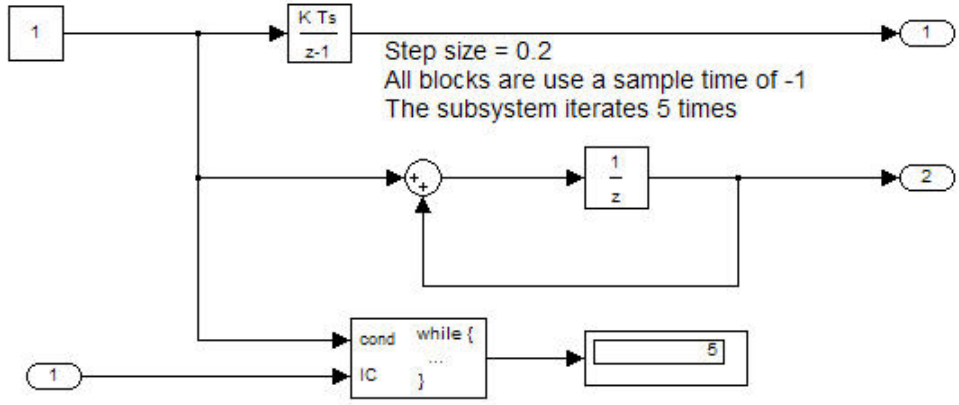
- **By Task > Modeling Standards for IEC-61508 > "Check usage of Simulink constructs"**
- **By Task > Modeling Standards for DO-178B > "Check for proper usage of While Iterator blocks"**

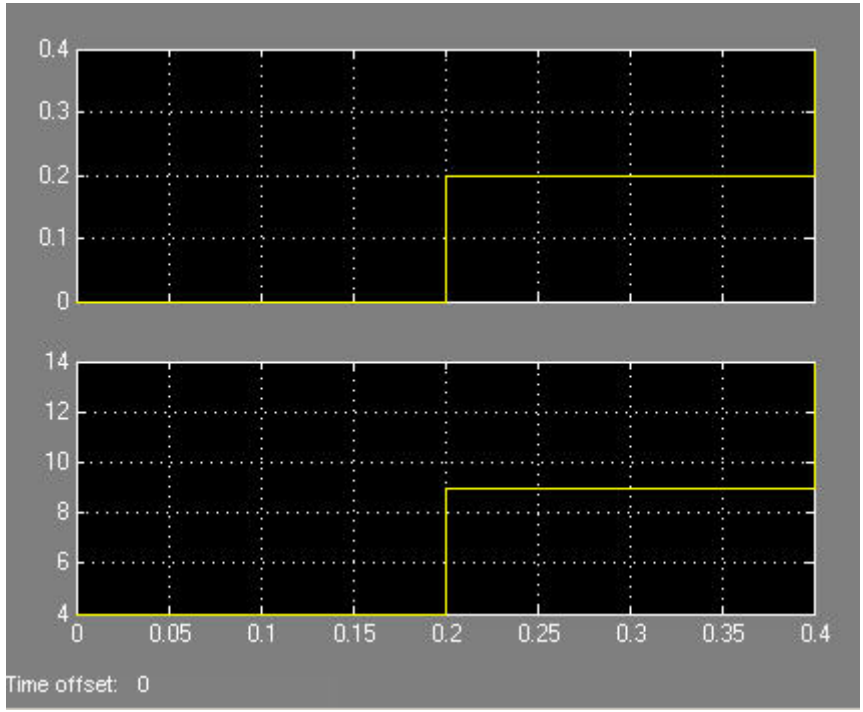
hisl_0007: Usage of While Iterator Subsystems

Example

For iterative subsystems, the value `delta T` is nonzero for the first iteration only. For subsequent iterations, the value is zero.

In the following example, in the output of the Sum block calculation that uses the unit delay, the Sum block calculation does not require `delta T`. The output of the Discrete-Time Integrator block displays the effect of the zero `delta T` value.




0.4
0.3
0.2
0.1
0
14
12
10
8
6
4
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
Time offset: 0

hisl_0008: Usage of For Iterator Blocks

ID: Title hisl_0008: Usage of For Iterator blocks

Priority Strongly recommended

Prerequisites Not applicable

Description To support statistically deterministic generated code, when using For Iterator blocks, do one of the following:

- In the For Iterator block parameters dialog box, set **Iteration limit source** to `internal`.
- If **Iteration limit source** must be `external`, use a block that has a constant value, such as a Width, Probe, or Constant block.
- In the For Iterator block parameters dialog box, clear **Set next i (iteration variable) externally**.
- In the For Iterator block parameters dialog box, consider selecting **Show iteration variable** to observe the iteration value during simulation.

Note When you use the For Iterator block, you might get a variable or unlimited number of iterations. This results in unpredictable execution times and, in the case of external iteration variables, infinite loops in the generated code, leading to execution-time overruns. Avoid these issues by feeding the loop control variable with fixed (nonvariable) values.

Rationale

- Simulation
- Verification and Validation
- Code Generation
- High Integrity Systems

Notes This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.3.1e 'Review and Analyses of the High-Level Requirements: Conformance to standards'
  DO-178B, Section 6.3.2e 'Review and Analyses of the Low-Level Requirements: Conformance to standards'
- MISRA-C:2004, Rule 13.6

Model Advisor Checks

- **By Task > Modeling Standards for IEC 61508 > "Check usage of Simulink constructs"**
- **By Task > Modeling Standards for DO-178B > "Check for proper usage of For Iterator blocks"**

Example

Not applicable

hisl_0009: Usage of For Iterator Subsystem Blocks

ID: Title

hisl_0009: Usage of For Iterator Subsystem blocks

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support unambiguous behavior, when using For Iterator Subsystem blocks:

- Use inherited (`-1`) or constant (`inf`) sample times for all the blocks within the subsystems.
- Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems.

Rationale

- Code Generation
- Safety-related Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.4.2.2d 'Robustness Test Cases: For for loops'
- MISRA-C:2004, Rule 13.6

Model Advisor Checks

- **By Task > Modeling Standards for IEC-61508 > "Check usage of Simulink constructs"**
- **By Task > Modeling Standards for DO-178B > "Check for proper usage of For Iterator blocks"**

Example

See the "Example" on page 2-18 in `hisl_0007: Usage of While Iterator Subsystems`.

hisl_0010: Usage of If Blocks and If Action Subsystem Blocks

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

Priority Strongly recommended

Prerequisites `hisl_0016: Usage of Blocks That Compute Relational Operators`

Description To support verifiable generated code, when using If blocks with nonempty `Elseif` expressions:

- In the block dialog box, select **Show else condition**.
- Connect the outports of the If block to an If Action Subsystem block.

Note 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

- Verification and Validation
- Code Generation
- Safety-related Systems

Notes This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- MISRA-C:2004, Rule 14.10

See Also:

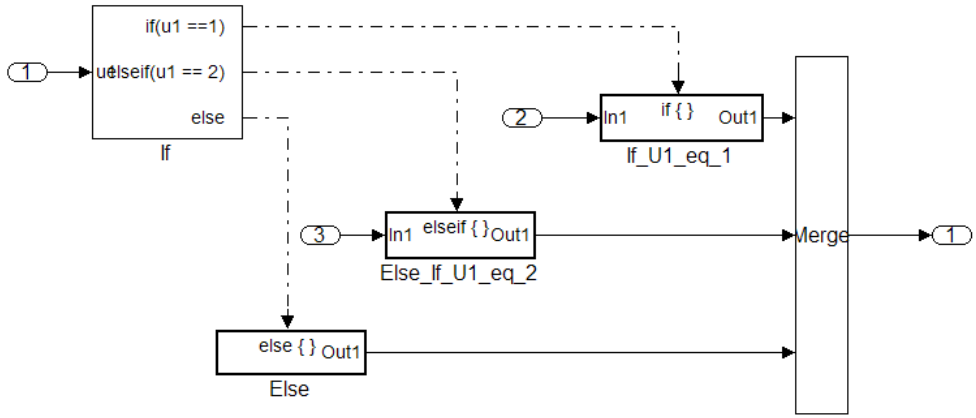
- na_0012: Use of Switch vs. If-Then-Else Action Subsystem in the Simulink® Verification and Validation™ documentation.

hisl_0010: Usage of If Blocks and If Action Subsystem Blocks

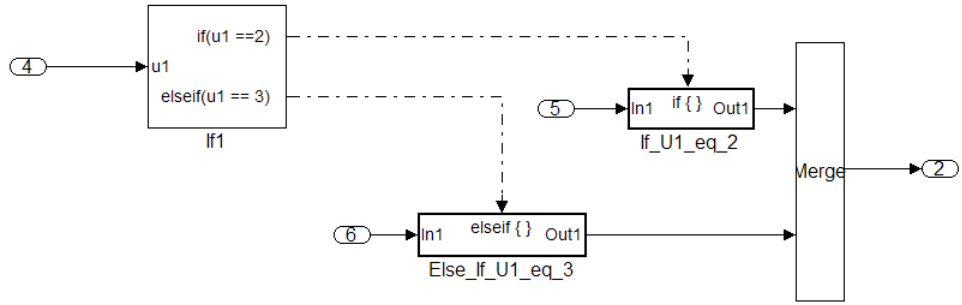
Model Advisor Checks

Not applicable

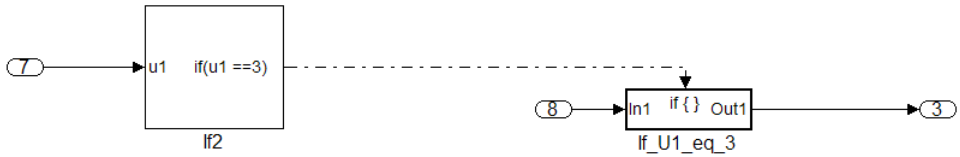
Example



Correct: Elseif with Else



Incorrect: No Else Path



Correct: 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

Priority

Strongly recommended

Prerequisites

`hisl_0016: Usage of Blocks That Compute Relational Operators`

Description

To support verifiable generated code, when using Switch Case blocks:

- In the Switch Case block dialog box, select **Show default case**.
- Connect the outports of the Switch Case block to an If Action Subsystem block.
- Use an integer data type for the inputs to Switch Case blocks.

Note 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. For an example of a "Default" block, see the "Example" on page 2-27.

Rationale

- Verification and Validation
- Code Generation
- Safety-related Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- MISRA-C:2004, Rule 15.3

See Also:

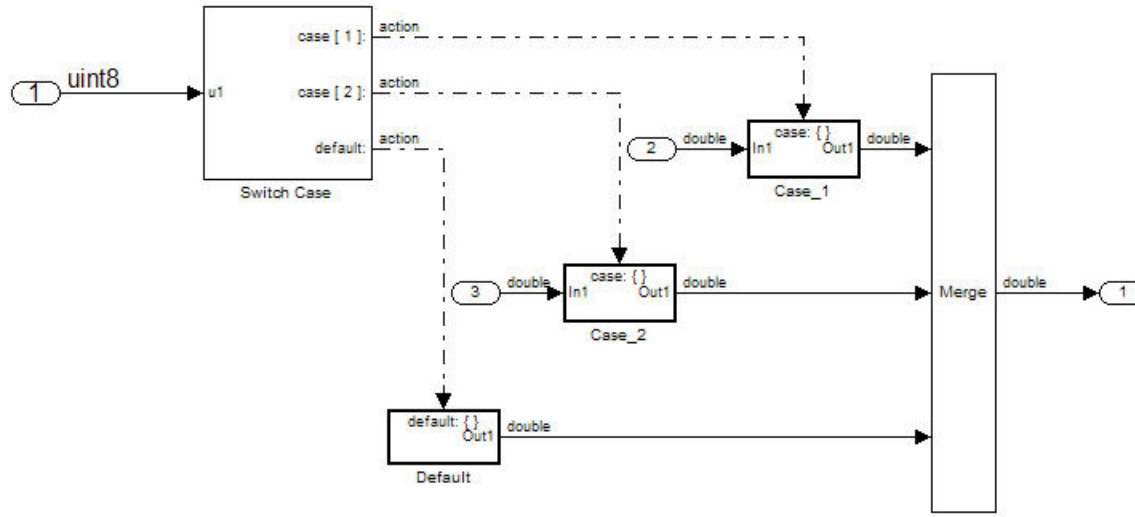
- db_0115: Simulink patterns for case constructs in the Simulink Verification and Validation documentation.

Model Advisor Checks

Not applicable

Example

The following graphic displays an example of providing a default path of execution using a “Default” block.



hisl_0012: Usage of Triggered Subsystems

ID: Title

hisl_0012: Usage of triggered subsystems

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support unambiguous behavior, when using triggered subsystems:

- Use inherited (`-1`) sample times for all blocks, except Constant blocks, within the systems. Constant blocks may use infinite (`inf`) sample time.
- Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems.

Rationale

- Simulation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'

Model Advisor Checks

Not applicable

Example

Not applicable

hisl_0012_b: Usage of Function-Call Subsystems

ID: Title

hisl_0012_b: Usage of function-call subsystems

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support unambiguous behavior, when using function-call subsystems:

- Use inherited (`-1`) sample times for all blocks, except Constant blocks, within the systems. Constant blocks may use infinite (`inf`) sample time.
- Avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions, within the subsystems.

Rationale

- Simulation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'

Model Advisor Checks

Not applicable

Example

Not applicable

Signal Routing

hisl_0013: Usage of Data Store Blocks

hisl_0015: Usage of Merge Blocks

hisl_0013: Usage of Data Store Blocks

ID: Title hisl_0013: Usage of data store blocks

Priority Strongly recommended

Prerequisites Not applicable

Description To support statistically deterministic behavior across different sample times or models, when using data store blocks, including Data Store Memory, Data Store Read, and Data Store Write blocks:

- In the Configuration Parameters dialog box, on the **Diagnostics > Data Validity** pane, set the following diagnostics in the **Data Store Memory Block** box to `error`:
  - **Detect read before write**
  - **Detect write after read**
  - **Detect write after write**
  - **Multitask data store**
  - **Duplicate data store names**
- Avoid data store reads and writes that occur across model and atomic subsystem boundaries. The sorting algorithm in Simulink does not take into account data coupling between models and atomic subsystems.
- Avoid using data stores to write and read data at different rates, because different rates can result in inconsistent exchanges of data. To provide deterministic data coupling in multirate systems, use Rate Transition blocks before Data Store Write blocks, or after Data Store Read blocks.

Note Using data store blocks can have significant effects on your software verification effort. Models and subsystems that use only inports and outports to pass data are clean, deterministic, and verifiable interfaces in the generated code.

---

hisl_0013: Usage of Data Store Blocks

Rationale

- Verification and Validation
- Code Generation
- Safety-related Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset'
- IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.3.3b 'Review and Analyses of the Software Architecture: Consistency'

Model Advisor Checks

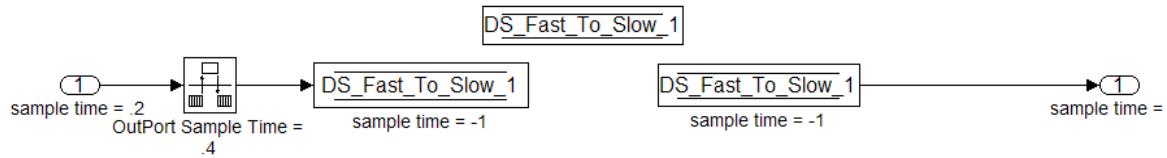
By Task > Modeling Standards for DO-178B > "Check safety-related diagnostic settings for data store memory"

Example

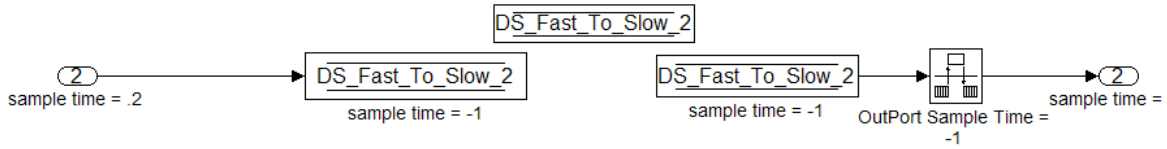
To provide deterministic data coupling in multirate systems, use Rate Transition blocks before Data Store Write blocks, or after Data Store Read blocks.

- For fast-to-slow transitions:

  Set the rate of the slow sample time on either the Rate Transition block or the Data Store Write block.

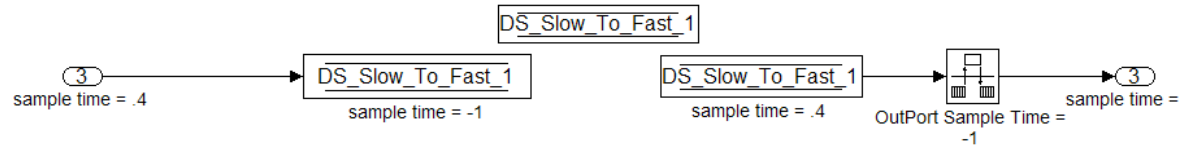


  Do not place the Rate Transition block after the Data Store Read block.



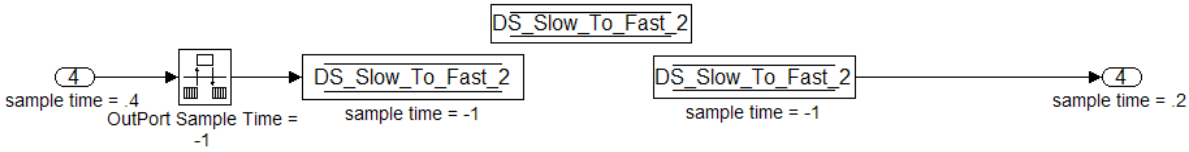
- For slow-to-fast transitions:

  If the Rate Transition block is after the Data Store Read block, specify the slow rate on the Data Store Read block.



  If the Rate Transition block is before the Data Store Write block, use the inherited sample time for all blocks.

hisl_0013: Usage of Data Store Blocks



ID: Title

hisl_0015: Usage of Merge blocks

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support unambiguous behavior from Merge blocks:

- Use Merge blocks only with conditionally executed subsystems.
- Specify the execution of the conditionally executed subsystems such that only one subsystem executes during a time step in all cases.
- Clear **Allow unequal port widths**.

Note Simulink combines the inputs of the Merge block into a single output. The output value at any time is equal to the most recently computed output of the blocks that drive the Merge block. Therefore, the Merge block output is dependent upon the execution order of the input computations.

To provide predictable behavior of the Merge block output, you must have mutual exclusion between the conditionally executed subsystems feeding a Merge block. If the inputs are not mutually exclusive, Simulink uses the last input port.

Rationale

- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

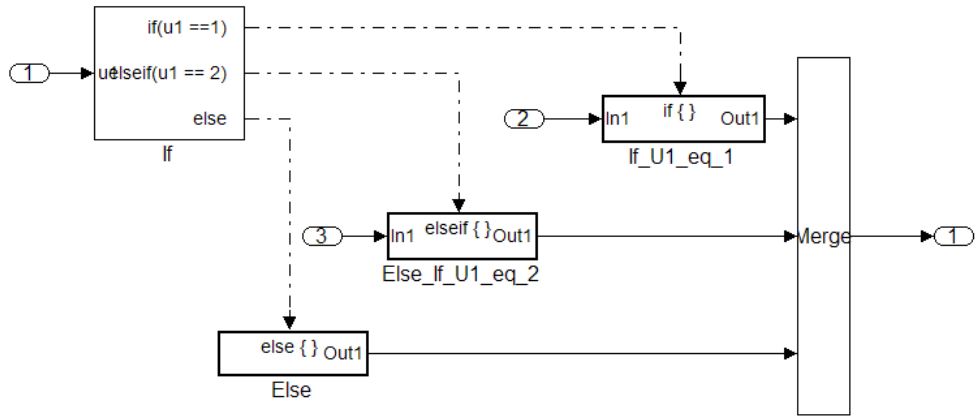
- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.4 (3) 'Defensive programming'

- DO-178B, Section 6.3.3b 'Reviews and Analyses of the Software Architecture: Consistency'

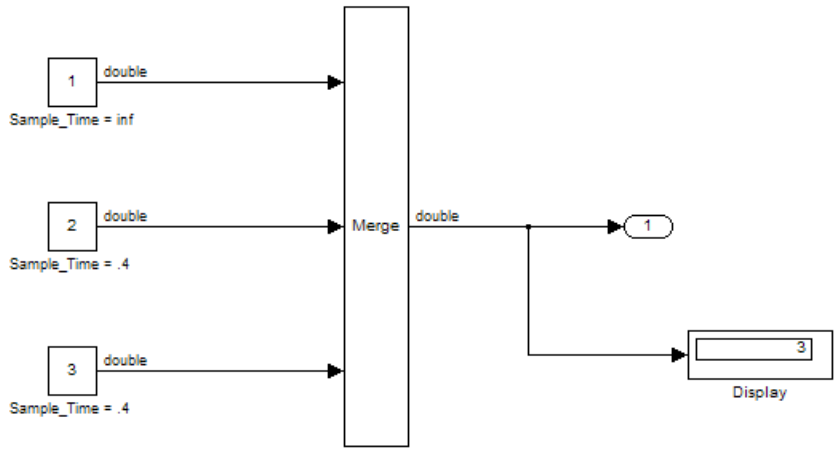
Model Advisor Checks

Not applicable

Example



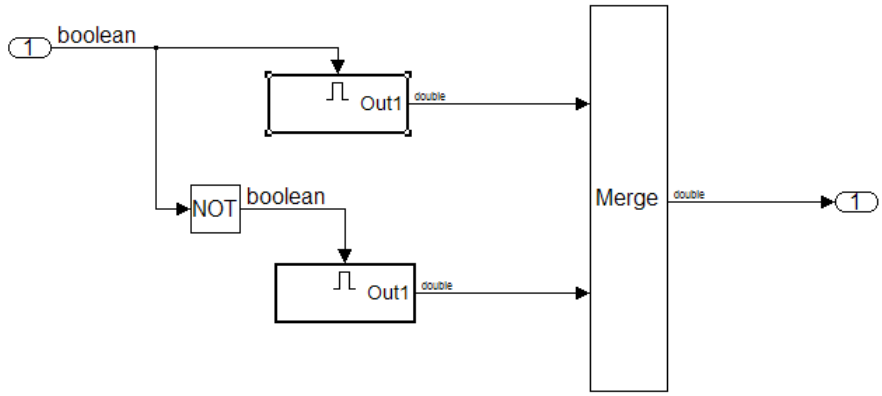
Correct



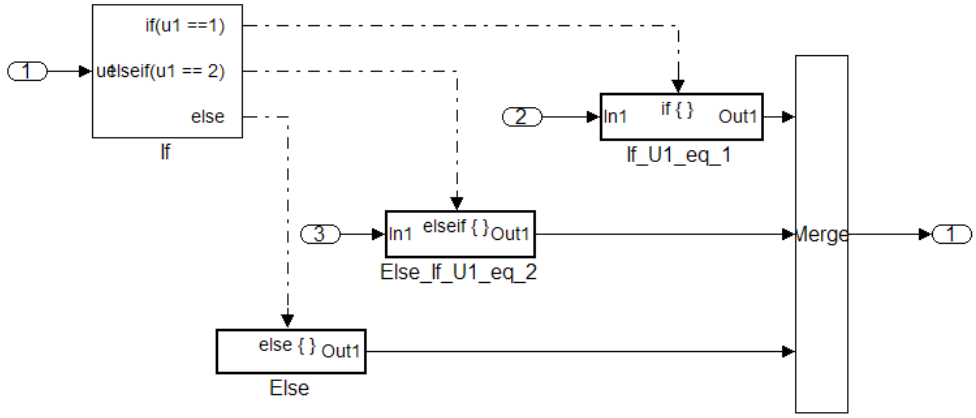
Incorrect

To ensure predictability:

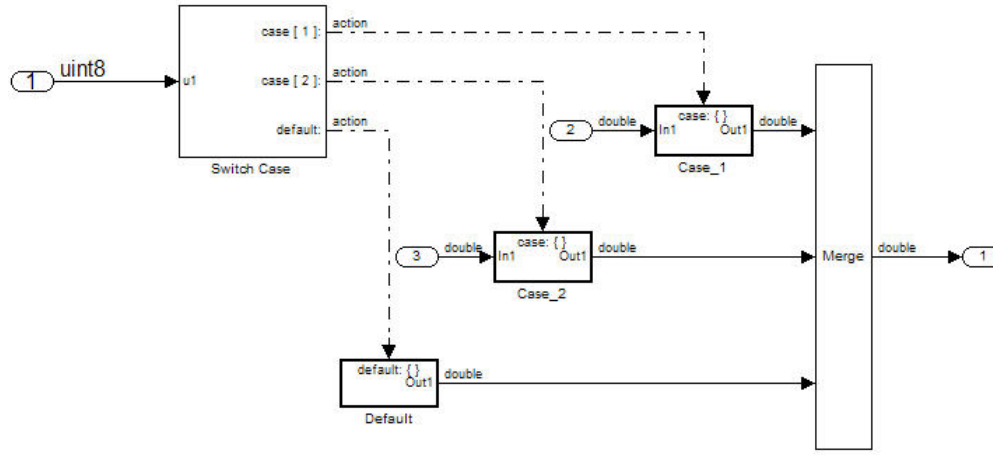
- Use Enabled Subsystem inputs with enable logic that provide exclusive execution of the subsystems.



- Enable Action Subsystem inputs from the same If-Else block that provides exclusive execution of the subsystems.



- Enable Action Subsystem inputs from the same Switch-Case block that provide exclusive execution of the subsystems.


1
uint8
u1
case [1]:
case [2]:
default:
action
action
action
Switch Case
2
double
case: { }
In1
Out1
double
Case_1
3
double
case: { }
In1
Out1
double
Case_2
default: { }
Out1
double
Default
Merge
double
1

Logic and Bit Operations

hisl_0016: Usage of Blocks That Compute Relational Operators

hisl_0017: Usage of Blocks That Compute Relational Operators (2)

hisl_0018: Usage of Logical Operator Blocks

hisl_0019: Usage of Bitwise Operator Blocks

hisl_0016: Usage of Blocks That Compute Relational Operators

ID: Title hisl_0016: Usage of blocks that compute relational operators

Priority Strongly recommended

Prerequisites Not applicable

Description To support the robustness of the operations when using blocks that compute relational operators, including Relational Operator, Compare To Constant, Compare to Zero, and Detect Change blocks:

- Avoid comparisons using the == or ~= operators on floating-point data types.

Note Due to floating-point precision issues, do not test floating-point expressions for equality (==) or inequality (~=). The software might not evaluate the comparison of floating-point expressions correctly.

When the model contains a block computing a relational operator with the == or ~= operators, the inputs to the block must not be single, double, or any custom storage class that is a floating-point type. Change the data type of the input signals, or rework the model to eliminate using the == or ~= operators within blocks that compute relational operators.

Rationale

- Verification and Validation
- Code Generation
- High Integrity Systems

Notes This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset' ;
  IEC 61508-3, Table A.4 (3) 'Defensive programming'
- DO-178B, Section 6.3.1g 'Algorithms are accurate"
  DO-178B, Section 6.3.2g 'Algorithms are accurate'
- MISRA-C:2004, Rule 13.3

hisl_0016: Usage of Blocks That Compute Relational Operators

See also:

- `hisl_0017: Usage of Blocks That Compute Relational Operators (2)`

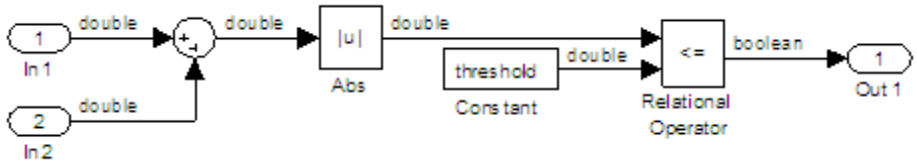
Model Advisor Checks

- **By Task > Modeling Standards for IEC 61508 > "Check usage of Simulink constructs"**
- **By Task > Modeling Standards for DO-178B > "Check for proper usage of Relational Operator blocks"**

Example

Positive Pattern: To test whether two floating-point variables or expressions are equal, compare the difference of the two variables against a threshold that takes into account the floating-point relative accuracy (`eps`) and the magnitude of the numbers.

The following pattern shows how to test two double-precision input signals, In1 and In2, for equality.



hisl_0017: Usage of Blocks That Compute Relational Operators (2)

ID: Title

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

Priority

Strongly recommended

Prerequisites

Not applicable

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 blocks:

- On the **Signal Attributes** pane of the block that computes a relational operator, set the **Output data type** to `Boolean`.

Rationale

- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.3 (2) 'Strongly typed programming language'
- MISRA-C:2004, Rule 12.6

See also:

- `hisl_0016: Usage of Blocks That Compute Relational Operators`

Model Advisor Checks

By Task > Modeling Standards for IEC 61508 > "Check usage of Simulink constructs"

Example

Not applicable

hisl_0018: Usage of Logical Operator Blocks

ID: Title

hisl_0018: Usage of Logical Operator blocks

Priority

Strongly recommended

Prerequisites

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

Description

To support unambiguous behavior in the generated code when using the Logical Operator block

- In the Logical Operator block parameters dialog box, on the **Signal Attributes** pane, set the **Output data type** to `Boolean`.

Rationale

- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.3 (2) 'Strongly typed programming language'
- MISRA-C:2004, Rule 12.6

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable

hisl_0019: Usage of Bitwise Operator Blocks

ID: Title

hisl_0019: Usage of Bitwise Operator blocks

Priority

Strongly Recommended

Prerequisites

Not applicable

Description

To support unambiguous behavior when using Bitwise Operator blocks:

- Avoid signed integer data types as inputs to the Bitwise Operator block.
- Choose an output data type that represents zero exactly.

Note Bitwise operations on signed integers are not meaningful. If a shift operation moves the sign bit into a numeric bit, or a numeric bit into the sign bit, you can see unpredictable and unwanted behavior.

Rationale

- Readability
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Table A.3 (2) 'Strongly typed programming language'
- MISRA-C:2004, Rule 12.7

Model Advisor Checks

Not applicable

Example

Not applicable

3

Configuration Parameter Considerations

Solver

hisl_0040: Configuration Parameters > Solver > Simulation time

hisl_0041: Configuration Parameters > Solver > Solver options

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

hisl_0040: Configuration Parameters > Solver > Simulation time

ID: Title

hisl_0040: Configuration Parameters > Solver > Simulation time

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support specified models, set the Configuration Parameters pertaining to the simulation time:

- On the **Configuration Parameters > Solver** pane, set **Start time** to `0.0`.
- On the **Configuration Parameters > Solver** pane, set **Stop time** to any positive value that is less than the value of **Application lifespan (days)**.

Note Simulink allows nonzero start times for simulation, however, production code generation using the Real-Time Workshop Embedded Coder product requires a zero start time.

By default, Simulink sets **Application lifespan (days)** to `inf`. If you do not change this setting, any positive value for **Stop time** is valid and this setting has no effect on generated code.

You specify **Stop time** using seconds, whereas **Application lifespan (days)** is in days.

Rationale

- Simulation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset'

hisl_0040: Configuration Parameters > Solver > Simulation time

For more information, see the Solver Pane section of the Simulink documentation

See also:

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

Model Advisor Checks

Not applicable

Example

Not applicable

hisl_0041: Configuration Parameters > Solver > Solver options

ID: Title

hisl_0041: Configuration Parameters > Solver > Solver options

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support specified models, set the configuration parameters that pertain to solver options, on the **Configuration Parameters > Solver** pane, set:

- **Type** to `Fixed-step`.
- **Solver** to `discrete (no continuous states)`.

Note Generating code for production using the Real-Time Workshop Embedded Coder product requires a fixed-step, discrete solver.

Rationale

- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset'

For more information, see "Solver Pane" in the Simulink documentation.

Model Advisor Checks

Not applicable

Example

Not applicable

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

ID: Title

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

Priority

Recommended

Prerequisites

Not applicable

Description

To support specified models, set configuration parameters pertaining to tasking and sample time options. On the **Configuration Parameters > Solver** pane:

- Set **Periodic sample time constraint** to `Specified` and assign appropriate values to **Sample time properties**.

  Caution

  If you use a referenced model as a reusable function, set **Periodic sample time constraint** to `Ensure sample time independent`.

- Set **Tasking mode for periodic sample times** to `SingleTasking` or `MultiTasking`.
- Clear **Automatically handle data transfers between tasks**.

Note Selecting the **Automatically handle data transfers between tasks** check box might result in inserting rate transition code without a corresponding model construct. This might impede establishing full traceability or showing that unintended functions are not introduced.

You can select or clear the **Higher priority value indicates higher task priority** check box . Selecting this check box determines whether the priority for **Sample time properties** uses the lowest values as highest priority, or the highest values as highest priority.

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

Rationale

- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
  IEC 61508-3, Clauses 7.4.7.2, 7.4.8.3, and 7.7.2.8 which require to demonstrate that no unintended functionality has been introduced
- DO-178B, Section 6.3.4e 'Source code is traceable to low-level requirements'

For more information, see "Solver Pane" in the Simulink documentation.

Model Advisor Checks

Not applicable

Example

Not applicable

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

Diagnostics

hisl_0043: Configuration Parameters > Diagnostics > Solver

hisl_0044: Configuration Parameters > Diagnostics > Sample Time

hisl_0043: Configuration Parameters > Diagnostics > Solver

ID: Title hisl_0043: Configuration Parameters > Diagnostics > Solver

Priority Strongly recommended

Prerequisites Not applicable

Description To support specified models, set the diagnostic settings pertaining to the solver:

- In the Configuration Parameters dialog box, on the **Diagnostics > Solver** pane, set **Algebraic loop** to `error`.
- In the Configuration Parameters dialog box, on the **Diagnostics > Solver** pane, set **Minimize algebraic loop** to `error`.
- If you are using block priorities, in the Configuration Parameters dialog box, on the **Diagnostics > Solver** pane, set **Block priority violation** to `error`.
- In the Configuration Parameters dialog box, on the **Diagnostics > Solver** pane, set **Unspecified inheritability of sample times** to `error`.
- In the Configuration Parameters dialog box, on the **Diagnostics > Solver** pane, set **Automatic solver parameter selection** to `error`.
- In the Configuration Parameters dialog box, on the **Diagnostics > Solver** pane, set **State name clash** to `warning`.

hisl_0043: Configuration Parameters > Diagnostics > Solver

Note The **Algebraic loop** diagnostic parameter detects automatic breakage of algebraic loops. The **Minimize algebraic loop** diagnostic parameter detects automatic breakage of algebraic loops for Model blocks and atomic subsystems. Breaking algebraic loops can affect the predictability of the order of block execution.

The **Block priority violation** diagnostic parameter detects potential conflicts in the block execution order that can affect the predictability of the order of block execution.

The **Unspecified inheritability of sample times** diagnostic parameter detects whether a model contains an S-function that is not explicitly set to inherit sample time. Correct these S-function parameters to prevent unpredictable behavior.

The **Automatic solver parameter selection** diagnostic parameter detects whether Simulink automatically modifies the solver, step size, or simulation stop time. Such changes can affect the operation of generated code. Explicitly set these parameters to known values.

The **State name clash** diagnostics parameter detects when you use a name for more than one state in the model. Make state names within a model unique.

Enabling the diagnostics pertaining to the solver provides information to detect violations of the previous guidelines.

In the Configuration Parameters dialog box, on the **Diagnostics > Solver** pane, you can set the following diagnostic parameters to any value:

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

hisl_0043: Configuration Parameters > Diagnostics > Solver

Rationale

- Simulation
- Code Generation
- Verification and Validation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
- DO-178B, 6.3.3e 'Software architecture conforms to standards'

For more information, see:

- "Diagnostics Pane: Solver" in the Simulink documentation.
- jc_0021: Model diagnostic settings in the Simulink Verification and Validation documentation.

Model Advisor Checks

- **By Task > Modeling Standards for DO-178B > "Check safety-related model referencing settings"**
- **By Task > Modeling Standards for DO-178B > "Check safety-related diagnostic settings for solvers"**

Example

Not applicable

hisl_0044: Configuration Parameters > Diagnostics > Sample Time

ID: Title hisl_0044: Configuration Parameters > Diagnostics > Sample Time

Priority Strongly recommended

Prerequisites Not applicable

Description To support specified models, set the diagnostic settings pertaining to the sample times. In the Configuration Parameters dialog box, on the **Diagnostics > Sample Time** pane:

- Set **Source block specifies -1 sample time** to `error`.
- Set **Discrete used as continuous** to `error`.
- Set **Multitask rate transition**to `error`.
- Set **Single task rate transition** to `error`.
- Set **Multitask conditionally executed subsystem** to `error`.
- Set **Tasks with equal priority** to `error`.
  If the target system does not allow preemption between tasks that have equal priority, set **Tasks with equal priority** to `none`.
- Set **Enforce sample times specified by Signal Specification blocks** to `error`.

Note The **Source block specifies -1 sample time** diagnostic detects when a source block, such as a Sine Wave block, inherits a sample time (specified as -1). Using inherited sample times for a source block can result in unpredictable execution rates for the source and downstream blocks. To prevent incorrect execution sequencing, explicitly specify the sample times of source blocks.

The **Discrete used as continuous** diagnostic detects whether the input for a discrete block, such as the Unit Delay block, is a continuous signal. Do not use signals with continuous sample times for embedded real-time software applications.

The **Multitask rate transition** diagnostic detects invalid rate transitions between two blocks operating in multitasking mode. Do not use invalid rate transitions for embedded real-time software applications.

The **Single task rate transition** diagnostic detects rate transition between two blocks operating in single tasking mode. If you intend to convert to a multitasking model, do not use single tasking rate transitions for embedded real-time software applications.

The **Multitask conditionally executed subsystems** diagnostic detects whether conditionally executed multirate subsystems operate in multitasking mode. These subsystems can corrupt data or show nondeterministic behavior in target systems that allow preemption.

The **Tasks with equal priority** diagnostic detects whether two asynchronous tasks have equal priority. If the real-time environment does not allow preemption between tasks that have equal priority, equal priority is acceptable. However, such tasks can show nondeterministic behavior in target systems that allow preemption.

The **Enforce sample times specified by Signal Specification blocks** diagnostic checks sample time consistency between a Signal Specification block and the connected destination block. The diagnostic reports an overspecified sample time. Overspecified sample times can result in unpredictable execution rates.

Enabling the diagnostics pertaining to the sample times provides information to detect violations of the previous guidelines.

hisl_0044: Configuration Parameters > Diagnostics > Sample Time

Rationale

- Simulation
- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language subset';
- DO-178B, Section 6.3.1b; 'High-level requirements are accurate and consistent'
  DO-178B, Section 6.3.2b; 'Low-level requirements are accurate and consistent'
  DO-178B, Section 6.3.3b;' Software architecture is consistent'

For more information, see "Diagnostics Pane: Sample Time" in the Simulink documentation.

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related diagnostic settings for sample time"

Example

Not applicable

Optimizations

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

hisl_0046: Configuration Parameters > Optimization > Block reduction

hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution

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

hisl_0051: Configuration Parameters > Optimization > Loop unrolling threshold

hisl_0052: Configuration Parameters > Optimization > Data Initialization

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

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

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

ID: Title

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

Priority

Strongly recommended

Prerequisites

Not applicable

Description

To support unambiguous behavior when using logical operators, relational operators, and Combinatorial Logic blocks:

- In the Configuration Parameters dialog box, on the **Optimization** pane, select **Implement logic signals as Boolean data (vs. double)**.

Note Selecting this check box 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

- Simulation
- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (2) 'Strongly typed programming language'
- DO-178B, 6.3.1e: High-level requirements conform to standards
  DO-178B, 6,3,2e: Low-level requirements conform to standards
- MISRA-C:2004, Rule 12.6

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

For more information, see "Implement logic signals as Boolean data (vs. double)" in the Simulink documentation.

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable

hisl_0046: Configuration Parameters > Optimization > Block reduction

ID: Title

hisl_0046: Configuration Parameters > Optimization > Block reduction

Priority

Recommended

Prerequisites

Not applicable

Description

To support unambiguous presentation of the generated code, and to support traceability between the model and generated code:

- In the Configuration Parameters dialog box, on the **Optimization** pane, consider clearing **Block reduction**.

Note Selecting **Block reduction** might optimize blocks out of the code. This results in requirements with no associated code and violates traceability objectives.

Rationale

- Readability
- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Clauses 7.4.7.2, 7.4.8.3, and 7.7.2.8 which require to demonstrate that no unintended functionality has been introduced
- DO-178B, Section 6.3.4e: 'Source code is traceable to low-level requirements'

For more information, see "Block reduction" in the Simulink documentation.

hisl_0046: Configuration Parameters > Optimization > Block reduction

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable

hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution

ID: Title

hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution

Priority

Recommended

Prerequisites

Not applicable

Description

To facilitate structural testing:

- In the Configuration Parameters dialog box, on the **Optimization** pane, consider clearing **Conditional input branch execution**.

Note The Model Coverage tool in the Simulink Verification and Validation product does not account for this optimization. This optimization can result in reporting 100% coverage, but for the same test cases, code coverage might be less than 100%.

Rationale

- Simulation
- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.4 (6) 'Structure-based testing'
- DO-178B, Section 6.4.4.2: Structural Coverage Analysis: Test coverage of software structure is achieved

For more information, see "Conditional input branch execution" in the Simulink documentation.

hisl_0047: Configuration Parameters > Optimization > Conditional input branch execution

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable

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

ID: Title

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

Priority

Strongly Recommended

Prerequisites

Not applicable

Description

To support the robustness of the behavior of systems that are continuously running:

- In the Configuration Parameters dialog box, on the **Optimization** pane, set **Application lifespan (days)** to `inf`.

Note Embedded applications may be running continuously. Do not assume a limited lifespan for Timers and counters. Setting **Application lifespan (days)** to `inf` guarantees that the simulation time is always less than the application lifespan.

Rationale

- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.4 (3) 'Defensive Programming'
- DO-178B, Section 6.3.1g 'Algorithms are accurate'
  DO-178B, Section 6.3.2g 'Algorithms are accurate'

For more information, see "Application lifespan (days)" in the Simulink documentation.

See also:

- `hisl_0040: Configuration Parameters > Solver > Simulation time`

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable

hisl_0051: Configuration Parameters > Optimization > Loop unrolling threshold

ID: Title

hisl_0051: Configuration Parameters > Optimization > Loop unrolling threshold

Priority

Strongly Recommended

Prerequisites

Not applicable

Description

To support unambiguous code, set the minimum signal or parameter width for generating a for loop.

- In the Configuration Parameters dialog box, on the **Optimization** pane, set **Loop unrolling threshold** to a value of `2` or greater.

Note The **Loop unrolling threshold** parameter specifies the array size at which the code generator begins to use a for loop, instead of separate assignment statements, to assign values to the elements of a signal or parameter array. The default value is `5`.

Rationale

- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language Subset'

For more information, see "Loop unrolling threshold" in the Simulink documentation.

Model Advisor Checks

Not applicable

Example Not applicable

hisl_0052: Configuration Parameters > Optimization > Data Initialization

ID: Title

hisl_0052: Configuration Parameters > Optimization > Data initialization

Priority

Recommended

Prerequisites

Not applicable

Description

To support complete definition of data and to ensure that all internal and external data is initialized to zero, in the Configuration Parameters dialog box, in the **Optimization > Data initialization** box:

- Consider clearing **Remove root level I/O zero initialization**.
- Consider clearing **Remove internal state zero initialization**.

Note For safety-critical software, explicitly initialize all variables. If the run-time environment of the target system provides mechanisms to initialize all I/O and state variables, consider using the initialization of the target as an alternative to the suggested settings.

Rationale

- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.4 (3) 'Defensive Programming'
- MISRA-C:2004, Rule 9.1

For more information, see "Remove root level I/O zero initialization" and "Remove internal data zero initialization" in the Simulink documentation.

hisl_0052: Configuration Parameters > Optimization > Data Initialization

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable

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

ID: Title

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

Priority

Recommended

Prerequisites

Not applicable

Description

To support verifiable code:

- In the Configuration Parameters dialog box, in the **Optimization > Integer and fixed-point section** box , consider selecting **Remove code from floating-point to integer conversions that wraps out-of-range values**.

Note For safety-critical software, avoid overflows as opposed to handling them with special wrapping code. For blocks that have cleared **Saturate on overflow**, clearing **Remove code from floating-point to integer conversions that wraps out-of-range values** might add code that wraps out of range values. This code results in unreachable, for example, untestable, code.

Rationale

- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.4 (3) 'Defensive Programming'
- MISRA-C:2004, Rule 14.1

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

For more information, see "Remove code from floating-point to integer conversions that wraps out-of-range values" in the Simulink documentation.

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable

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

ID: Title

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

Priority

Strongly Recommended

Prerequisites

Not applicable

Description

To support the robustness of the operations:

- In the Configuration Parameters dialog box, in the **Optimization > Integer and fixed-point** box , clear **Remove code that protects against division arithmetic exceptions**.

Note For safety-critical software, avoid division-by-zero exceptions. When you clear **Remove code that protects against division arithmetic exceptions**, the Real-Time Workshop Embedded Coder software generates code that guards against division by zero for fixed-point data.

Rationale

- Verification and Validation
- Code Generation
- High Integrity Systems

Notes

This guideline supports adhering to:

- IEC 61508-3, Table A.3 (3) 'Language Subset';
  IEC 61508-3 Table A.4 (3) 'Defensive Programming'
- MISRA-C:2004, Rule 21.1

For more information, see "Remove code that protects against division arithmetic exceptions" in the Simulink documentation.

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

Model Advisor Checks

By Task > Modeling Standards for DO-178B > "Check safety-related optimization settings"

Example

Not applicable