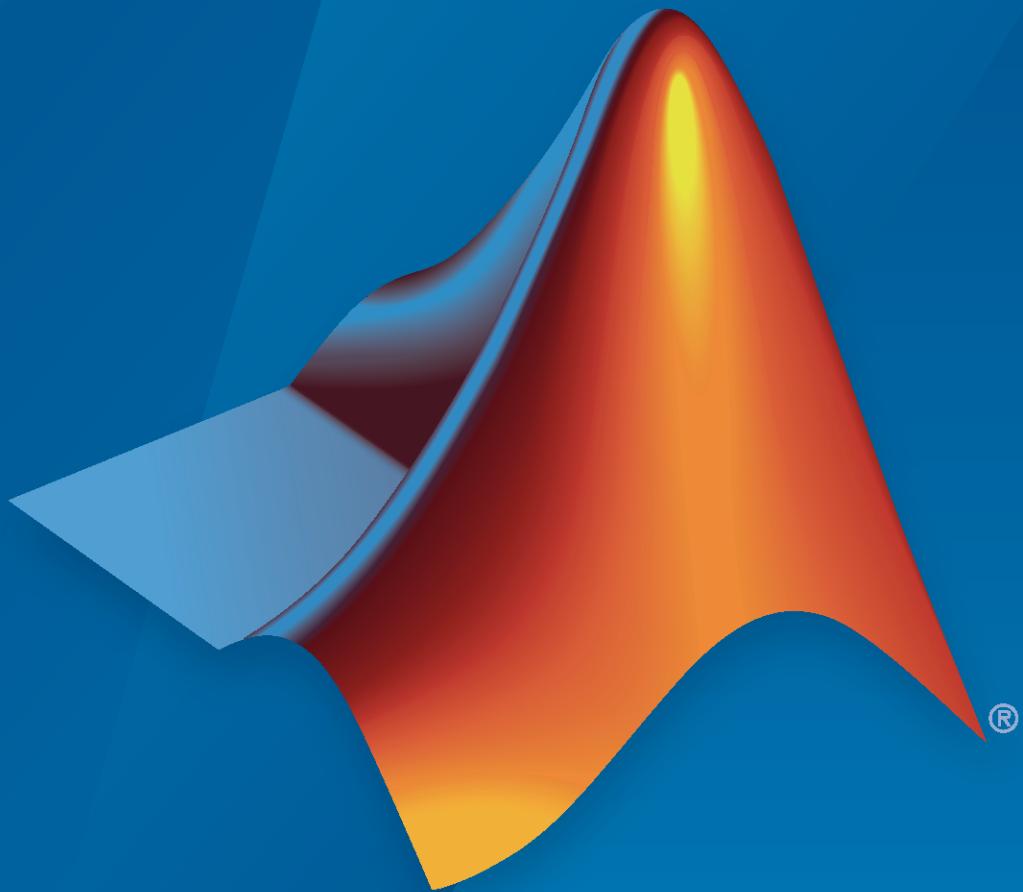


Simulink®

MathWorks® Advisory Board Control Algorithm
Modeling Guidelines



MATLAB®&SIMULINK®

R2024b

 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

MathWorks® Advisory Board Modeling Guidelines

© COPYRIGHT 2007-2024 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

March 2009	Online only	New for Version 2.0 (Release 2009a)
September 2009	Online only	Revised for Version 2.1 (Release 2009b)
March 2010	Online only	Rereleased for Version 2.1 (Release 2010a)
September 2010	Online only	Rereleased for Version 2.1 (Release 2010b)
April 2011	Online only	Rereleased for Version 2.1 (Release 2011a)
September 2011	Online only	Rereleased for Version 2.1 (Release 2011b)
March 2012	Online only	Rereleased for Version 2.2 (Release 2012a)
September 2012	Online only	Rereleased for Version 2.2 (Release 2012b)
March 2013	Online only	Revised for Version 3.0 (Release 2013a)
September 2013	Online only	Rereleased for Version 3.0 (Release 2013b)
March 2014	Online only	Revised for Version 3.1 (Release 2014a)
October 2014	Online only	Rereleased for Version 3.1 (Release 2014b)
March 2015	Online only	Rereleased for Version 3.1 (Release 2015a)
September 2015	Online only	Rereleased for Version 3.1 (Release 2015b)
March 2016	Online only	Rereleased for Version 3.1 (Release 2016a)
September 2016	Online only	Rereleased for Version 3.1 (Release 2016b)
March 2017	Online only	Rereleased for Version 3.1 (Release 2017a)
September 2017	Online only	Rereleased for Version 3.1 (Release 2017b)
March 2018	Online only	Rereleased for Version 3.1 (Release 2018a)
September 2018	Online only	Rereleased for Version 3.1 (Release 2018b)
March 2019	Online only	Rereleased for Version 3.1 (Release 2019a)
September 2019	Online only	Rereleased for Version 3.2 (Release 2019b)
March 2020	Online only	MAAB guidelines revised and reintroduced as the MathWorks Advisory Board (MAB) Modeling Guidelines, Version 5.0 (Release 2020a)
September 2020	Online only	Rereleased for Version 5.0 (Release 2020b)
March 2021	Online only	Rereleased for Version 5.0 (Release 2021a)
September 2021	Online only	Rereleased for Version 5.0 (Release 2021b)
March 2022	Online only	Revised for Version 5.0 (Release 2022a)
September 2022	Online only	Revised for Version 5.0 (Release 2022b)
March 2023	Online only	Revised for Version 5.0 (Release 2023a)
September 2023	Online only	Revised for Version 23.2 (R2023b)
March 2024	Online only	Revised for Version 24.1 (R2024a)
September 2024	Online only	Revised for Version 24.2 (R2024b)

Introduction

1

Purpose of the Guidelines and Template	1-2
Model Advisor Checks for MAB Modeling Guidelines	1-6
Model Advisor Checks for JMAAB Modeling Guidelines	1-17
See Also	1-32

Naming Conventions

2

General Conventions	2-2
Content Conventions	2-11

Simulink

3

Configuration Parameters	3-2
Diagram Appearance	3-12
Signal	3-68
Conditional Subsystem Relations	3-103
Operation Blocks	3-125
Other Blocks	3-174

Stateflow

4

Stateflow Blocks / Data / Events	4-2
---	------------

Stateflow Diagram	4-29
Conditional Transition / Action	4-82
Label Description	4-149
Miscellaneous	4-176

MATLAB

5

MATLAB Appearance	5-2
MATLAB Data and Operations	5-7
MATLAB Usage	5-15

Considerations

6

Considerations for Determining Guideline Operation Rules	6-2
Process Definition and Development Environment	6-2
MATLAB and Simulink Versions	6-2
MATLAB and Simulink Settings	6-2
Usable Blocks	6-3
Using Optimization and Configuration Parameters	6-3
Considerations for Applying Guidelines to a Project	6-5
Using the Model Analysis Process When Applying Guidelines	6-5
Adoption of the Guideline Rule and Process Settings	6-5
Setting the Guideline Rule Application Field and the Clarifying the Exclusion Condition	6-5
Parameter Recommendations in the Guidelines	6-6
Verifying Adherence to the Guidelines	6-6
Modifying Adherence to the Guidelines	6-6

Using Simulink and Stateflow

A

Understanding Model Architecture	A-2
Hierarchical Structure of a Controller Model	A-2
Relationship Between Simulink Models and Embedded Implementation	A-9
Using Simulink and Stateflow in Modeling	A-16
Simulink Functionality	A-17

Stateflow Functionality	A-24
Initialization	A-31

Modeling Knowledge

B

Usage Patterns	B-2
Simulink Patterns for If, elseif, else Constructs	B-2
Simulink Patterns for case Constructs	B-2
Simulink Patterns for Logical Constructs	B-3
Simulink Patterns for Vector Signals	B-4
Using Switch and If, Elseif, Else Action Subsystems	B-6
Use of If, Elseif, Else Action Subsystem to Replace Multiple Switches	B-8
Usage Rules for Action Subsystems Using Conditional Control Flow	B-12
Test for Information From Errors	B-16
Flow Chart Patterns for Conditions	B-17
Flow Chart Patterns for Condition Actions	B-18
Flow Chart Patterns for If, Elseif, Else Constructs	B-19
Flow Chart Patterns for Case Constructs	B-20
Flow Chart Patterns for Loop Constructs	B-21
State Machine Patterns for Conditions	B-22
State Machine Patterns for Transition Actions	B-22
jc_0321Limiting State Layering	B-23
Number of States per Stateflow Container	B-24
Function Call from Stateflow	B-24
Function Types Available in Stateflow	B-24

Glossary

C

Introduction

- “Purpose of the Guidelines and Template” on page 1-2
- “Model Advisor Checks for MAB Modeling Guidelines” on page 1-6
- “Model Advisor Checks for JMAAB Modeling Guidelines” on page 1-17

Purpose of the Guidelines and Template

MathWorks Advisory Board (MAB) guidelines stipulate important basic rules for modeling in Simulink and Stateflow. The overall purpose of these modeling guidelines is to allow for a simple, common understanding by modelers and consumers of control system models.

The main objectives of these guidelines are:

- Readability
 - Improve graphical understandability
 - Improve readability of functional analysis
 - Prevent connection mistakes
 - Comments, etc.
- Simulation and verification
 - Mechanism to enable simulation
 - Testability
- Code Generation
 - Improve the efficiency of code generation (ROM, RAM efficiency)
 - Ensure the robustness of generated code

Note Model runtime errors and recommendations that cannot be implemented are outside of the scope of these rules.

Guidelines are documented by using a standard template. Use of this template is recommended when creating original guidelines.

Note This template specifies the minimum requirements that are needed to understand a guideline. New items can be added to the template as long as they do not duplicate existing information.

Section Heading	Format	Section Description
Rule ID: Title	<i>XX_nnnn</i> : Title of the guideline (unique, short)	A rule ID, which is used to identify the guideline, consists of two lower case letters and a four-digit number. The letter and number combination is separated by an underscore. For example, xx_nnnn . A rule ID is permanent and will not change. Note The two-letters in the rule ID identify the guideline author.

Section Heading	Format	Section Description
Sub ID Recommendations	NA-MAAB: x, y, z JMAAB: x, y, z	Specifies guideline sub IDs that are recommended for use by the NA-MAAB (North American MathWorks Automotive Advisory Board) and JMAAB (Japan MathWorks Automotive Advisory Board) modeling standards organizations. Each organization is a region-specific consortium of automotive OEMs and suppliers; NA-MAAB represents North America and Europe. JMAAB represents Japan.
MATLAB Versions	All RX, RY, RZ RX and earlier RX and later RX through RY	<p>MAB guidelines support all versions of MATLAB and Simulink products. When a rule applies only to a specific version(s), the version is identified in the MATLAB Version field by using one of these formats:</p> <ul style="list-style-type: none"> • All — All versions of MATLAB • RX, RY, RZ — A specific version of MATLAB • RX and earlier — Versions of MATLAB until version RX • RX and later — Versions of MATLAB from version RX to the current version • RX through RY — Versions of MATLAB between RX and RY

Section Heading	Format	Section Description
Rule > Sub ID	<p>Specifies the condition(s) of the rule by using Sub IDs. There can be multiple sub IDs per rule ID.</p> <p>The subsections for a Sub ID include:</p> <ul style="list-style-type: none"> • Custom Parameter • (Optional) Exclusion • (Optional) Example 	<p>Specifies the condition(s) of the rule. Sub IDs are designated as either:</p> <ul style="list-style-type: none"> • Selectable — Consist of one lower-case letter (alphabetical order). The choice of whether to adopt a selectable sub ID is left to the user. • Mutually Exclusive — Consist of one lower case letter (alphabetical order) and a single-digit number. When choosing to accept or reject a mutually exclusive sub ID, only one option can be selected. <p>For example, Sub IDs for guideline xy_0000 are:</p> <ul style="list-style-type: none"> • xy_0000a — Represents a selectable (user's choice) option • xy_0000b1 xy_0000b2 — Mutually Exclusive (if using, choose from xy_0000b1 or xy_0000b2) <p>For rules that include custom parameters, the chosen value is specific for the project with regard to the item being described. Example of objects and values are provided in the description field. However, a project's processes, condition of the control target, and skill levels of the engineers should be comprehensively evaluated when specifying a custom parameter.</p>

Section Heading	Format	Section Description
Rationale	Motivation for the Sub IDs	Provides reasoning for the use of the guideline with regard to readability, verification efficiency, efficiency of code after code generation, etc.
Verification	Verification methods	Method(s) that verify compliance of the model with guideline conditions, such as: <ul style="list-style-type: none"> • Model Advisor check • Manual verification method
Last Change	R<year>	MATLAB revision in which the content of the guideline was last updated. For example, R2020a.
See Also	References	Additional information that can be helpful to better understand the guideline.

Model Advisor Checks for MAB Modeling Guidelines

This table identifies the Model Advisor checks that you can use to verify compliance of your model with MathWorks Advisor Board (MAB) modeling guidelines.

In the Model Advisor, to access the MAB checks, select:

- **By Product > Simulink Check > Modeling Standards > MAB Checks**
- **By Task > Modeling Standards for MAB**

You need Simulink Check™ to execute these checks.

Note Some guidelines do not have corresponding Model Advisor checks:

- *No check* — Indicates that the guideline can be checked by using a Model Advisor check, however, the check does not currently exist.
 - *Not checkable* — Indicates that it is not possible to verify compliance to this guideline by using a Model Advisor check.
-

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
ar_0001: Usable characters for file names	“Check file names” (Simulink Check)	mathworks.jmaab.ar_0001
ar_0002: Usable characters for folder names	“Check folder names” (Simulink Check)	mathworks.jmaab_v6.ar_0002
db_0032: Signal line connections	“Check signal line connections” (Simulink Check)	mathworks.jmaab_v6.db_0032
db_0042: Usage of Inport and Outport blocks	“Check position of Inport and Outport blocks” (Simulink Check)	mathworks.jmaab.db_0042
db_0043: Model font and font size	“Check model font settings” (Simulink Check)	mathworks.jmaab.db_0043
db_0081: Unconnected signals and blocks	“Check for unconnected signal lines and blocks” (Simulink Check)	mathworks.jmaab.db_0081
db_0097: Position of labels for signals and buses	“Check position of signal labels” (Simulink Check)	mathworks.jmaab_v6.db_0097
db_0110: Block parameters	“Check usage of tunable parameters in blocks” (Simulink Check)	mathworks.maab.db_0110
db_0112: Usage of index	“Check Indexing Mode” (Simulink Check)	mathworks.jmaab.db_0112

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
db_0123: Stateflow port names	"Check for names of Stateflow ports and associated signals" (Simulink Check)	mathworks.maab.db_0123
db_0125: Stateflow local data	"Check definition of Stateflow data" (Simulink Check)	mathworks.jmaab_v6.db_0125
db_0126: Defining Stateflow events	"Check definition of Stateflow events" (Simulink Check)	mathworks.jmaab.db_0126
db_0127: Limitation on MATLAB commands in Stateflow blocks	"Check for MATLAB expressions in Stateflow charts" (Simulink Check)	mathworks.jmaab_v6.db_0127
db_0129: Stateflow transition appearance	"Check for Stateflow transition appearance" (Simulink Check)	mathworks.jmaab_v6.db_0129
db_0132: Transitions in flow charts	"Check transitions in Stateflow Flow charts" (Simulink Check)	mathworks.jmaab.db_0132
db_0137: States in state machines	"Check for state in state machines" (Simulink Check)	mathworks.jmaab.db_0137
db_0140: Display of block parameters	"Check for display of block parameter" (Simulink Check)	mathworks.maab.db_0140
db_0141: Signal flow in Simulink models	"Check signal flow in model" (Simulink Check)	mathworks.maab.db_0141
db_0142: Position of block names	"Check whether block names appear below blocks" (Simulink Check)	mathworks.maab.db_0142
db_0143: Usable block types in model hierarchy	"Check for mixing basic blocks and subsystems" (Simulink Check)	mathworks.maab.db_0143
db_0144: Use of subsystems	<i>Not checkable</i>	
db_0146: Block layout in conditional subsystems	"Check position of conditional blocks and iterator blocks" (Simulink Check)	mathworks.jmaab.db_0146
hd_0001: Prohibited Simulink sinks	"Check for prohibited sink blocks" (Simulink Check)	mathworks.maab.hd_0001
jc_0008: Definition of signal names	"Check definition of signal labels" (Simulink Check)	mathworks.jmaab.jc_0008
jc_0009: Signal name propagation	"Check signal name propagation" (Simulink Check)	mathworks.jmaab_v6.jc_0009
jc_0011: Optimization parameters for Boolean data types	"Check Implement logic signals as Boolean data (vs. double)" (Simulink Check)	mathworks.maab.jc_0011
jc_0021: Model diagnostic settings	"Check model diagnostic parameters" (Simulink Check)	mathworks.maab.jc_0021

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
jc_0061: Display of block names	"Check the display attributes of block names" (Simulink Check)	mathworks.maab.jc_0061
jc_0081: Import and Outport block icon display	"Check display for port blocks" (Simulink Check)	mathworks.maab.jc_0081
jc_0110: Direction of block	"Check block orientation" (Simulink Check)	mathworks.jmaab.jc_0110
jc_0121: Usage of add and subtraction blocks	"Check usage of Sum blocks" (Simulink Check)	mathworks.jmaab.jc_0121
jc_0131: Usage of Relational Operator blocks	"Check usage of Relational Operator blocks" (Simulink Check)	mathworks.maab.jc_0131
jc_0141: Usage of the Switch blocks	"Check usage of Switch blocks" (Simulink Check)	mathworks.maab.jc_0141
jc_0161: Definition of Data Store Memory blocks	"Check for usage of Data Store Memory blocks" (Simulink Check)	mathworks.jmaab.jc_0161
jc_0171: Clarification of connections between structural subsystems	"Check connections between structural subsystems" (Simulink Check)	mathworks.jmaab.jc_0171
jc_0201: Usable characters for subsystem names	"Check subsystem names" (Simulink Check)	mathworks.jmaab.jc_0201
jc_0211: Usable characters for Inport blocks and Outport block	"Check port block names" (Simulink Check)	mathworks.jmaab.jc_0211
jc_0222: Usable characters for signal and bus names	"Check usable characters for signal names and bus names" (Simulink Check)	mathworks.jmaab.jc_0222
jc_0231: Usable characters for block names	"Check character usage in block names" (Simulink Check)	mathworks.jmaab.jc_0231
jc_0232: Usable characters for parameter names	"Check usable characters for parameter names" (Simulink Check)	mathworks.jmaab_v6.jc_0232
jc_0241: Length restriction for model file names	"Check length of model file name" (Simulink Check)	mathworks.jmaab.jc_0241
jc_0242: Length restriction for folder names	"Check length of folder name at every level of model path" (Simulink Check)	mathworks.jmaab.jc_0242
jc_0243: Length restriction for subsystem names	"Check length of subsystem names" (Simulink Check)	mathworks.jmaab.jc_0243
jc_0244: Length restriction for Inport and Outport names	"Check length of Inport and Outport names" (Simulink Check)	mathworks.jmaab.jc_0244

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
jc_0245: Length restriction for signal and bus names	"Check length of signal and bus names" (Simulink Check)	mathworks.jmaab.jc_0245
jc_0246: Length restriction for parameter name	"Check length of parameter names" (Simulink Check)	mathworks.jmaab.jc_0246
jc_0247: Length restriction for block names	"Check length of block names" (Simulink Check)	mathworks.jmaab.jc_0247
jc_0281: Trigger signal names	"Check trigger signal names" (Simulink Check)	mathworks.jmaab.jc_0281
jc_0451: Use of unary minus on unsigned integers	"Check usage of unary minus operations in Stateflow charts" (Simulink Check)	mathworks.jmaab.jc_0451
jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	mathworks.jmaab_v6.jc_0481
jc_0491: Reuse of Stateflow data	<i>Not checkable</i>	
jc_0501: Format of entries in a State block	"Check entry formatting in State blocks in Stateflow charts" (Simulink Check)	mathworks.jmaab.jc_0501
jc_0511: Return values from a graphical function	"Check return value assignments in Stateflow graphical functions" (Simulink Check)	mathworks.maab.jc_0511
jc_0531: Default transition	"Check default transition placement in Stateflow charts" (Simulink Check)	mathworks.jmaab.jc_0531
jc_0602: Consistency in model element names	"Check for consistency in model element names" (Simulink Check)	mathworks.jmaab.jc_0602
jc_0603: Model description	"Check Model Description" (Simulink Check)	mathworks.jmaab.jc_0603
jc_0604: Using block shadow	"Check if blocks are shaded in the model" (Simulink Check)	mathworks.jmaab.jc_0604
jc_0610: Operator order for multiplication and division block	"Check operator order of Product blocks" (Simulink Check)	mathworks.jmaab.jc_0610
jc_0611: Input sign for multiplication and division blocks	"Check signs of input signals in product blocks" (Simulink Check)	mathworks.jmaab.jc_0611
jc_0621: Usage of Logical Operator blocks	"Check icon shape of Logical Operator blocks" (Simulink Check)	mathworks.jmaab.jc_0621

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
jc_0622: Usage of Fcn blocks	"Check for parentheses in Fcn block expressions" (Simulink Check)	mathworks.jmaab.jc_0622
jc_0623: Usage of continuous-time Delay blocks and discrete-time Delay blocks	"Check usage of Memory and Unit Delay blocks" (Simulink Check)	mathworks.jmaab.jc_0623
jc_0624: Usage of Tapped Delay blocks/Delay blocks	"Check for cascaded Unit Delay blocks" (Simulink Check)	mathworks.jmaab.jc_0624
jc_0626: Usage of Lookup Table blocks	"Check usage of Lookup Tables" (Simulink Check)	mathworks.jmaab.jc_0626
jc_0627: Usage of Discrete-Time Integrator blocks	"Check usage of Discrete-Time Integrator block" (Simulink Check)	mathworks.jmaab_v6.jc_0627
jc_0628: Usage of Saturation blocks	"Check usage of the Saturation blocks" (Simulink Check)	mathworks.jmaab.jc_0628
jc_0630: Usage of Multiport Switch blocks	"Check settings for data ports in Multiport Switch blocks" (Simulink Check)	mathworks.jmaab_v6.jc_0630
jc_0640: Initial value settings for Outport blocks in conditional subsystems	"Check undefined initial output for conditional subsystems" (Simulink Check)	mathworks.jmaab.jc_0640
jc_0641: Sample time setting	"Check for sample time setting" (Simulink Check)	mathworks.jmaab.jc_0641
jc_0642: Integer rounding mode setting	"Check Signed Integer Division Rounding mode" (Simulink Check)	mathworks.jmaab.jc_0642
jc_0643: Fixed-point setting	"Check usage of fixed-point data type with non-zero bias" (Simulink Check)	mathworks.jmaab.jc_0643
jc_0644: Type setting	"Check type setting by data objects" (Simulink Check)	mathworks.jmaab_v6.jc_0644
jc_0645: Parameter definition for calibration	"Check if tunable block parameters are defined as named constants" (Simulink Check)	mathworks.jmaab.jc_0645
jc_0650: Block input/output data type with switching function	"Check input and output datatype for Switch blocks" (Simulink Check)	mathworks.jmaab.jc_0650
jc_0651: Implementing a type conversion	"Check Output data type of operation blocks" (Simulink Check)	mathworks.jmaab_v6.jc_0651

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
jc_0653: Delay block layout in feedback loops	"Check for avoiding algebraic loops between subsystems" (Simulink Check)	mathworks.jmaab.jc_0653
jc_0655: Prohibition of logical value comparison in Stateflow	"Check prohibited comparison operation of logical type signals" (Simulink Check)	mathworks.jmaab.jc_0655
jc_0656: Usage of Conditional Control blocks	"Check default/else case in Switch Case blocks and If blocks" (Simulink Check)	mathworks.jmaab.jc_0656
jc_0657: Retention of output value based on conditional control flow blocks and Merge blocks	<i>Not checkable</i>	
jc_0659: Usage restrictions of signal lines input to Merge blocks	"Check usage of Merge block" (Simulink Check)	mathworks.jmaab.jc_0659
jc_0700: Unused data in Stateflow block	"Check for unused data in Stateflow Charts" (Simulink Check)	mathworks.jmaab.jc_0700
jc_0701: Usable number for first index	"Check usable number for first index" (Simulink Check)	mathworks.jmaab.jc_0701
jc_0702: Use of named Stateflow parameters and constants	"Check usage of numeric literals in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0702
jc_0711: Division in Stateflow	<i>Not checkable</i>	
jc_0712: Execution timing for default transition path	"Check execution timing for default transition path" (Simulink Check)	mathworks.jmaab.jc_0712
jc_0721: Usage of parallel states	"Check usage of parallel states" (Simulink Check)	mathworks.jmaab.jc_0721
jc_0722: Local data definition in parallel states	"Check scope of data in parallel states" (Simulink Check)	mathworks.jmaab.jc_0722
jc_0723: Prohibited direct transition from external state to child state	"Check usage of transitions to external states" (Simulink Check)	mathworks.jmaab.jc_0723
jc_0730: Unique state name in Stateflow blocks	"Check uniqueness of State names" (Simulink Check)	mathworks.jmaab.jc_0730
jc_0731: State name format	"Check usage of State names" (Simulink Check)	mathworks.jmaab.jc_0731
jc_0732: Distinction between state names, data names, and event names	"Check uniqueness of Stateflow state, data and event names" (Simulink Check)	mathworks.jmaab.jc_0732

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
jc_0733: Order of state action types	"Check order of state action types" (Simulink Check)	mathworks.jmaab.jc_0733
jc_0734: Number of state action types	"Check repetition of action types" (Simulink Check)	mathworks.jmaab.jc_0734
jc_0736: Uniform indentations in Stateflow blocks	"Check indentation of code in Stateflow states" (Simulink Check)	mathworks.jmaab.jc_0736
jc_0738: Usage of Stateflow comments	"Check usage of Stateflow comments" (Simulink Check)	mathworks.jmaab.jc_0738
jc_0740: Limitation on use of exit state action	"Check if state action type 'exit' is used in the model" (Simulink Check)	mathworks.jmaab.jc_0740
jc_0741: Timing to update data used in state chart transition conditions	"Check updates to variables used in state transition conditions" (Simulink Check)	mathworks.jmaab_v6.jc_0741
jc_0751: Backtracking prevention in state transition	"Check for unexpected backtracking in state transitions" (Simulink Check)	mathworks.jmaab.jc_0751
jc_0752: Condition action in transition label	"Check usage of parentheses in Stateflow transitions" (Simulink Check)	mathworks.jmaab.jc_0752
jc_0753: Condition actions and transition actions in Stateflow	"Check condition actions and transition actions in Stateflow" (Simulink Check)	mathworks.jmaab_v6.jc_0753
jc_0760: Starting point of internal transition	"Check starting point of internal transition in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0760
jc_0762: Prohibition of state action and flow chart combination	"Check prohibited combination of state action and flow chart" (Simulink Check)	mathworks.jmaab.jc_0762
jc_0763: Usage of multiple internal transitions	"Check usage of internal transitions in Stateflow states" (Simulink Check)	mathworks.jmaab.jc_0763
jc_0770: Position of transition label	"Check placement of Label String in Transitions" (Simulink Check)	mathworks.jmaab_v6.jc_0770
jc_0771: Comment position in transition labels	"Check position of comments in transition labels" (Simulink Check)	mathworks.jmaab.jc_0771
jc_0772: Execution order and transition conditions of transition lines	"Check usage of transition conditions in Stateflow transitions" (Simulink Check)	mathworks.jmaab.jc_0772

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
jc_0773: Unconditional transition of a flow chart	"Check usage of unconditional transitions in flow charts" (Simulink Check)	mathworks.jmaab.jc_0773
jc_0774: Comments for through transition	"Check for comments in unconditional transitions" (Simulink Check)	mathworks.jmaab.jc_0774
jc_0775: Terminating junctions in flow charts	"Check terminal junctions in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0775
jc_0790: Action language of Chart block	"Check Stateflow chart action language" (Simulink Check)	mathworks.jmaab.jc_0790
jc_0791: Duplicate data name definitions	"Check duplication of Simulink data names" (Simulink Check)	mathworks.jmaab.jc_0791
jc_0792: Unused Data	"Check unused data in Simulink Model" (Simulink Check)	mathworks.jmaab.jc_0792
jc_0794: Division in Simulink	"Check for division by zero in Simulink" (Simulink Check)	mathworks.jmaab.jc_0794
jc_0795: Usable characters for Stateflow data names	"Check usable characters for Stateflow data names" (Simulink Check)	mathworks.jmaab.jc_0795
jc_0796: Length restriction for Stateflow data names	"Check length of Stateflow data name" (Simulink Check)	mathworks.jmaab.jc_0796
jc_0797: Unconnected transitions / states / connective junctions	"Check for unconnected objects in Stateflow Charts" (Simulink Check)	mathworks.jmaab.jc_0797
jc_0800: Comparing floating-point types in Simulink	"Check comparison of floating point types in Simulink" (Simulink Check)	mathworks.jmaab.jc_0800
jc_0801: Prohibited use of the /* and */ comment symbols	"Check for use of C-style comment symbols" (Simulink Check)	mathworks.jmaab.jc_0801
jc_0802: Prohibited use of implicit type casting in Stateflow	"Check for implicit type casting in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0802
jc_0803: Passing values to library functions	<i>Not checkable</i>	
jc_0804: Prohibited use of recursive calls with graphical functions	"Check usage of graphical functions in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0804
jc_0805: Numerical operation block inputs	<i>No check</i>	

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
jc_0806: Detecting incorrect calculation results	"Check diagnostic settings for incorrect calculation results" (Simulink Check)	mathworks.jmaab.jc_0806
jc_0900: Usable characters for data type definition	"Check bus and enumeration data type names" (Simulink Check)	mathworks.jmaab_v6.jc_0900
jc_0901: Length restriction for Bus and Enumeration data type names	"Check length of bus and enumeration data type names" (Simulink Check)	mathworks.jmaab_v6.jc_0901
jc_0902: Arrowhead size of transition lines	"Check arrowhead size of transition lines" (Simulink Check)	mathworks.jmaab_v6.jc_0902
jc_0903: Prohibition of overlapping or crossing of blocks and signal lines	"Check for prohibited overlapping or intersecting blocks and signal lines" (Simulink Check)	mathworks.jmaab_v6.jc_0903
jc_0904: Prohibition of overlap/intersection of states and transition lines	"Check for prohibited overlapping of states and transition lines in Stateflow charts" (Simulink Check)	mathworks.jmaab_v6.jc_0904
jc_0905: Usable characters for data names in Functions	"Check data names in MATLAB functions" (Simulink Check)	mathworks.jmaab_v6.jc_0905
jc_0906: Length restriction of data names in MATLAB Functions	"Check the length of data names in MATLAB functions" (Simulink Check)	mathworks.jmaab_v6.jc_0906
jc_0907: Size of a Junction	"Check size of junctions" (Simulink Check)	mathworks.jmaab_v6.jc_0907
jm_0002: Block resizing	<i>No check</i>	
jm_0011: Pointers in Stateflow	"Check for pointers in Stateflow charts" (Simulink Check)	mathworks.maab.jm_0011
jm_0012: Usage restrictions of events and broadcasting events	"Check for usage of events in Stateflow charts" (Simulink Check)	mathworks.jmaab_v6.jm_0012
mp_0007: How to describe executable statements	"Check description of execution statements" (Simulink Check)	mathworks.jmaab_v6.mp_0007
mp_0008: Format of parentheses	"Check for spaces between function or variable names and left parenthesis symbol" (Simulink Check)	mathworks.jmaab_v6.mp_0008
mp_0010: Precedence of Operators in Arithmetic Expressions	"Check for operator precedence" (Simulink Check)	mathworks.jmaab_v6.mp_0010

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
mp_0011: Method of inserting a single-width space	"Check spaces in expressions" (Simulink Check)	mathworks.jmaab_v6.mp_0011
mp_0022: Format of Conditional Expressions	"Check description of conditional expressions" (Simulink Check)	mathworks.jmaab_v6.mp_0022
mp_0023: Format of relational operations using constants	"Check relational operators usage" (Simulink Check)	mathworks.jmaab_v6.mp_0023
mp_0032: Function header information	"Check function headers" (Simulink Check)	mathworks.jmaab_v6.mp_0032
mp_0034: Number of lines in a function	"Check number of lines of functions" (Simulink Check)	mathworks.jmaab_v6.mp_0034
mp_0040: Using the return value of a function	"Check for utilization of the return value of functions" (Simulink Check)	mathworks.jmaab_v6.mp_0040
mp_0046: Usage of expressions in array indices	"Check array indices" (Simulink Check)	mathworks.jmaab_v6.mp_0046
mp_0047: Conditions that a nonempty statement must satisfy	"Check for usage of nonempty statements" (Simulink Check)	mathworks.jmaab_v6.mp_0047
na_0001: Standard usage of Stateflow operators	"Check Stateflow operators" (Simulink Check)	mathworks.jmaab.na_0001
na_0002: Appropriate usage of basic logical and numerical operations	"Check fundamental logical and numerical operations" (Simulink Check)	mathworks.jmaab.na_0002
na_0003: Usage of If blocks	"Check logical expressions in If blocks" (Simulink Check)	mathworks.maab.na_0003
na_0004: Simulink model appearance settings	"Check for Simulink diagrams using nonstandard display attributes" (Simulink Check)	mathworks.maab.na_0004
na_0008: Display of labels on signals	"Check signal line labels" (Simulink Check)	mathworks.maab.na_0008
na_0009: Entry versus propagation of signal labels	"Check for propagated signal labels" (Simulink Check)	mathworks.maab.na_0009
na_0010: Usage of vector and bus signals	"Check usage of vector and bus signals" (Simulink Check)	mathworks.jmaab.na_0010
na_0011: Scope of Goto and From blocks	"Check scope of From and Goto blocks" (Simulink Check)	mathworks.jmaab_v6.na_0011
na_0016: Source lines of MATLAB Functions	"Check lines of code in MATLAB Functions" (Simulink Check)	mathworks.jmaab.na_0016
na_0017: Number of called function levels	"Check the number of function calls in MATLAB Function blocks" (Simulink Check)	mathworks.jmaab.na_0017

MAB Modeling Guideline (Version 6.0)	Model Advisor Check	Check ID
na_0018: Number of nested if/else and case statement	"Check nested conditions in MATLAB Functions" (Simulink Check)	mathworks.jmaab.na_0018
na_0019: Restricted variable names	"Check usage of restricted variable names" (Simulink Check)	mathworks.maab.na_0019
na_0020: Number of inputs to variant subsystems	"Check for missing ports in Variant Subsystems" (Simulink Check)	mathworks.jmaab_v6.na_0020
na_0021: Strings in MATLAB functions	"Check usage of character vector inside MATLAB Function block" (Simulink Check)	mathworks.maab.na_0021
na_0022: Recommended patterns for Switch/Case statements	"Check usage of recommended patterns for Switch/Case statements" (Simulink Check)	mathworks.maab.na_0022
na_0024: Shared data in MATLAB functions	"Check MATLAB code for global variables" (Simulink Check)	mathworks.maab.na_0024
na_0025: MATLAB Function header	<i>No check</i>	
na_0031: Definition of default enumerated value	"Check usage of enumerated values" (Simulink Check)	mathworks.maab.na_0031
na_0034: MATLAB Function block input/output settings	"Check input and output settings of MATLAB Functions" (Simulink Check)	mathworks.maab.na_0034
na_0036: Default variant	"Check use of default variants" (Simulink Check)	mathworks.maab.na_0036
na_0037: Use of single variable for variant condition	"Check use of single variable variant conditionals" (Simulink Check)	mathworks.maab.na_0037
na_0039: Limitation on Simulink functions in Chart blocks	"Check use of Simulink in Stateflow charts" (Simulink Check)	mathworks.maab.na_0039
na_0042: Usage of Simulink functions	"Check usage of Simulink function in Stateflow" (Simulink Check)	mathworks.jmaab.na_0042

Model Advisor Checks for JMAAB Modeling Guidelines

This table identifies the Model Advisor checks that you can use to verify compliance of your model with Japan MathWorks Automotive Advisor Board (JMAAB) modeling guidelines.

In the Model Advisor, JMAAB checks are organized by guideline version. To access the checks, select:

- **By Product > Simulink Check > Modeling Standards** and choose:
 - **JMAAB v5.1 Checks** — Checks for version 5.1 of the JMAAB modeling guidelines.
 - **JMAAB v6.0 Checks** — Checks for version 6.0 of the JMAAB modeling guidelines.
- **By Task** and choose:
 - **Modeling Standards for JMAAB v5.1** — Checks for version 5.1 of the JMAAB modeling guidelines.
 - **Modeling Standards for JMAAB v6.0** — Checks for version 6.0 of the JMAAB modeling guidelines.

You need Simulink Check to execute the Model Advisor checks.

Several modeling guidelines have changed between versions of publication *JMAAB Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow*, resulting in new Model Advisor checks. New check IDs for version 6.0 are in the format `mathworks.jmaab_v6.<guideline ID>`. For all other checks, a single Model Advisor check is applicable for both modeling guideline versions.

Guidelines that do not have corresponding Model Advisor checks are identified by using:

- *No check* — Indicates that the guideline can be checked by using a Model Advisor check, however, the check does not currently exist.
- *Not checkable* — Indicates that it is not possible to verify compliance to this guideline by using a Model Advisor check.

Use these links to view the version-specific modeling guideline:

- For version 5.1, see *Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow*
- For version 6.0, see *Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow*

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
ar_0001: Usable characters for file names	Version 5.1	“Check file names” (Simulink Check)	<code>mathworks.jmaab.ar_0001</code>
	Version 6.0	“Check file names” (Simulink Check)	<code>mathworks.jmaab_v6.ar_0001</code>
ar_0002: Usable characters for folder names	Version 5.1	“Check folder names” (Simulink Check)	<code>mathworks.jmaab.ar_0002</code>
	Version 6.0	“Check folder names” (Simulink Check)	<code>mathworks.jmaab_v6.ar_0002</code>

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
db_0032: Signal line connections	Version 5.1	"Check signal line connections" (Simulink Check)	mathworks.jmaab.db_0032
	Version 6.0	"Check signal line connections" (Simulink Check)	mathworks.jmaab_v6.db_0032
db_0042: Usage of Import and Outport blocks	Version 5.1	"Check position of Import and Outport blocks" (Simulink Check)	mathworks.jmaab.db_0042
	Version 6.0		
db_0043: Model font and font size	Version 5.1	"Check model font settings" (Simulink Check)	mathworks.jmaab.db_0043
	Version 6.0		
db_0081: Unconnected signals and blocks	Version 5.1	"Check for unconnected signal lines and blocks" (Simulink Check)	mathworks.jmaab.db_0081
	Version 6.0		
db_0097: Position of labels for signals and buses	Version 5.1	"Check position of signal labels" (Simulink Check)	mathworks.jmaab.db_0097
	Version 6.0		
db_0110: Block parameters	Version 5.1	"Check usage of tunable parameters in blocks" (Simulink Check)	mathworks.maab.db_0110
	Version 6.0		
db_0112: Usage of index	Version 5.1	"Check Indexing Mode" (Simulink Check)	mathworks.jmaab.db_0112
	Version 6.0		
db_0125: Stateflow local data	Version 5.1	"Check definition of Stateflow data" (Simulink Check)	mathworks.jmaab.db_0125
	Version 6.0		
db_0126: Defining Stateflow events	Version 5.1	"Check definition of Stateflow events" (Simulink Check)	mathworks.jmaab.db_0126
	Version 6.0		
db_0127: Limitation on MATLAB commands in Stateflow blocks	Version 5.1	"Check for MATLAB expressions in Stateflow charts" (Simulink Check)	mathworks.jmaab.db_0127
	Version 6.0		

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
db_0129: Stateflow transition appearance	Version 5.1	"Check for Stateflow transition appearance" (Simulink Check)	mathworks.jmaab.db_0129
	Version 6.0	"Check for Stateflow transition appearance" (Simulink Check)	mathworks.jmaab_v6.db_0129
db_0132: Transitions in flow charts	Version 5.1	"Check transitions in Stateflow Flow charts" (Simulink Check)	mathworks.jmaab.db_0132
	Version 6.0	"Check transitions in Stateflow Flow charts" (Simulink Check)	
db_0137: States in state machines	Version 5.1	"Check for state in state machines" (Simulink Check)	mathworks.jmaab.db_0137
	Version 6.0	"Check for state in state machines" (Simulink Check)	
db_0140: Display of block parameters	Version 5.1	"Check for display of block parameter" (Simulink Check)	mathworks.maab.db_0140
	Version 6.0	"Check for display of block parameter" (Simulink Check)	
db_0141: Signal flow in Simulink models	Version 5.1	"Check signal flow in model" (Simulink Check)	mathworks.maab.db_0141
	Version 6.0	"Check signal flow in model" (Simulink Check)	
db_0142: Position of block names	Version 5.1	"Check whether block names appear below blocks" (Simulink Check)	mathworks.maab.db_0142
	Version 6.0	"Check whether block names appear below blocks" (Simulink Check)	
db_0143: Usable block types in model hierarchy	Version 5.1	"Check for mixing basic blocks and subsystems" (Simulink Check)	mathworks.maab.db_0143
	Version 6.0	"Check for mixing basic blocks and subsystems" (Simulink Check)	
db_0144: Use of subsystems	Version 5.1	<i>Not checkable</i>	
	Version 6.0	<i>Not checkable</i>	
db_0146: Block layout in conditional subsystems	Version 5.1	"Check position of conditional blocks and iterator blocks" (Simulink Check)	mathworks.jmaab.db_0146
	Version 6.0	"Check position of conditional blocks and iterator blocks" (Simulink Check)	
jc_0008: Definition of signal names	Version 5.1	"Check definition of signal labels" (Simulink Check)	mathworks.jmaab.jc_0008
	Version 6.0	"Check definition of signal labels" (Simulink Check)	
jc_0009: Signal name propagation	Version 5.1	"Check signal name propagation" (Simulink Check)	mathworks.jmaab.jc_0009
	Version 6.0	"Check signal name propagation" (Simulink Check)	mathworks.jmaab_v6.jc_0009
jc_0011: Optimization parameters for Boolean data types	Version 5.1	"Check Implement logic signals as Boolean data (vs. double)" (Simulink Check)	mathworks.maab.jc_0011
	Version 6.0	"Check Implement logic signals as Boolean data (vs. double)" (Simulink Check)	

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
jc_0061: Display of block names	Version 5.1	“Check the display attributes of block names” (Simulink Check)	mathworks.maab.jc_0061
	Version 6.0		
jc_0081: Import and Outport block icon display	Version 5.1	“Check display for port blocks” (Simulink Check)	mathworks.maab.jc_0081
	Version 6.0		
jc_0110: Direction of block	Version 5.1	“Check block orientation” (Simulink Check)	mathworks.jmaab.jc_0110
	Version 6.0		
jc_0121: Usage of add and subtraction blocks	Version 5.1	“Check usage of Sum blocks” (Simulink Check)	mathworks.jmaab.jc_0121
	Version 6.0		
jc_0131: Usage of Relational Operator blocks	Version 5.1	“Check usage of Relational Operator blocks” (Simulink Check)	mathworks.maab.jc_0131
	Version 6.0		
jc_0141: Usage of the Switch blocks	Version 5.1	“Check usage of Switch blocks” (Simulink Check)	mathworks.maab.jc_0141
	Version 6.0		
jc_0161: Definition of Data Store Memory blocks	Version 5.1	“Check for usage of Data Store Memory blocks” (Simulink Check)	mathworks.jmaab.jc_0161
	Version 6.0		
jc_0171: Clarification of connections between structural subsystems	Version 5.1	“Check connections between structural subsystems” (Simulink Check)	mathworks.jmaab.jc_0171
	Version 6.0		
jc_0201: Usable characters for subsystem names	Version 5.1	“Check subsystem names” (Simulink Check)	mathworks.jmaab.jc_0201
	Version 6.0		
jc_0211: Usable characters for Import blocks and Outport block	Version 5.1	“Check port block names” (Simulink Check)	mathworks.jmaab.jc_0211
	Version 6.0		
jc_0222: Usable characters for signal and bus names	Version 5.1	“Check usable characters for signal names and bus names” (Simulink Check)	mathworks.jmaab.jc_0222
	Version 6.0		
jc_0231: Usable characters for block names	Version 5.1	“Check character usage in block names” (Simulink Check)	mathworks.jmaab.jc_0231
	Version 6.0		

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
jc_0232: Usable characters for parameter names	Version 5.1	"Check usable characters for parameter names" (Simulink Check)	mathworks.jmaab.jc_0232
	Version 6.0	"Check usable characters for parameter names" (Simulink Check)	mathworks.jmaab_v6.jc_0232
jc_0241: Length restriction for model file names	Version 5.1	"Check length of model file name" (Simulink Check)	mathworks.jmaab.jc_0241
	Version 6.0	"Check length of folder name at every level of model path" (Simulink Check)	
jc_0242: Length restriction for folder names	Version 5.1	"Check length of folder name at every level of model path" (Simulink Check)	mathworks.jmaab.jc_0242
	Version 6.0	"Check length of subsystem names" (Simulink Check)	
jc_0243: Length restriction for subsystem names	Version 5.1	"Check length of Inport and Outport names" (Simulink Check)	mathworks.jmaab.jc_0243
	Version 6.0	"Check length of signal and bus names" (Simulink Check)	
jc_0244: Length restriction for Import and Outport names	Version 5.1	"Check length of parameter names" (Simulink Check)	mathworks.jmaab.jc_0244
	Version 6.0	"Check length of block names" (Simulink Check)	
jc_0245: Length restriction for signal and bus names	Version 5.1	"Check trigger signal names" (Simulink Check)	mathworks.jmaab.jc_0245
	Version 6.0	"Check usage of unary minus operations in Stateflow charts" (Simulink Check)	
jc_0246: Length restriction for parameter name	Version 5.1	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	mathworks.jmaab.jc_0246
	Version 6.0	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	
jc_0247: Length restriction for block names	Version 5.1	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	mathworks.jmaab.jc_0247
	Version 6.0	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	
jc_0281: Trigger signal names	Version 5.1	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	mathworks.jmaab.jc_0281
	Version 6.0	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	
jc_0451: Use of unary minus on unsigned integers	Version 5.1	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	mathworks.jmaab.jc_0451
	Version 6.0	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	
jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow	Version 5.1	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	mathworks.maab.jc_0481
	Version 6.0	"Check usage of floating-point expressions in Stateflow charts" (Simulink Check)	

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
		“Check usage of floating-point expressions in Stateflow charts” (Simulink Check)	mathworks.jmaab_v6.jc_0481
jc_0491: Reuse of Stateflow data	Version 5.1	<i>Not checkable</i>	
	Version 6.0		
jc_0501: Format of entries in a State block	Version 5.1	“Check entry formatting in State blocks in Stateflow charts” (Simulink Check)	mathworks.jmaab.jc_0501
	Version 6.0		
jc_0511: Return values from a graphical function	Version 5.1	“Check return value assignments in Stateflow graphical functions” (Simulink Check)	mathworks.maab.jc_0511
	Version 6.0		
jc_0531: Default transition	Version 5.1	“Check default transition placement in Stateflow charts” (Simulink Check)	mathworks.jmaab.jc_0531
	Version 6.0		
jc_0602: Consistency in model element names	Version 5.1	“Check for consistency in model element names” (Simulink Check)	mathworks.jmaab.jc_0602
	Version 6.0		
jc_0603: Model description	Version 5.1	“Check Model Description” (Simulink Check)	mathworks.jmaab.jc_0603
	Version 6.0		
jc_0604: Using block shadow	Version 5.1	“Check if blocks are shaded in the model” (Simulink Check)	mathworks.jmaab.jc_0604
	Version 6.0		
jc_0610: Operator order for multiplication and division block	Version 5.1	“Check operator order of Product blocks” (Simulink Check)	mathworks.jmaab.jc_0610
	Version 6.0		
jc_0611: Input sign for multiplication and division blocks	Version 5.1	“Check signs of input signals in product blocks” (Simulink Check)	mathworks.jmaab.jc_0611
	Version 6.0		
jc_0621: Usage of Logical Operator blocks	Version 5.1	“Check icon shape of Logical Operator blocks” (Simulink Check)	mathworks.jmaab.jc_0621
	Version 6.0		
jc_0622: Usage of Fcn blocks	Version 5.1	“Check for parentheses in Fcn block expressions” (Simulink Check)	mathworks.jmaab.jc_0622

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
	Version 6.0		
jc_0623: Usage of continuous-time Delay blocks and discrete-time Delay blocks	Version 5.1	“Check usage of Memory and Unit Delay blocks” (Simulink Check)	<code>mathworks.jmaab.jc_0623</code>
	Version 6.0		
jc_0624: Usage of Tapped Delay blocks/Delay blocks	Version 5.1	“Check for cascaded Unit Delay blocks” (Simulink Check)	<code>mathworks.jmaab.jc_0624</code>
	Version 6.0		
jc_0626: Usage of Lookup Table blocks	Version 5.1	“Check usage of Lookup Tables” (Simulink Check)	<code>mathworks.jmaab.jc_0626</code>
	Version 6.0		
jc_0627: Usage of Discrete-Time Integrator blocks	Version 5.1	“Check usage of Discrete-Time Integrator block” (Simulink Check)	<code>mathworks.jmaab.jc_0627</code>
	Version 6.0	“Check usage of Discrete-Time Integrator block” (Simulink Check)	<code>mathworks.jmaab_v6.jc_0627</code>
jc_0628: Usage of Saturation blocks	Version 5.1	“Check usage of the Saturation blocks” (Simulink Check)	<code>mathworks.jmaab.jc_0628</code>
	Version 6.0		
jc_0630: Usage of Multiport Switch blocks	Version 5.1	“Check settings for data ports in Multiport Switch blocks” (Simulink Check)	<code>mathworks.jmaab.jc_0630</code>
	Version 6.0	“Check settings for data ports in Multiport Switch blocks” (Simulink Check)	<code>mathworks.jmaab_v6.jc_0630</code>
jc_0640: Initial value settings for Outport blocks in conditional subsystems	Version 5.1	“Check undefined initial output for conditional subsystems” (Simulink Check)	<code>mathworks.jmaab.jc_0640</code>
	Version 6.0		
jc_0641: Sample time setting	Version 5.1	“Check for sample time setting” (Simulink Check)	<code>mathworks.jmaab.jc_0641</code>
	Version 6.0		
jc_0642: Integer rounding mode setting	Version 5.1	“Check Signed Integer Division Rounding mode” (Simulink Check)	<code>mathworks.jmaab.jc_0642</code>
	Version 6.0		
jc_0643: Fixed-point setting	Version 5.1	“Check usage of fixed-point data type with non-zero bias” (Simulink Check)	<code>mathworks.jmaab.jc_0643</code>
	Version 6.0		

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
jc_0644: Type setting	Version 5.1	"Check type setting by data objects" (Simulink Check)	mathworks.jmaab.jc_0644
	Version 6.0	"Check type setting by data objects" (Simulink Check)	mathworks.jmaab_v6.jc_0644
jc_0645: Parameter definition for calibration	Version 5.1	"Check if tunable block parameters are defined as named constants" (Simulink Check)	mathworks.jmaab.jc_0645
	Version 6.0		
jc_0650: Block input/output data type with switching function	Version 5.1	"Check input and output datatype for Switch blocks" (Simulink Check)	mathworks.jmaab.jc_0650
	Version 6.0		
jc_0651: Implementing a type conversion	Version 5.1	"Check output data type of operation blocks" (Simulink Check)	mathworks.jmaab.jc_0651
	Version 6.0	"Check Output data type of operation blocks" (Simulink Check)	mathworks.jmaab_v6.jc_0651
jc_0653: Delay block layout in feedback loops	Version 5.1	"Check for avoiding algebraic loops between subsystems" (Simulink Check)	mathworks.jmaab.jc_0653
	Version 6.0		
jc_0655: Prohibition of logical value comparison in Stateflow	Version 5.1	"Check prohibited comparison operation of logical type signals" (Simulink Check)	mathworks.jmaab.jc_0655
	Version 6.0		
jc_0656: Usage of Conditional Control blocks	Version 5.1	"Check default/else case in Switch Case blocks and If blocks" (Simulink Check)	mathworks.jmaab.jc_0656
	Version 6.0		
jc_0657: Retention of output value based on conditional control flow blocks and Merge blocks	Version 5.1	<i>Not checkable</i>	
	Version 6.0		
jc_0659: Usage restrictions of signal lines input to Merge blocks	Version 5.1	"Check usage of Merge block" (Simulink Check)	mathworks.jmaab.jc_0659
	Version 6.0		
jc_0700: Unused data in Stateflow block	Version 5.1	"Check for unused data in Stateflow Charts" (Simulink Check)	mathworks.jmaab.jc_0700
	Version 6.0		

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
jc_0701: Usable number for first index	Version 5.1	"Check usable number for first index" (Simulink Check)	mathworks.jmaab.jc_0701
	Version 6.0		
jc_0702: Use of named Stateflow parameters and constants	Version 5.1	"Check usage of numeric literals in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0702
	Version 6.0		
jc_0711: Division in Stateflow	Version 5.1	<i>Not checkable</i>	
	Version 6.0		
jc_0712: Execution timing for default transition path	Version 5.1	"Check execution timing for default transition path" (Simulink Check)	mathworks.jmaab.jc_0712
	Version 6.0		
jc_0721: Usage of parallel states	Version 5.1	"Check usage of parallel states" (Simulink Check)	mathworks.jmaab.jc_0721
	Version 6.0		
jc_0722: Local data definition in parallel states	Version 5.1	"Check scope of data in parallel states" (Simulink Check)	mathworks.jmaab.jc_0722
	Version 6.0		
jc_0723: Prohibited direct transition from external state to child state	Version 5.1	"Check usage of transitions to external states" (Simulink Check)	mathworks.jmaab.jc_0723
	Version 6.0		
jc_0730: Unique state name in Stateflow blocks	Version 5.1	"Check uniqueness of State names" (Simulink Check)	mathworks.jmaab.jc_0730
	Version 6.0		
jc_0731: State name format	Version 5.1	"Check usage of State names" (Simulink Check)	mathworks.jmaab.jc_0731
	Version 6.0		
jc_0732: Distinction between state names, data names, and event names	Version 5.1	"Check uniqueness of Stateflow state, data and event names" (Simulink Check)	mathworks.jmaab.jc_0732
	Version 6.0		
jc_0733: Order of state action types	Version 5.1	"Check order of state action types" (Simulink Check)	mathworks.jmaab.jc_0733
	Version 6.0		
jc_0734: Number of state action types	Version 5.1	"Check repetition of action types" (Simulink Check)	mathworks.jmaab.jc_0734
	Version 6.0		
jc_0736: Uniform indentations in Stateflow blocks	Version 6.0	"Check indentation of code in Stateflow states" (Simulink Check)	mathworks.jmaab.jc_0736
	Version 5.1		

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
jc_0738: Usage of Stateflow comments	Version 6.0	“Check usage of Stateflow comments” (Simulink Check)	mathworks.jmaab.jc_0738
	Version 5.1		
jc_0739: Describing text inside states	Version 5.1	“Check for usage of text inside states” (Simulink Check)	mathworks.jmaab.jc_0739
jc_0740: Limitation on use of exit state action	Version 5.1	“Check if state action type ‘exit’ is used in the model” (Simulink Check)	mathworks.jmaab.jc_0740
	Version 6.0		
jc_0741: Timing to update data used in state chart transition conditions	Version 5.1	“Check updates to variables used in state transition conditions” (Simulink Check)	mathworks.jmaab.jc_0741
	Version 6.0	“Check updates to variables used in state transition conditions” (Simulink Check)	mathworks.jmaab_v6.jc_0741
jc_0751: Backtracking prevention in state transition	Version 5.1	“Check for unexpected backtracking in state transitions” (Simulink Check)	mathworks.jmaab.jc_0751
	Version 6.0		
jc_0752: Condition action in transition label	Version 5.1	“Check usage of parentheses in Stateflow transitions” (Simulink Check)	mathworks.jmaab.jc_0752
	Version 6.0		
jc_0753: Condition actions and transition actions in Stateflow	Version 5.1	“Check condition actions and transition actions in Stateflow” (Simulink Check)	mathworks.jmaab.jc_0753
	Version 6.0	“Check condition actions and transition actions in Stateflow” (Simulink Check)	mathworks.jmaab_v6.jc_0753
jc_0760: Starting point of internal transition	Version 5.1	“Check starting point of internal transition in Stateflow” (Simulink Check)	mathworks.jmaab.jc_0760
	Version 6.0		
jc_0762: Prohibition of state action and flow chart combination	Version 5.1	“Check prohibited combination of state action and flow chart” (Simulink Check)	mathworks.jmaab.jc_0762
	Version 6.0		
jc_0763: Usage of multiple internal transitions	Version 5.1	“Check usage of internal transitions in	mathworks.jmaab.jc_0763

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
	Version 6.0	"Stateflow states" (Simulink Check)	
jc_0770: Position of transition label	Version 5.1	"Check placement of Label String in Transitions" (Simulink Check)	mathworks.jmaab.jc_0770
	Version 6.0	"Check placement of Label String in Transitions" (Simulink Check)	mathworks.jmaab_v6.jc_0770
jc_0771: Comment position in transition labels	Version 5.1	"Check position of comments in transition labels" (Simulink Check)	mathworks.jmaab.jc_0771
	Version 6.0		
jc_0772: Execution order and transition conditions of transition lines	Version 5.1	"Check usage of transition conditions in Stateflow transitions" (Simulink Check)	mathworks.jmaab.jc_0772
	Version 6.0		
jc_0773: Unconditional transition of a flow chart	Version 5.1	"Check usage of unconditional transitions in flow charts" (Simulink Check)	mathworks.jmaab.jc_0773
	Version 6.0		
jc_0774: Comments for through transition	Version 5.1	"Check for comments in unconditional transitions" (Simulink Check)	mathworks.jmaab.jc_0774
	Version 6.0		
jc_0775: Terminating junctions in flow charts	Version 5.1	"Check terminal junctions in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0775
	Version 6.0		
jc_0790: Action language of Chart block	Version 5.1	"Check Stateflow chart action language" (Simulink Check)	mathworks.jmaab.jc_0790
	Version 6.0		
jc_0791: Duplicate data name definitions	Version 5.1	"Check duplication of Simulink data names" (Simulink Check)	mathworks.jmaab.jc_0791
	Version 6.0		
jc_0792: Unused Data	Version 5.1	"Check unused data in Simulink Model" (Simulink Check)	mathworks.jmaab.jc_0792
	Version 6.0		
jc_0794: Division in Simulink	Version 5.1	"Check for division by zero in Simulink" (Simulink Check)	mathworks.jmaab.jc_0794
	Version 6.0		
jc_0795: Usable characters for Stateflow data names	Version 5.1	"Check usable characters for Stateflow	mathworks.jmaab.jc_0795

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
	Version 6.0	"data names" (Simulink Check)	
jc_0796: Length restriction for Stateflow data names	Version 5.1	"Check length of Stateflow data name" (Simulink Check)	mathworks.jmaab.jc_0796
	Version 6.0		
jc_0797: Unconnected transitions / states / connective junctions	Version 5.1	"Check for unconnected objects in Stateflow Charts" (Simulink Check)	mathworks.jmaab.jc_0797
	Version 6.0		
jc_0800: Comparing floating-point types in Simulink	Version 5.1	"Check comparison of floating point types in Simulink" (Simulink Check)	mathworks.jmaab.jc_0800
	Version 6.0		
jc_0801: Prohibited use of the /* and // comment symbols	Version 5.1	"Check for use of C-style comment symbols" (Simulink Check)	mathworks.jmaab.jc_0801
	Version 6.0		
jc_0802: Prohibited use of implicit type casting in Stateflow	Version 5.1	"Check for implicit type casting in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0802
	Version 6.0		
jc_0803: Passing values to library functions	Version 5.1	<i>Not checkable</i>	
	Version 6.0		
jc_0804: Prohibited use of recursive calls with graphical functions	Version 5.1	"Check usage of graphical functions in Stateflow" (Simulink Check)	mathworks.jmaab.jc_0804
	Version 6.0		
jc_0805: Numerical operation block inputs	Version 5.1	<i>No check</i>	
	Version 6.0		
jc_0806: Detecting incorrect calculation results	Version 5.1	"Check diagnostic settings for incorrect calculation results" (Simulink Check)	mathworks.jmaab.jc_0806
jc_0900: Usable characters for Data Type definition	Version 6.0	"Check bus and enumeration data type names" (Simulink Check)	mathworks.jmaab_v6.jc_0900
jc_0901: Length restriction for Data Type definition	Version 6.0	"Check length of bus and enumeration data type names" (Simulink Check)	mathworks.jmaab_v6.jc_0901
jc_0902: Arrowhead size of transition lines	Version 6.0	"Check arrowhead size of transition lines" (Simulink Check)	mathworks.jmaab_v6.jc_0902

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
jc_0903: Prohibition of overlapping/crossing of blocks and signal line	Version 6.0	"Check for prohibited overlapping or intersecting blocks and signal lines" (Simulink Check)	mathworks.jmaab_v6.jc_0903
jc_0904: Prohibition of overlap/intersection of states and transition lines	Version 6.0	"Check for prohibited overlapping of states and transition lines in Stateflow charts" (Simulink Check)	mathworks.jmaab_v6.jc_0904
jc_0905: Usable characters for data names in MATLAB function	Version 6.0	"Check data names in MATLAB functions" (Simulink Check)	mathworks.jmaab_v6.jc_0905
jc_0906: Length restriction for data names in MATLAB function	Version 6.0	"Check the length of data names in MATLAB functions" (Simulink Check)	mathworks.jmaab_v6.jc_0906
jc_0907: Size of junctions	Version 6.0	"Check size of junctions" (Simulink Check)	mathworks.jmaab_v6.jc_0907
jm_0002: Block resizing	Version 5.1	<i>No check</i>	
	Version 6.0		
jm_0011: Pointers in Stateflow	Version 5.1	"Check for pointers in Stateflow charts" (Simulink Check)	mathworks.maab.jm_0011
	Version 6.0	"Check for pointers in Stateflow charts" (Simulink Check)	
jm_0012: Usage restrictions of events and broadcasting events	Version 5.1	"Check for usage of events in Stateflow charts" (Simulink Check)	mathworks.jmaab.jm_0012
	Version 6.0	"Check for usage of events in Stateflow charts" (Simulink Check)	mathworks.jmaab_v6.jm_0012
na_0001: Standard usage of Stateflow operators	Version 5.1	"Check Stateflow operators" (Simulink Check)	mathworks.jmaab.na_0001
	Version 6.0	"Check Stateflow operators" (Simulink Check)	
na_0002: Appropriate usage of basic logical and numerical operations	Version 5.1	"Check fundamental logical and numerical operations" (Simulink Check)	mathworks.jmaab.na_0002
	Version 6.0	"Check fundamental logical and numerical operations" (Simulink Check)	
na_0003: Usage of If blocks	Version 5.1	"Check logical expressions in If blocks" (Simulink Check)	mathworks.maab.na_0003
	Version 6.0	"Check logical expressions in If blocks" (Simulink Check)	

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
na_0004: Simulink model appearance settings	Version 5.1	“Check for Simulink diagrams using nonstandard display attributes” (Simulink Check)	mathworks.maab.na_0004
	Version 6.0		
na_0010: Usage of vector and bus signals	Version 5.1	“Check usage of vector and bus signals” (Simulink Check)	mathworks.jmaab.na_0010
	Version 6.0		
na_0011: Scope of Goto and From blocks	Version 5.1	“Check scope of From and Goto blocks” (Simulink Check)	mathworks.maab.na_0011
	Version 6.0	“Check scope of From and Goto blocks” (Simulink Check)	mathworks.jmaab_v6.na_0011
na_0020: Number of inputs to variant subsystems	Version 5.1	“Check for missing ports in Variant Subsystems” (Simulink Check)	mathworks.jmaab.na_0020
	Version 6.0	“Check for missing ports in Variant Subsystems” (Simulink Check)	mathworks.jmaab_v6.na_0020
na_0021: Strings in MATLAB functions	Version 5.1	“Check usage of character vector inside MATLAB Function block” (Simulink Check)	mathworks.maab.na_0021
	Version 6.0		
na_0024: Shared data in MATLAB functions	Version 5.1	“Check MATLAB code for global variables” (Simulink Check)	mathworks.maab.na_0024
	Version 6.0		
na_0031: Definition of default enumerated value	Version 5.1	“Check usage of enumerated values” (Simulink Check)	mathworks.maab.na_0031
	Version 6.0		
na_0034: MATLAB Function block input/output settings	Version 5.1	“Check input and output settings of MATLAB Functions” (Simulink Check)	mathworks.maab.na_0034
	Version 6.0		
na_0036: Default variant	Version 5.1	“Check use of default variants” (Simulink Check)	mathworks.maab.na_0036
	Version 6.0		
na_0037: Use of single variable for variant condition	Version 5.1	“Check use of single variable variant conditionals” (Simulink Check)	mathworks.maab.na_0037
	Version 6.0		

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
na_0039: Limitation on Simulink functions in Chart blocks	Version 5.1	“Check use of Simulink in Stateflow charts” (Simulink Check)	mathworks.maab.na_0039
	Version 6.0		
na_0042: Usage of Simulink functions	Version 5.1	“Check usage of Simulink function in Stateflow” (Simulink Check)	mathworks.jmaab.na_0042
	Version 6.0		
mp_0007: How to describe execution statements	Version 6.0	“Check description of execution statements” (Simulink Check)	mathworks.jmaab_v6.mp_0007
mp_0008: Format of parenthesis	Version 6.0	“Check for spaces between function or variable names and left parenthesis symbol” (Simulink Check)	mathworks.jmaab_v6.mp_0008
mp_0010: How to describe the priority of operators	Version 6.0	“Check for operator precedence” (Simulink Check)	mathworks.jmaab_v6.mp_0010
mp_0011: How to insert one single-byte space	Version 6.0	“Check spaces in expressions” (Simulink Check)	mathworks.jmaab_v6.mp_0011
mp_0016: Nesting levels of control statements	Version 6.0	“Check nested conditions in MATLAB Functions” (Simulink Check)	mathworks.jmaab.na_0018
mp_0020: How to describe Switch/Case statements	Version 6.0	“Check usage of recommended patterns for Switch/Case statements” (Simulink Check)	mathworks.maab.na_0022
mp_0022: How to describe conditional expression	Version 6.0	“Check description of conditional expressions” (Simulink Check)	mathworks.jmaab_v6.mp_0022
mp_0023: How to describe relational operators	Version 6.0	“Check relational operators usage” (Simulink Check)	mathworks.jmaab_v6.mp_0023
mp_0025: How to call functions	Version 6.0	“Check the number of function calls in MATLAB Function blocks” (Simulink Check)	mathworks.maab.na_0017
mp_0032: Function headers	Version 6.0	“Check function headers” (Simulink Check)	mathworks.jmaab_v6.mp_0032

JMAAB Modeling Guideline	Guideline Publication Version	Model Advisor Check	Check ID
mp_0034: Number of lines of functions	Version 6.0	"Check number of lines of functions" (Simulink Check)	mathworks.jmaab_v6.mp_0034
mp_0040: Utilizing the return value of functions	Version 6.0	"Check for utilization of the return value of functions" (Simulink Check)	mathworks.jmaab_v6.mp_0040
mp_0046: How to describe array indexes	Version 6.0	"Check array indices" (Simulink Check)	mathworks.jmaab_v6.mp_0046
mp_0047: The conditions for non-empty statements to be satisfied	Version 6.0	"Check for usage of nonempty statements" (Simulink Check)	mathworks.jmaab_v6.mp_0047

See Also

- For more information on Reserved MATLAB words, Built-in MATLAB functions and MATLAB keywords, refer Appendix C.

Naming Conventions

- “General Conventions” on page 2-2
- “Content Conventions” on page 2-11

General Conventions

ar_0001: Usable characters for file names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d, e, f, g
- JMAAB — a, b, c, d, e, f, g

MATLAB Versions

All

Rule

Sub ID a

Only these character types shall be used in file names:

- Single-byte alphanumeric characters (a-z, A-Z, 0-9)
- Single-byte underscore (_)

Line breaks, single-byte spaces, double-byte characters, and control characters shall not be used. File types that are checked for model and MATLAB files shall be set in the project settings.

Custom Parameter

File (extension)

Example — Incorrect

MAB Model.slx — Single-byte spaces are used

JMAAB 設定.m or NA-MAABModel.p — Double-byte characters are used.

JMAAB(Model).mdl — Symbol characters are used.

Sub ID b

The file name shall not use numbers at the beginning.

Custom Parameter

File (extension)

Example — Incorrect

001_JMAABModel.slx

Sub ID c

The file name shall not use underscores at the beginning.

Custom Parameter

File (extension)

Example — Incorrect

_JMAABModel.slx

Sub ID d

The file name shall not use an underscore at the end.

Custom Parameter

File (extension)

Example — Incorrect

MABModel_.slx

Sub ID e

The file name shall not use consecutive underscores.

Custom Parameter

File (extension)

Example — Incorrect

JMAAB__Model.slx

Sub ID f

The file name shall not consist solely of a single reserved MATLAB word.

Custom Parameter

File (extension)

Example — Incorrect

ans.slx, double.slx

Sub ID g

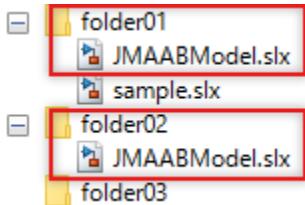
File names on the MATLAB path shall not be identical.

Custom Parameter

File (extension)

Example — Incorrect

Files with the same name are saved to the folder that goes through the MATLAB path.



Rationale

Sub IDs a, b, c, f:

- Readability is impaired.
- Deviation from the rule can cause unexpected issues.

Sub IDs d, e

- Readability is impaired.

Sub ID g:

- If there are multiple files with the same name, the one higher on the path is loaded. As a result, unnecessary files might be included.
- Readability is impaired.
- Deviation from the rule can cause unexpected issues.

Verification

Model Advisor check: “Check file names” (Simulink Check)

Last Changed

R2020a

See Also

- “Reserved Keywords” (Embedded Coder)
- “Reserved Identifiers and Code Replacement” (Embedded Coder)

Version History

Introduced in R2020a

ar_0002: Usable characters for folder names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d, e, f
- JMAAB — a, b, c, d, e, f

MATLAB Versions

All

Rule

Sub ID a

Only these character types shall be used in folder names:

- Single-byte alphanumeric characters (a - z, A - Z, 0 - 9)
- Single-byte underscore (_)

Line breaks, single-byte spaces, double-byte characters, and control characters shall not be used.

Custom Parameter

Applicable folder for the rule

Example — Incorrect

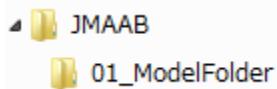


Sub ID b

The folder name shall not use numbers at the beginning.

Custom Parameter

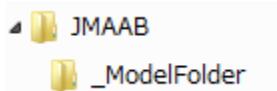
Applicable folder for the rule

Example — Incorrect**Sub ID c**

The folder name shall not use underscores at the beginning.

Custom Parameter

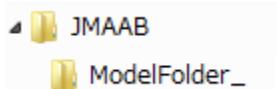
Applicable folder for the rule

Example — Incorrect**Sub ID d**

The folder name shall not use underscores at the end.

Custom Parameter

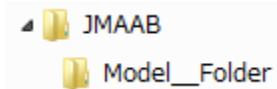
Applicable folder for the rule

Example — Incorrect**Sub ID e**

The folder name shall not use consecutive underscores.

Custom Parameter

Applicable folder for the rule

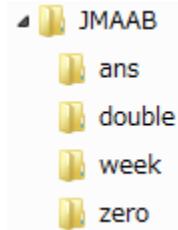
Example — Incorrect**Sub ID f**

The folder name shall not consist solely of a single reserved MATLAB word.

Custom Parameter

Applicable folder for the rule

Example — Incorrect



Rationale

Sub IDs a, b, c, d, e, f:

- Readability is impaired.
- Deviation from the rule can cause unexpected issues.

Verification

Model Advisor check: "Check folder names" (Simulink Check)

Last Changed

R2024b

See Also

- “Reserved Keywords” (Embedded Coder)
- “Reserved Identifiers and Code Replacement” (Embedded Coder)

Version History

Introduced in R2020a

jc_0241: Length restriction for model file names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Model file name length shall be a maximum of 63 characters (not including dots and extension).

Custom Parameter

Maximum model file name length

Rationale

Sub ID a:

- Possible that a long file name cannot be referred to in the model reference.

Verification

Model Advisor check: "Check length of model file name" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0242: Length restriction for folder names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Folder name length shall be a maximum of 63 characters.

Custom Parameter

Maximum folder name

Rationale

Sub ID a:

- Possible that the full path name cannot be displayed in the user interface.

Verification

Model Advisor check: "Check length of folder name at every level of model path" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

Content Conventions

jc_0201: Usable characters for subsystem names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d, e, f
- JMAAB — a, b, c, d, e, f

MATLAB Versions

All

Rule

Sub ID a

Only these character types shall be used in structural subsystem names:

- Single-byte alphanumeric characters (a - z, A - Z, 0 - 9)
- Single-byte underscore (_)

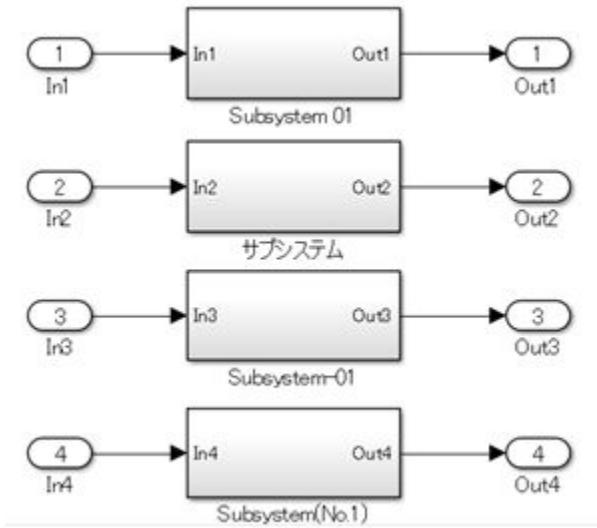
Line breaks, single-byte spaces, double-byte characters, and control characters shall not be used.

Custom Parameter

Not Applicable

Example — Incorrect

Single-byte spaces, double-byte, and symbol characters are used.



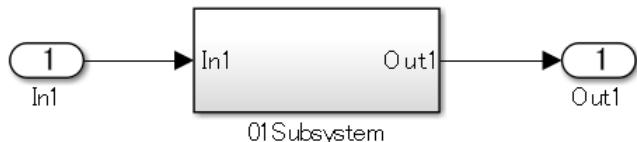
Sub ID b

A structural subsystem name shall not use numbers at the beginning.

Custom Parameter

Not Applicable

Example — Incorrect



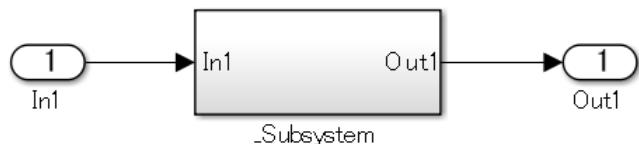
Sub ID c

A structural subsystem name shall not use an underscore at the beginning.

Custom Parameter

Not Applicable

Example — Incorrect



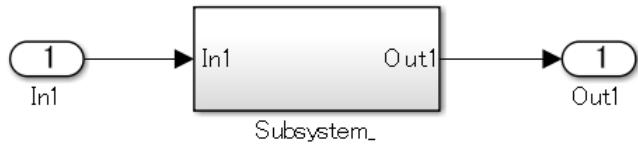
Sub ID d

A structural subsystem name shall not use an underscore at the end.

Custom Parameter

Not Applicable

Example — Incorrect



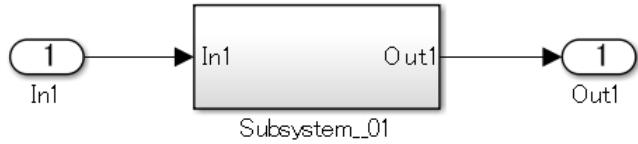
Sub ID e

A structural subsystem name shall not use consecutive underscores.

Custom Parameter

Not Applicable

Example — Incorrect



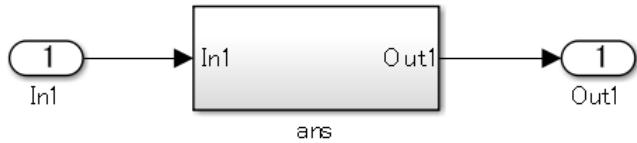
Sub ID f

A structural subsystem shall not consist solely of a single reserved MATLAB word.

Custom Parameter

Not Applicable

Example — Incorrect



Rationale

Sub IDs a, b, f:

- Cannot generate code using the configured structural subsystem name.

Sub IDs c, d, e:

- May not be able to generate code using the configured structural subsystem name.

Verification

Model Advisor check: "Check subsystem names" (Simulink Check)

Last Changed

R2020a

See Also

- “Explore Types of Subsystems”

Version History

Introduced in R2020a

jc_0231: Usable characters for block names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d, e, f
- JMAAB — a, b, c, d, e, f

MATLAB Versions

All

Rule

Sub ID a

Only these character types shall be used for basic block names:

- Single-byte alphanumeric characters (a-z, A-Z, 0-9)
- Single-byte underscore (_)

Line breaks and single-byte spaces shall not be permitted when adding a new block name. However, they shall be permitted when used initially as a block name that is saved in the Simulink library.

Double-byte characters and control characters shall not be used.

Exception

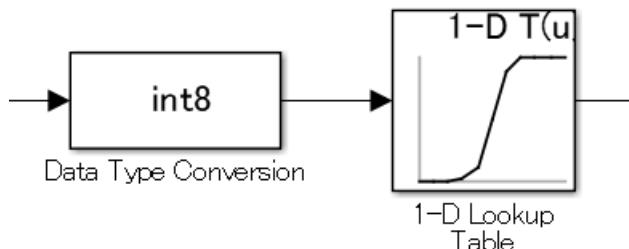
Import and Outport blocks

Custom Parameter

Not Applicable

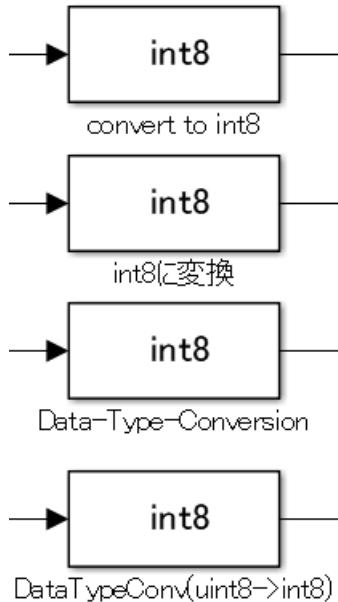
Example — Correct

Block names are registered in the Simulink library.



Example — Incorrect

Single-byte spaces, double-byte characters, and symbol characters are used.

**Sub ID b**

Basic block names shall not use numbers at the beginning.

Exception

Import and Outport blocks

Custom Parameter

Not Applicable

Example — Incorrect**Sub ID c**

Basic block names shall not use underscores at the beginning.

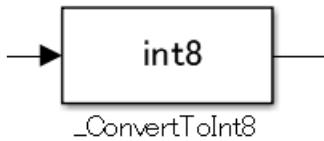
Exception

Import and Outport blocks

Custom Parameter

Not Applicable

Example — Incorrect



Sub ID d

Basic block names shall not use underscores at the end.

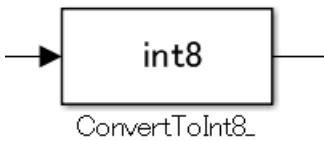
Exception

Input and Outport blocks

Custom Parameter

Not Applicable

Example — Incorrect



Sub ID e

Basic block names shall not use consecutive underscores.

Exception

Input and Outport blocks

Custom Parameter

Not Applicable

Example — Incorrect



Sub ID f

Basic block names shall not consist solely of a single reserved MATLAB word.

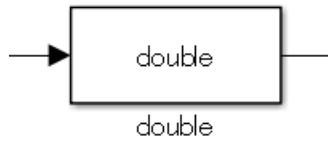
Exception

Input and Outport blocks

Custom Parameter

Not Applicable

Example — Incorrect



Rationale

Sub IDs a, b:

- Deviation from the rule can make it difficult to maintain the integrity of the model and code.

Sub IDs c, e:

- Readability is impaired.

Sub IDs d:

- Readability is impaired.
- Underscores can be used to separate words. However, they are typically used as word breaks and can cause misunderstanding in the description.

Sub IDs f:

- Readability is impaired.
- Deviation from the rule can cause unexpected issues.

Verification

Model Advisor check: “Check character usage in block names” (Simulink Check)

Last Changed

R2020a

See Also

- “Reserved Keywords” (Embedded Coder)
- “Reserved Identifiers and Code Replacement” (Embedded Coder)

Version History

Introduced in R2020a

jc_0211: Usable characters for Import blocks and Outport block

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d, e, f
- JMAAB — a, b, c, d, e, f

MATLAB Versions

All

Rule

Sub ID a

Only these character types shall be used in Import and Outport block names:

- Single-byte alphanumeric characters (a - z, A - Z, 0 - 9)
- Single-byte underscore (_)

Line breaks, single-byte spaces, double-byte characters, and control characters shall not be used.

Custom Parameter

Not Applicable

Example — Incorrect



Sub ID b

[Import] and [Outport] block names shall not use numbers at the beginning.

Custom Parameter

Not Applicable

Example — Incorrect



Sub ID c

[Import] and [Outport] block names shall not use underscores at the beginning.

Custom Parameter

Not Applicable

Example — Incorrect



Sub ID d

[Import] and [Outport] block names shall not use underscores at the end.

Custom Parameter

Not Applicable

Example — Incorrect



Sub ID e

[Import] and [Outport] block names shall not use consecutive underscores.

Custom Parameter

Not Applicable

Example — Incorrect



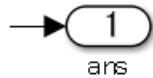
Sub ID f

[Import] and [Outport] block names shall not consist solely of a single reserved MATLAB word.

Custom Parameter

Not Applicable

Example — Incorrect



Rationale

Sub IDs a, b:

- Deviation from the rule can make it difficult to maintain the integrity of the model and code.

Sub IDs c, e:

- Readability is impaired.

Sub IDs d:

- Readability is impaired.
- Underscores can be used to separate words. However, they are typically used as word breaks and can cause misunderstanding in the description.

Sub IDs f:

- Readability is impaired.
- Deviation from the rule can cause unexpected issues.

Verification

Model Advisor check: "Check port block names" (Simulink Check)

Last Changed

R2020a

See Also

- “Reserved Keywords” (Embedded Coder)
- “Reserved Identifiers and Code Replacement” (Embedded Coder)

Version History

Introduced in R2020a

jc_0243: Length restriction for subsystem names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Structural subsystem name length shall be a maximum of 63 characters.

Custom Parameter

Maximum subsystem name length

Rationale

Sub ID a:

- Code generation may not be possible.

Verification

Model Advisor check: “Check length of subsystem names” (Simulink Check)

Last Changed

R2020a

See Also

- “Explore Types of Subsystems”

Version History

Introduced in R2020a

jc_0247: Length restriction for block names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Basic block name length shall be a maximum of 63 characters.

Exception

Import and Outport blocks

Custom Parameter

Maximum block name length

Rationale

Sub ID a:

- Code generation may not be possible.

Verification

Model Advisor check: “Check length of block names” (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0244: Length restriction for Import and Outport names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Import and Outport block name length shall be a maximum of 63 characters.

Custom Parameter

Maximum Import block name length

Maximum Outport block name length

Rationale

Sub ID a:

- Code generation may not be possible.

Verification

Model Advisor check: “Check length of Import and Outport names” (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0222: Usable characters for signal and bus names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d, e, f
- JMAAB — a, b, c, d, e, f

MATLAB Versions

All

Rule

Sub ID a

Only these character types shall be used in signal and bus names:

- Single-byte alphanumeric characters (a - z, A - Z, 0 - 9)
- Single-byte underscore (_)

Line breaks, single-byte spaces, double-byte characters, and control characters shall not be used.

Custom Parameter

Not Applicable

Sub ID b

Signal and bus names shall not use numbers at the beginning.

Custom Parameter

Not Applicable

Sub ID c

The signal or bus name shall not use underscores at the beginning.

Custom Parameter

Not Applicable

Sub ID d

Signal and bus names shall not use underscores at the end.

Custom Parameter

Not Applicable

Sub ID e

Signal and bus names shall not use consecutive underscores.

Custom Parameter

Not Applicable

Sub ID f

Signal and bus names shall not consist solely of a single reserved MATLAB word.

Custom Parameter

Not Applicable

Rationale

Sub IDs a, b:

- Deviation from the rule can make it difficult to maintain the integrity of the model and code.

Sub IDs c, e:

- Readability is impaired.

Sub IDs d:

- Readability is impaired.
- Underscores can be used to separate words. However, they are typically used as word breaks and can cause misunderstanding in the description.

Sub IDs f:

- Readability is impaired.
- Deviation from the rule can cause unexpected issues.

Verification

Model Advisor check: "Check usable characters for signal names and bus names" (Simulink Check)

Last Changed

R2020a

See Also

- “Reserved Keywords” (Embedded Coder)
- “Reserved Identifiers and Code Replacement” (Embedded Coder)
- “Signal Basics”
- “Simulink Bus Capabilities”

Version History

Introduced in R2020a

jc_0232: Usable characters for parameter names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

The parameter name shall not use underscores at the end.

Custom Parameter

Not Applicable

Sub ID b

The parameter name shall not use consecutive underscores.

Custom Parameter

Not Applicable

Sub ID c

The parameter name shall not consist solely of a single reserved MATLAB word.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Readability is impaired.
- Underscores can be used to separate words. However, they are typically used as word breaks and can cause misunderstanding in the description.

Sub ID b:

- Readability is impaired.

Sub ID c:

- Readability is impaired. Deviation from the rule can cause unexpected issues.

Verification

Model Advisor check: "Check usable characters for parameter names" (Simulink Check)

Last Changed

R2024b

See Also

- "Reserved Keywords" (Embedded Coder)
- "Reserved Identifiers and Code Replacement" (Embedded Coder)

Version History

Introduced in R2020a

jc_0245: Length restriction for signal and bus names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Signal and bus name length shall be a maximum of 63 characters.

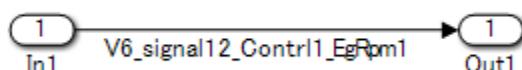
Custom Parameter

Maximum signal name length

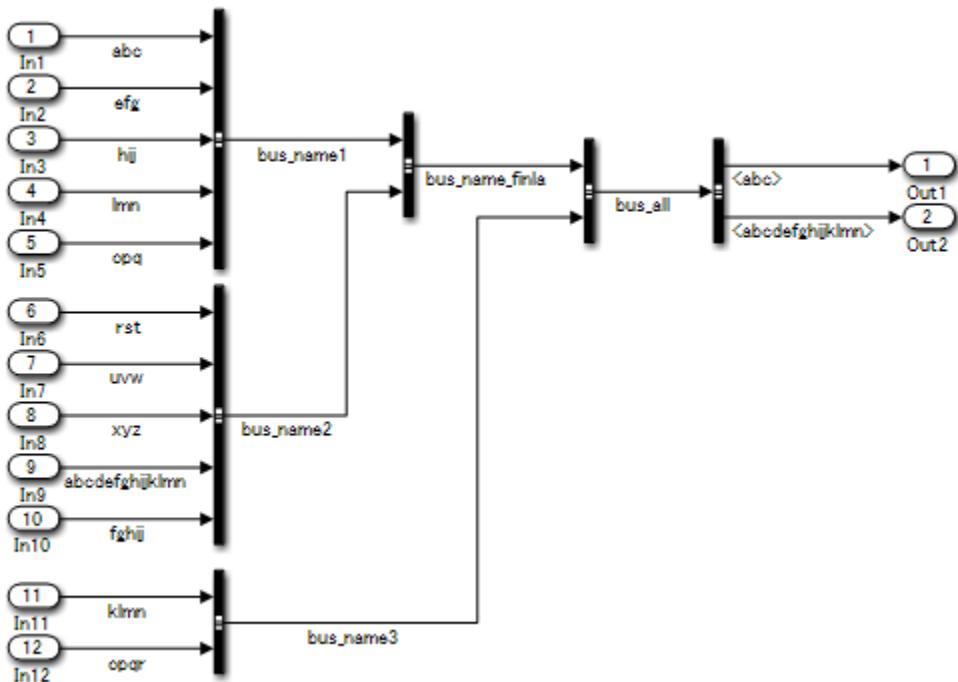
Maximum bus name length

Example — Correct

Signal name length is less than 63 characters.



The length of the hierarchical signal name (full path)
bus_all.bus_name_finla.bus_name2.abcdefghijklmn is less than or equal to 63 characters.



Rationale

Sub ID a:

- Code generation may not be possible.

Verification

Model Advisor check: "Check length of signal and bus names" (Simulink Check)

Last Changed

R2020a

See Also

- "Model Configuration Parameters"
- "Signal Basics"
- "Simulink Bus Capabilities"

Version History

Introduced in R2020a

jc_0246: Length restriction for parameter name

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Parameter name length shall be a maximum of 63 characters.

Custom Parameter

Maximum parameter name length

Rationale

Sub ID a:

- Code generation may not be possible.

Verification

Model Advisor check: “Check length of parameter names” (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0795: Usable characters for Stateflow data names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d
- JMAAB — a, b, c, d

MATLAB Versions

All

Rule

Sub ID a

Stateflow data {name} shall not use underscores at the beginning.

Custom Parameter

Not Applicable

Sub ID b

Stateflow data {name} shall not use underscores at the end.

Custom Parameter

Not Applicable

Sub ID c

Stateflow data {name} shall not use consecutive underscores.

Custom Parameter

Not Applicable

Sub ID d

Stateflow data {name} shall not consist solely of a single reserved MATLAB word.

Custom Parameter

Not Applicable

Rationale

Sub IDs a, b, c, d:

- Readability is impaired.
- Deviation from the rule may result in unintended code behavior.

Verification

Model Advisor check: “Check usable characters for Stateflow data names” (Simulink Check)

Last Changed

R2020a

See Also

- “Add Stateflow Data” (Stateflow)
- “Set Data Properties” (Stateflow)

Version History

Introduced in R2020a

jc_0796: Length restriction for Stateflow data names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d
- JMAAB — a, b, c, d

MATLAB Versions

All

Rule

Sub ID a

Stateflow data {name} shall be a maximum of 63 characters.

Custom Parameter

Stateflow data name character limit

Rationale

Sub ID a:

- Readability is impaired.
- Deviation from the rule can result in unintended code behavior

Verification

Model Advisor check: “Check length of Stateflow data name” (Simulink Check)

Last Changed

R2020a

See Also

- “Add Stateflow Data” (Stateflow)
- “Set Data Properties” (Stateflow)

Version History

Introduced in R2020a

jc_0791: Duplicate data name definitions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

Data name definitions shall not be duplicated in the base workspace and model workspace.

Custom Parameter

Not Applicable

Sub ID b

Data names shall not be duplicated in the base workspace and data dictionary (sldd).

Custom Parameter

Types of data dictionary

Sub ID c

Data name definitions shall not be duplicated in the model workspace and data dictionary (sldd).

Custom Parameter

Types of data dictionary

Rationale

Sub IDs a, b, c:

- Duplicated data name can cause unintended model behavior.

Verification

Model Advisor check: “Check duplication of Simulink data names” (Simulink Check)

Last Changed

R2020a

See Also

- “Set Data Properties” (Stateflow)
- “Add Stateflow Data” (Stateflow)
- “Model Reference Data Storage”

Version History

Introduced in R2020a

jc_0792: Unused Data

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

The data dictionary (`sldd`) shall define only the data that is used in the Simulink or Stateflow Coder™ model.

Custom Parameter

Types of data dictionary

Sub ID b

The model workspace shall define only the data that is used in the Simulink or Stateflow model.

Custom Parameter

Not Applicable

Rationale

Sub IDs a, b:

- Unused data can affect maintainability and operability.

Verification

Model Advisor check: "Check unused data in Simulink Model" (Simulink Check)

Last Changed

R2020a

See Also

- “Model Reference Data Storage”

Version History

Introduced in R2020a

jc_0700: Unused data in Stateflow block

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

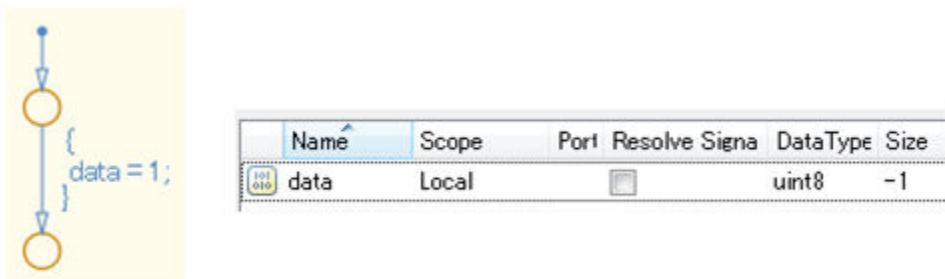
Sub ID a

Configuration parameter **Unused data, events, messages, and functions** shall be set to **Warning** or **Error** to prevent unused Stateflow data, events, and messages in the Stateflow block.

Custom Parameter

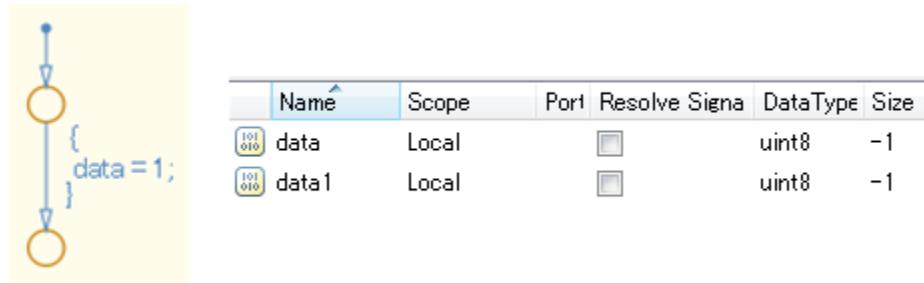
Not Applicable

Example — Correct



Example — Incorrect

Unused data is defined.



Rationale

Sub ID a:

- Unused data and events in the Stateflow block can affect maintainability and reusability.
- Affects code as a declarative statement concerning unused data is inserted into the generated code.

Verification

Model Advisor check: "Check for unused data in Stateflow Charts" (Simulink Check)

Last Changed

R2020a

See Also

- "Data, Events, and Messages" (Stateflow)
- "View Differences Between Stateflow Messages, Events, and Data" (Stateflow)
- "Manage Symbols in the Stateflow Editor" (Stateflow)

Version History

Introduced in R2020a

na_0019: Restricted variable names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

Reserved C variable names shall not be used as variable names in MATLAB code. For example, avoid using const, TRUE, FALSE, infinity, nil, double, single, or enum in MATLAB code.

Custom Parameter

Not Applicable

Sub ID b

Variable names that conflict with MATLAB functions, such as conv, shall not be used.

Custom Parameter

Not Applicable

Rationale

Sub IDs a, b:

- Improves readability of the code
- Code generation may not be possible.

Verification

Model Advisor check: "Check usage of restricted variable names" (Simulink Check)

Last Changed

R2020a

See Also

- “Reserved Keywords” (Embedded Coder)
- “Reserved Identifiers and Code Replacement” (Embedded Coder)
- “Integrate MATLAB Functions in a Stateflow Charts” (Stateflow)
- “Implement MATLAB Functions in Simulink with MATLAB Function Blocks”

Version History

Introduced in R2020a

Simulink

- “Configuration Parameters” on page 3-2
- “Diagram Appearance” on page 3-12
- “Signal” on page 3-68
- “Conditional Subsystem Relations” on page 3-103
- “Operation Blocks” on page 3-125
- “Other Blocks” on page 3-174

Configuration Parameters

jc_0011: Optimization parameters for Boolean data types

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Configuration parameter **Implement logic signals as Boolean data (vs. double)** shall be selected so that optimization parameters are activated for logic signals.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Using Boolean data can reduce RAM capacity when using C code.

Verification

Model Advisor check: "Check Implement logic signals as Boolean data (vs. double)" (Simulink Check)

Last Changed

R2020a

See Also

- “Signal Basics”

Version History

Introduced in R2020a

jc_0642: Integer rounding mode setting

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

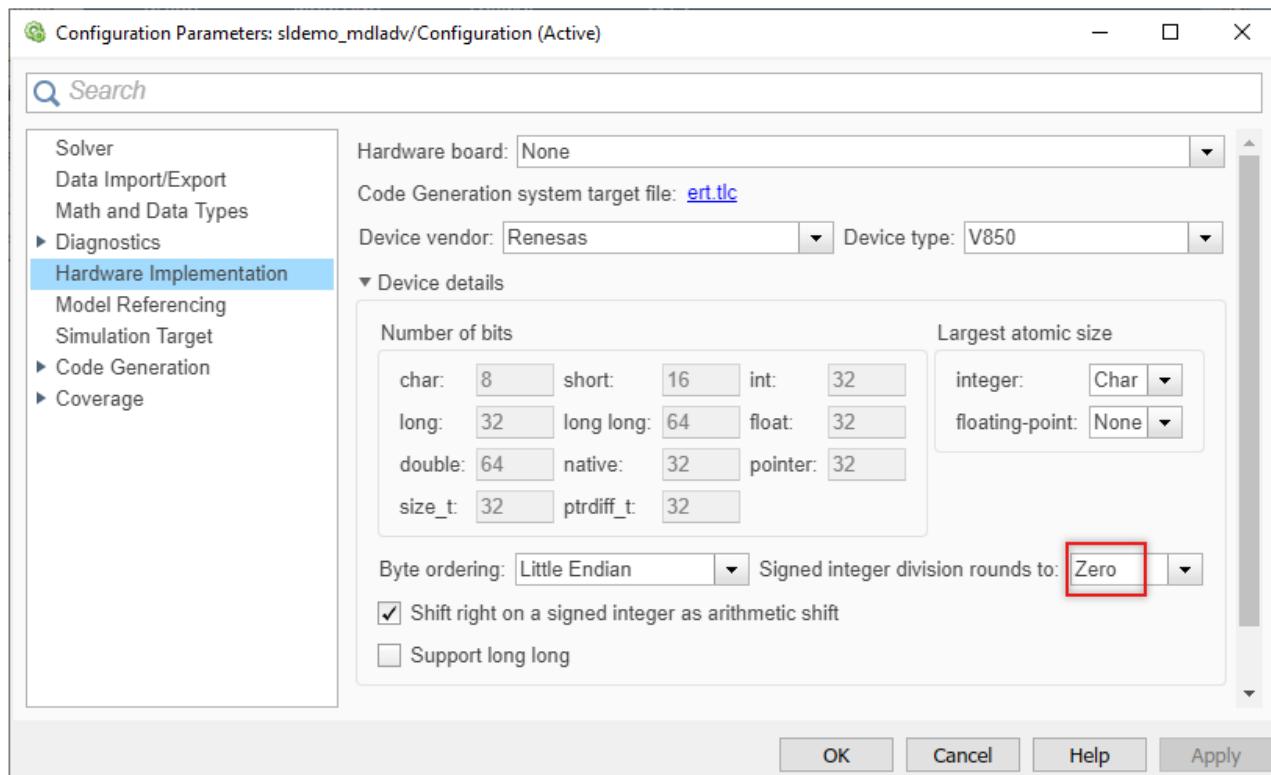
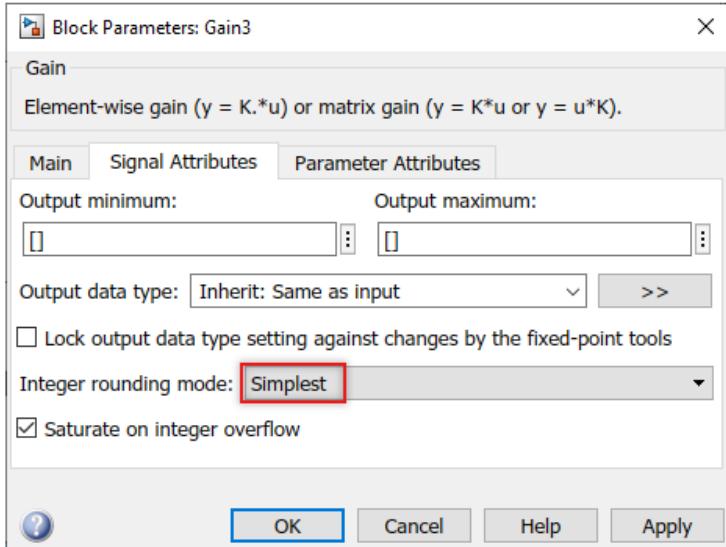
When block parameter **Integer rounding mode** is set to **Simplest**, configuration parameter **Production hardware signed integer division rounds to** shall be set to **Floor or Zero**.

Custom Parameter

Not Applicable

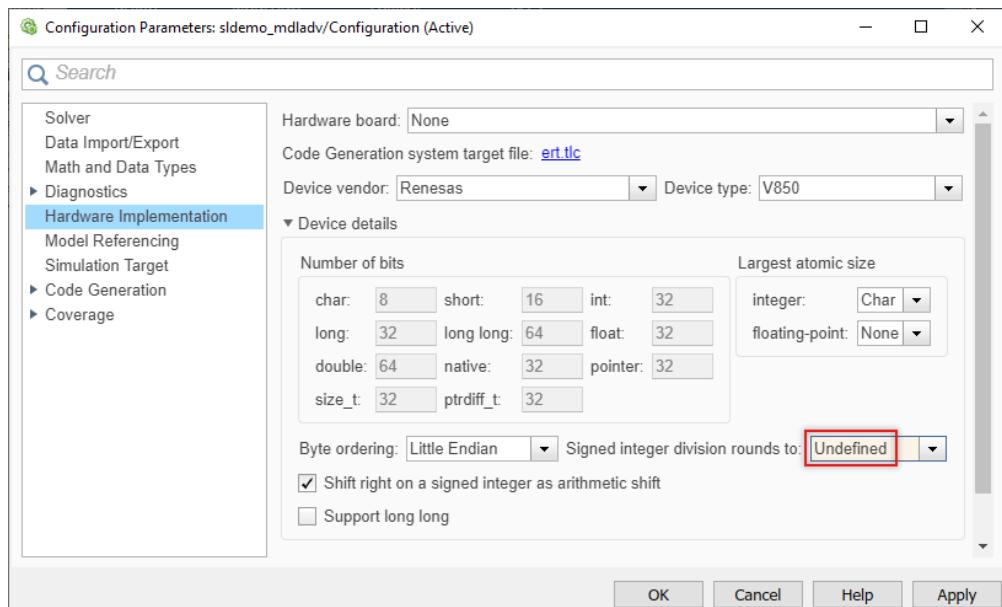
Example — Correct

Block parameter **Integer rounding mode** is set to **Simplest** and configuration parameter **Production hardware signed integer division rounds to** is set to **Zero**.



Example — Incorrect

Configuration parameter **Production hardware signed integer division rounds to** is set to **Undefined** when block parameter **Integer rounding mode** is set to **Simplest**.



Rationale

Sub ID a:

- Prevents unintended rounding of divided signed integers.

Verification

Model Advisor check: "Check Signed Integer Division Rounding mode" (Simulink Check)

Last Changed

R2024b

See Also

- Signed integer division rounds to

Version History

Introduced in R2020a

jc_0806: Detecting incorrect calculation results

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

Configuration parameter **Division by singular matrix** shall be set to Error.

Custom Parameter

Not Applicable

Sub ID b

Configuration parameter **Inf or NaN block output** shall be set to Error.

Custom Parameter

Not Applicable

Sub ID c

(R2014b and later) These configuration parameters shall be set to Error:

- **Wrap on overflow**
- **Saturate on overflow**

(R2010b to R2014a) Configuration parameter **Detect overflow** shall be set to Error.

Custom Parameter

Not Applicable

Rationale

Sub IDs a, b, c:

- Allows detection of operations with invalid values.

Verification

Model Advisor check: "Check diagnostic settings for incorrect calculation results" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0021: Model diagnostic settings

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

These configuration parameters shall be set to warning or error:

- Algebraic loop
- Minimize algebraic loop
- “Multitask data transfer”
- “Inf or NaN block output”
- “Duplicate data store names”
- Unconnected block input ports
- Unconnected block output ports
- Unconnected line
- Unspecified bus object at root Outport block
- Element name mismatch
- (R2017a and earlier) Mux blocks used to create bus signals
- (R2012a and earlier) Invalid function-call connection

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Improves model workflow.
- Code generation may not be possible

Verification

Model Advisor check “Check model diagnostic parameters” (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

Diagram Appearance

na_0004: Simulink model appearance settings

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Simulink model appearance settings shall conform with the project settings.

Custom Parameter

Display option

Example — View Options

On the Simulink toolbar, these parameters are available in the **Modeling** tab, under **Environment**.

Parameter	Setting
Model Browser	Deselect Model Browser .
Status Bar	Select Status Bar .
Toolbar	Select Toolbar .
Zoom	Select Zoom (Normal View 100%) .

Example — Block Display Options

On the Simulink toolbar, these parameters are available in the **Debug** tab, under **Information Overlays**.

Parameter	Setting
Library Links	Select Hide All Links .
Linearization Indicators	Select Linearization Indicators .

Parameter	Setting
Ref. Model I/O Mismatch	Deselect Ref. Model I/O Mismatch .
Ref. Model Version	Deselect Ref. Model Version .
Sample Time Colors	Deselect Colors .
Execution Order	Deselect Execution Order .

Example — Block Color Options

On the Simulink toolbar, these parameters are available in the **Format** tab.

Parameter	Setting
Background color	Select white option from the Background list.
Foreground color	Select black option from the Foreground list.

Example — Storage Class Indicators

To access the parameter below, On the Simulink toolbar, open Simulink Coder from Apps, select **C Code** tab and choose **Code Interface**.

Parameter	Setting
Storage Class Indicator	Deselect Storage Class Indicator .

Example — Signal Display Options

On the Simulink toolbar, these parameters are available in the **Debug** tab, under **Information Overlays**.

Parameter	Setting
Base Data Types	Deselect Base Data Types .
Alias Data Types	Deselect Alias Data Types .
Signal Dimensions	Deselect Signal Dimensions .
Testpoint	Select Test Point .
Logging and Viewer indicators	Select Logging & Viewers .
Nonscalar Signals	Select Nonscalar Signals .

Rationale

Sub ID a:

- Standard model appearance improves readability.

Verification

Model Advisor check: "Check for Simulink diagrams using nonstandard display attributes" (Simulink Check)

Last Changed

R2020a

See Also

- MISRA AC SLSF 023A
- “Mapping from Simulink Editor to the Simulink Toolstrip”

Version History

Introduced in R2020a

db_0043: Model font and font size

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d
- JMAAB — a, b, c, d

MATLAB Versions

All

Rule

Sub ID a

Block name *font* and *font style* shall conform with the project settings.

Signal name *font* and *font style* shall conform with the project settings.

Custom Parameter

Font

Font style

Sub ID b

Block name *font size* shall conform with the project settings.

Signal name *font size* shall conform with the project settings

Custom Parameter

Font size

Sub ID c

State labels and box name *font* and *font style* shall conform with the project settings.

Transition labels and comment *font* and *font style* shall conform with the project settings.

Custom Parameter

Font

Font style

Sub ID d

State labels and box name *font size* shall conform with the project settings.

Transition labels and comment *font size* shall conform with the project settings.

Custom Parameter

Font size

Rationale

Sub IDs a, c:

- Standard fonts improve readability.

Sub IDs b, d:

- Standard font size improves readability.

Verification

Model Advisor check: "Check model font settings" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jm_0002: Block resizing

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

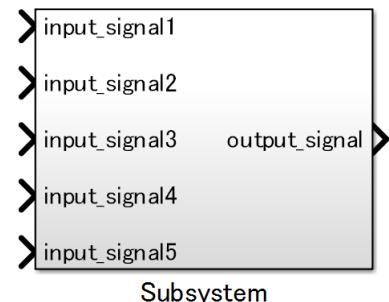
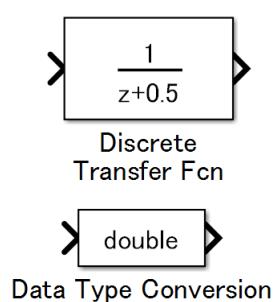
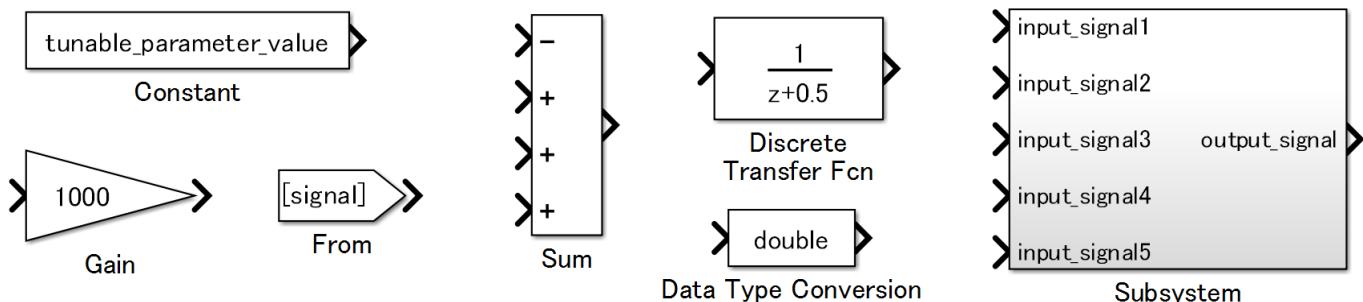
Blocks shall be sized so that the block icon is visible and recognizable.

Custom Parameter

Not Applicable

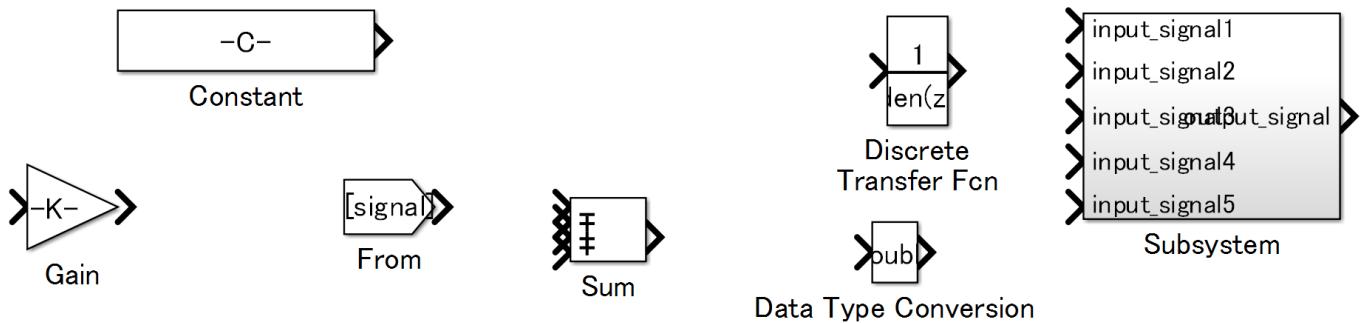
Example — Correct

The block icon is visible and recognizable.



Example — Incorrect

The block is too small so the icon is neither visible nor recognizable.



Rationale

Sub ID a:

- When a block is too small, the text and symbol displayed by the icon can be difficult to see, which impairs readability.

Verification

Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

See Also

- "Configure Model Layout"

Version History

Introduced in R2020a

db_0142: Position of block names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

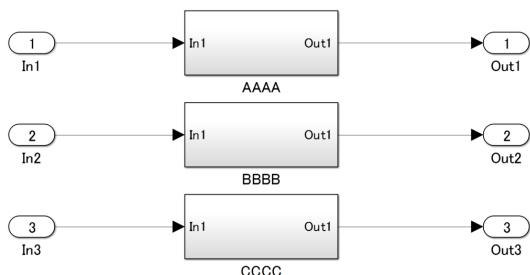
Sub ID a

The block name shall be positioned below the block.

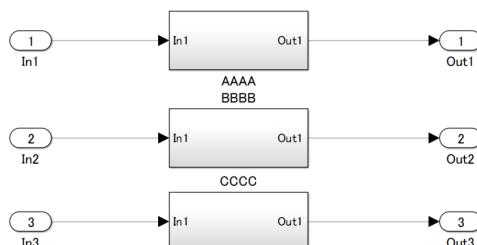
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- Consistent placement of the block name improves model readability because it is easier to determine which name corresponds to the block.

Verification

Model Advisor check: "Check whether block names appear below blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0061: Display of block names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Block names shall be hidden for blocks that meet either of these criteria:

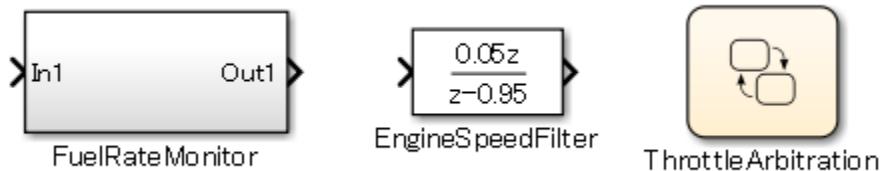
- Block type is evident from its visual appearance
- Uses the default block name (including instances where only a number has been added at the end)

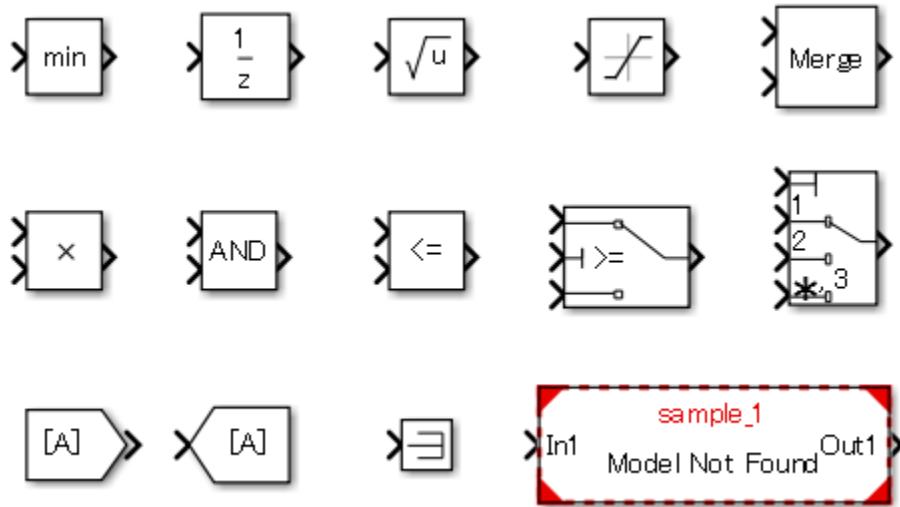
For blocks that do not meet the criteria, their name shall be displayed.

Custom Parameter

Blocks with a clear type due their appearance

Example — Displayed block names



Example — Hidden block names**Rationale**

Sub ID a:

- Improves model readability.

Verification

Model Advisor check: "Check the display attributes of block names" (Simulink Check)

Last Changed

R2020a

See Also

- MISRA AC SLSF 026A

Version History

Introduced in R2020a

db_0140: Display of block parameters

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

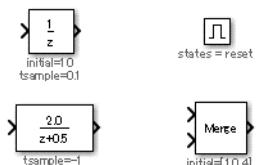
Sub ID a

Block annotation shall display block parameters that are defined by the project.

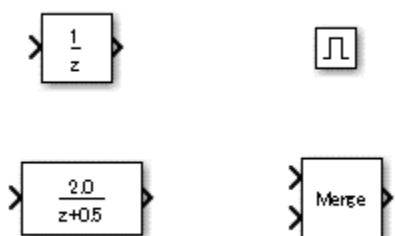
Custom Parameter

Block parameters

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- Readability improves when block parameters are displayed.

Verification

Model Advisor check: "Check for display of block parameter" (Simulink Check)

Last Changed

R2020a

See Also

- MISRA AC SLSF 026E
- "Set Block Annotation Properties"

Version History

Introduced in R2020a

jc_0603: Model description

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

The model layer shall include a description of the layer.

Layers that require a description are defined (by function and layer type) in the project.

Custom Parameter

Description object (block type, etc.)

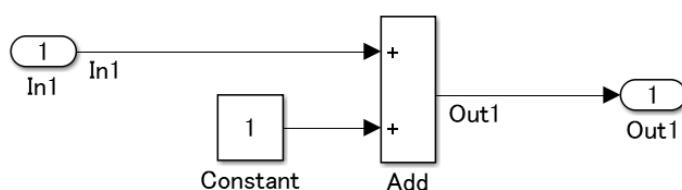
Layer being described

Example — Correct

Model layer includes description [Requirement].

【Requirement】

Performs an increment process on the input signal In1



Sub ID b

The format of the layer description shall be consistent in the model.

Custom Parameter

Model description format

Rationale

Sub ID a:

- When a description is not included, the readability of the control specifications is reduced. Usability, maintainability, and portability also decreases.

Sub ID b:

- Readability is impaired when the description format is not consistent.

Verification

Model Advisor check: "Check Model Description" (Simulink Check)

Last Changed

R2020a

See Also

- Sub ID a and b, see MISRA AC SLSF 022

Version History

Introduced in R2020a

jc_0604: Using block shadow

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

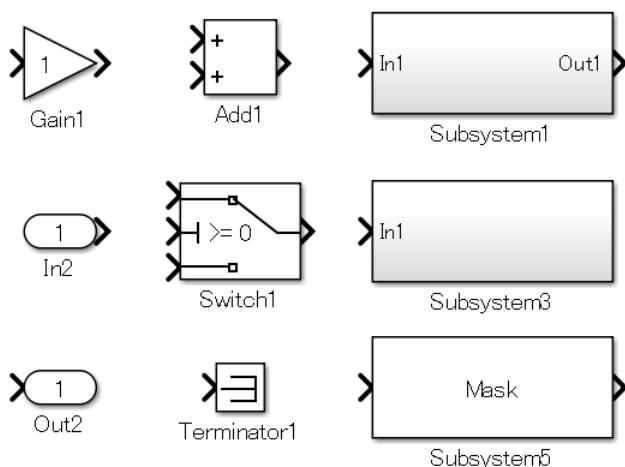
Sub ID a

Block format property **Shadow** (DropShadow) shall not be selected.

Custom Parameter

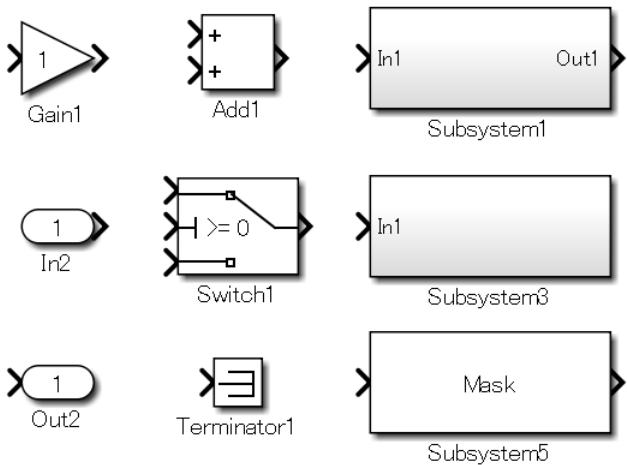
Not Applicable

Example — Correct



Example — Incorrect

Blocks have a drop shadow.



Rationale

Sub ID a:

- Difficult to determine if a port exists because it is hidden by the shading, which impairs readability.

Verification

Model Advisor check: "Check if blocks are shaded in the model" (Simulink Check)

Last Changed

R2020a

See Also

- "Common Block Properties"
- "Set Block Parameter Values"

Version History

Introduced in R2020a

db_0081: Unconnected signals and blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

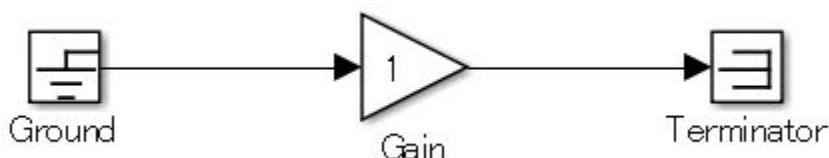
Sub ID a

The model shall not have signal lines that are not connected.

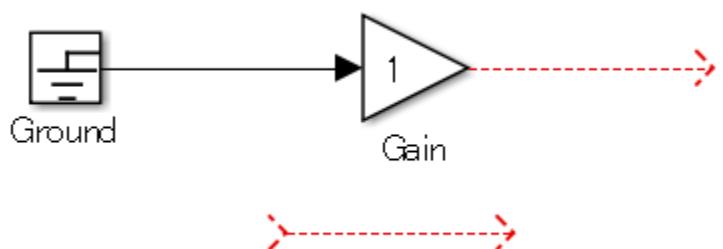
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



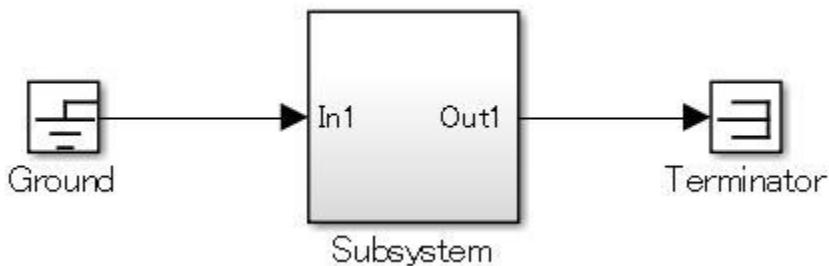
Sub ID b

The model shall not have subsystems or basic blocks that are not connected.

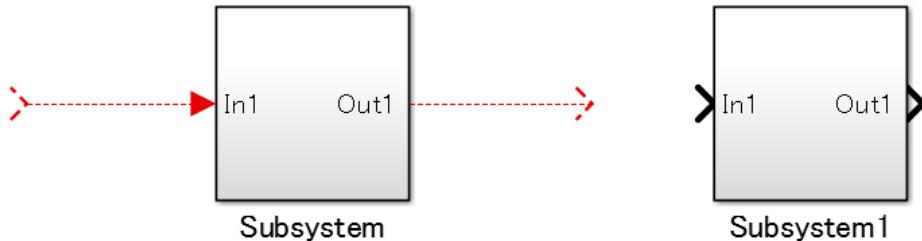
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub IDs a, b:

- Unconnected lines can have adverse effects, such as simulation errors or failure to generate code.

Verification

Model Advisor check: "Check for unconnected signal lines and blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Signal Basics”
- “Explore Types of Subsystems”

Version History

Introduced in R2020a

db_0032: Signal line connections

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c
- JMAAB — a, b, c

MATLAB Versions

All

Rule

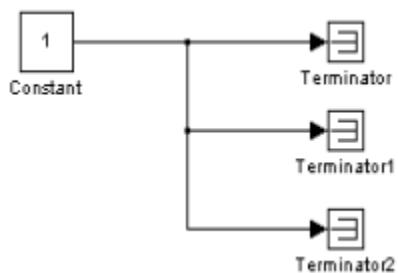
Sub ID a

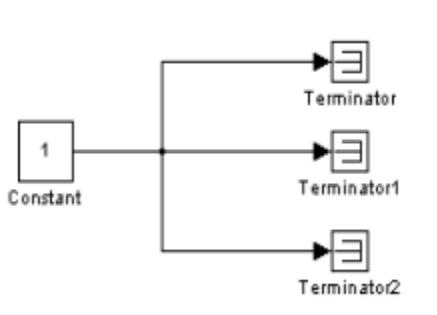
Signal lines shall not split into more than two sub lines at a single branching point.

Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect**Sub ID b**

Signal lines shall be resized vertically or horizontally as required for the model layout.

Custom Parameter

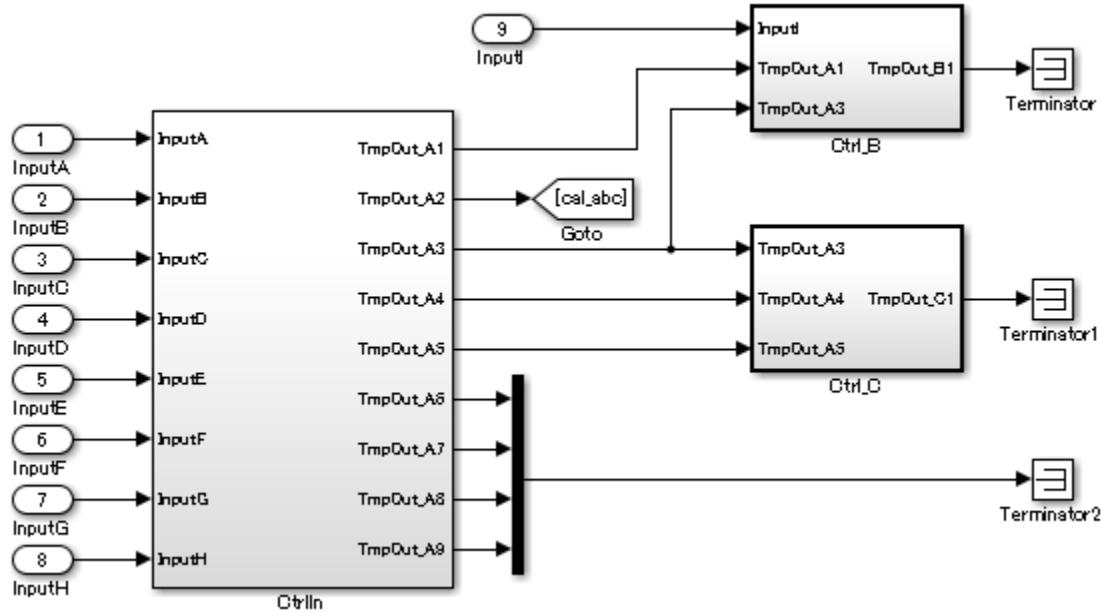
Not Applicable

Sub ID c

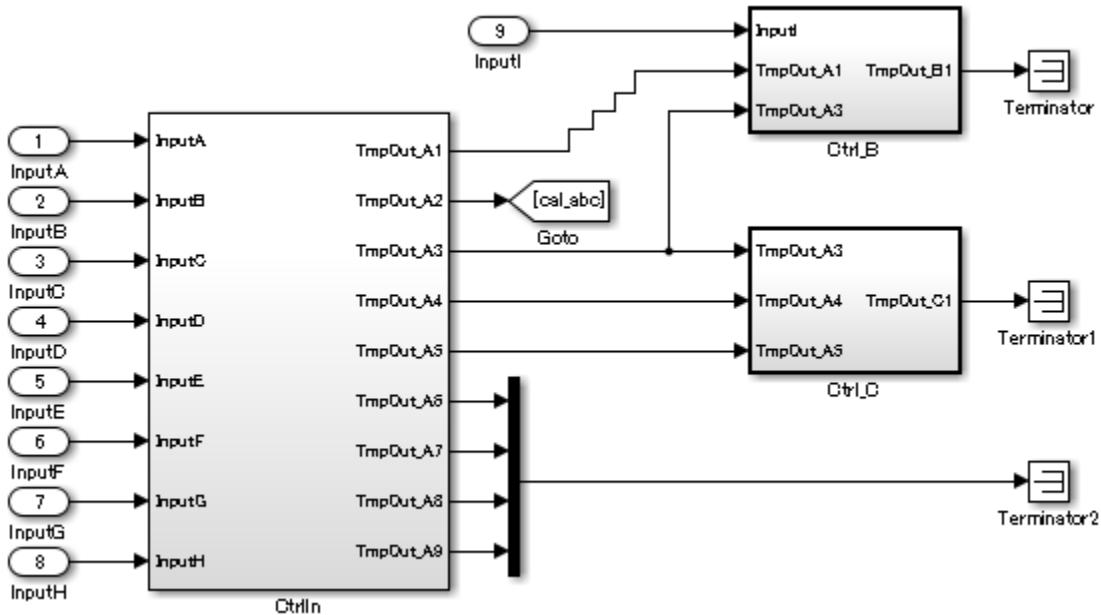
Signal lines shall not bend multiple times unnecessarily.

Custom Parameter

Not Applicable

Example — Correct

Example — Incorrect



Rationale

Sub ID a:

- Difficult to understand the relationships between blocks.

Sub ID b:

- Consistent application of signal lines improves readability.

Sub ID c:

- Deviation from the rules can impair readability.

Verification

Model Advisor check: "Check signal line connections" (Simulink Check)

Last Changed

R2024b

See Also

- “Signal Basics”

Version History

Introduced in R2020a

db_0141: Signal flow in Simulink models

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

Signals shall flow from left to right.

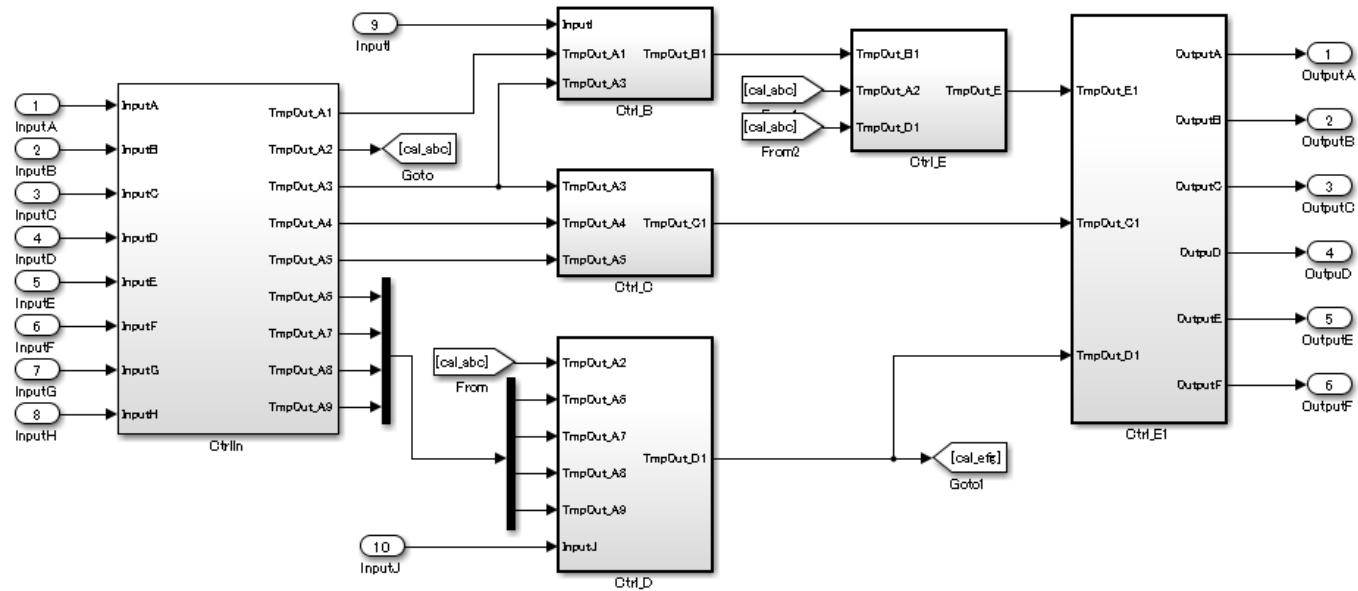
Exception

Feedback loops can flow from right to left.

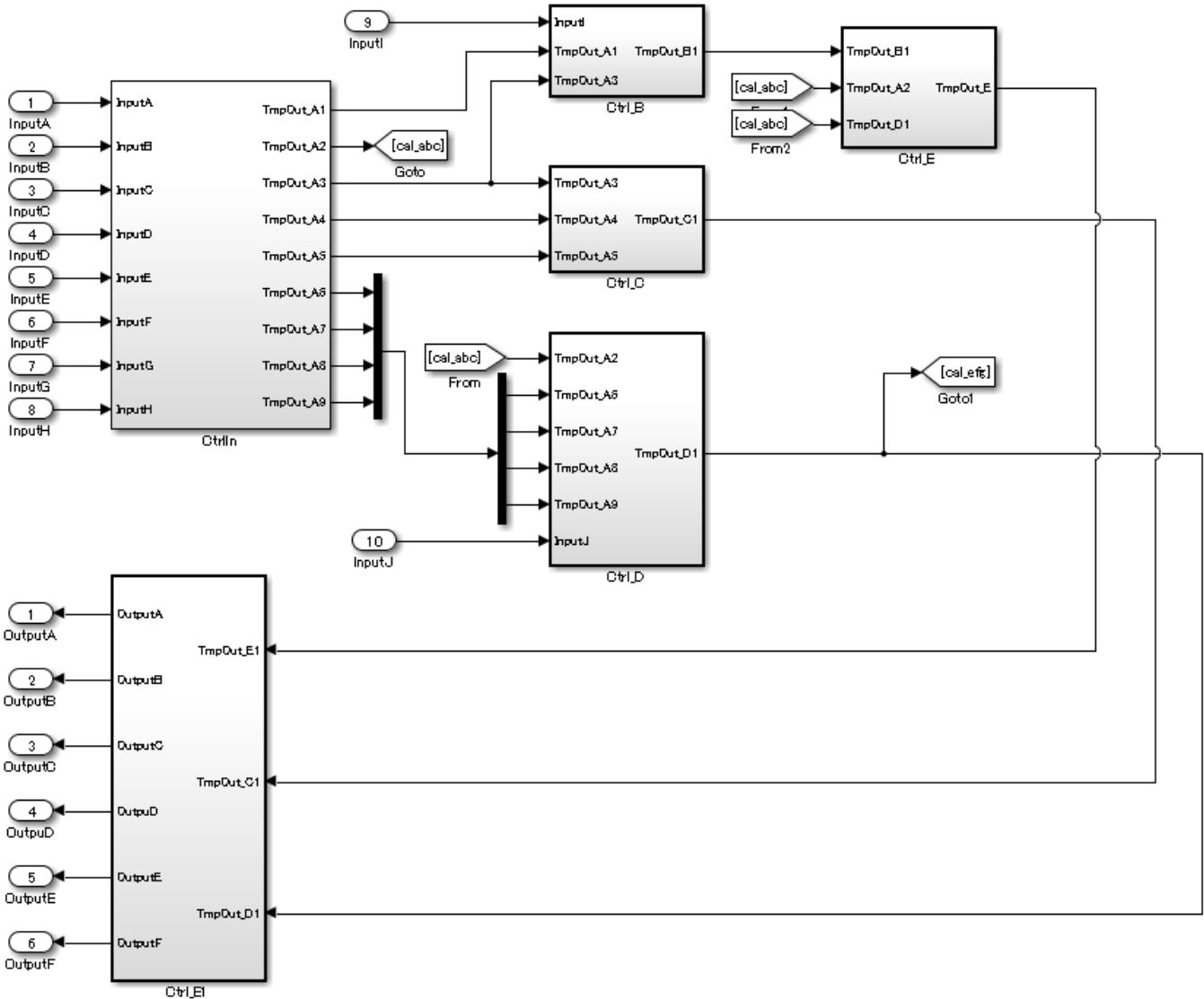
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



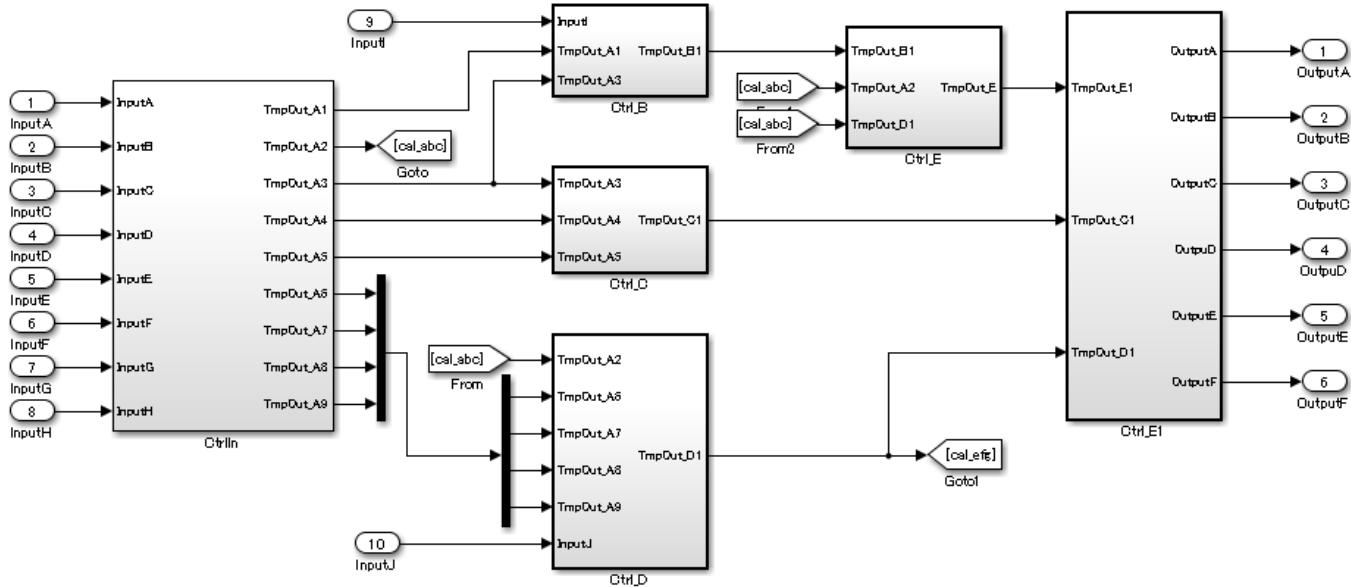
Sub ID b

Parallel blocks or subsystems shall be arranged from top to bottom.

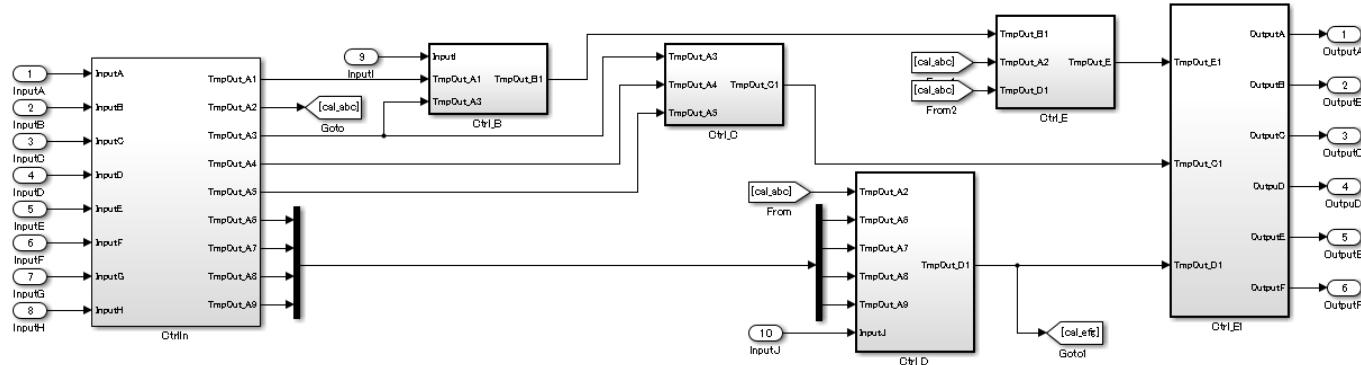
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub IDs a, b:

- Deviation from the rules can impair readability.

Verification

Model Advisor check: "Check signal flow in model" (Simulink Check)

Last Changed

R2024b

See Also

- “Signal Basics”
- “Explore Types of Subsystems”

Version History

Introduced in R2020a

jc_0110: Direction of block

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Blocks shall be arranged so the output is to the right.

Exception

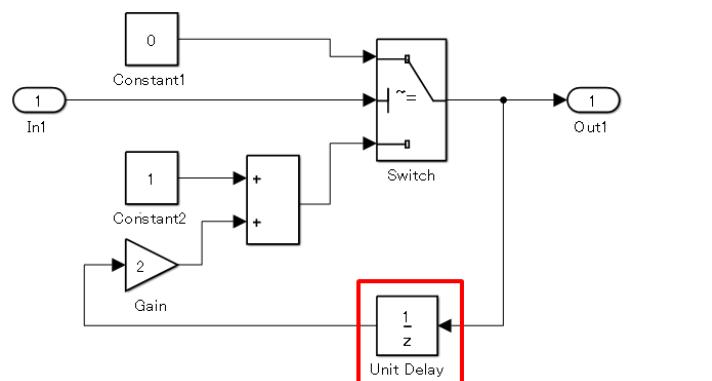
When a Delay is used in a feedback loop, the output can be to the left.

Custom Parameter

Not Applicable

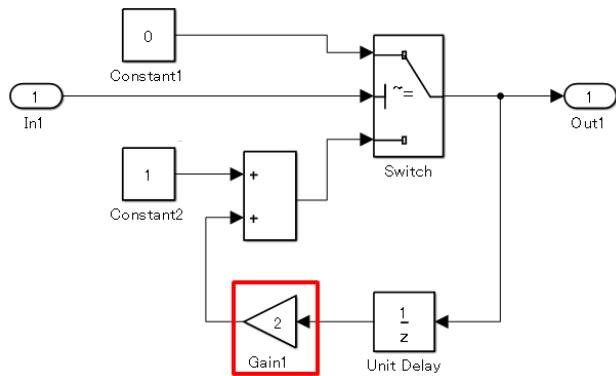
Example — Correct

The model is arranged so that the output is to the right. The output of the Delay block is to the left.



Example — Incorrect

The block is arranged so the output is to the left.



Rationale

Sub ID a:

- Signal flow can be difficult to understand if the direction of the signals is not consistent.

Verification

Model Advisor check: "Check block orientation" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0171: Clarification of connections between structural subsystems

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

A minimum of one signal line shall connect two structural subsystems.

When a two-way signal connection exists between two structural subsystems (A and B), each direction shall be connected to at least one signal line.

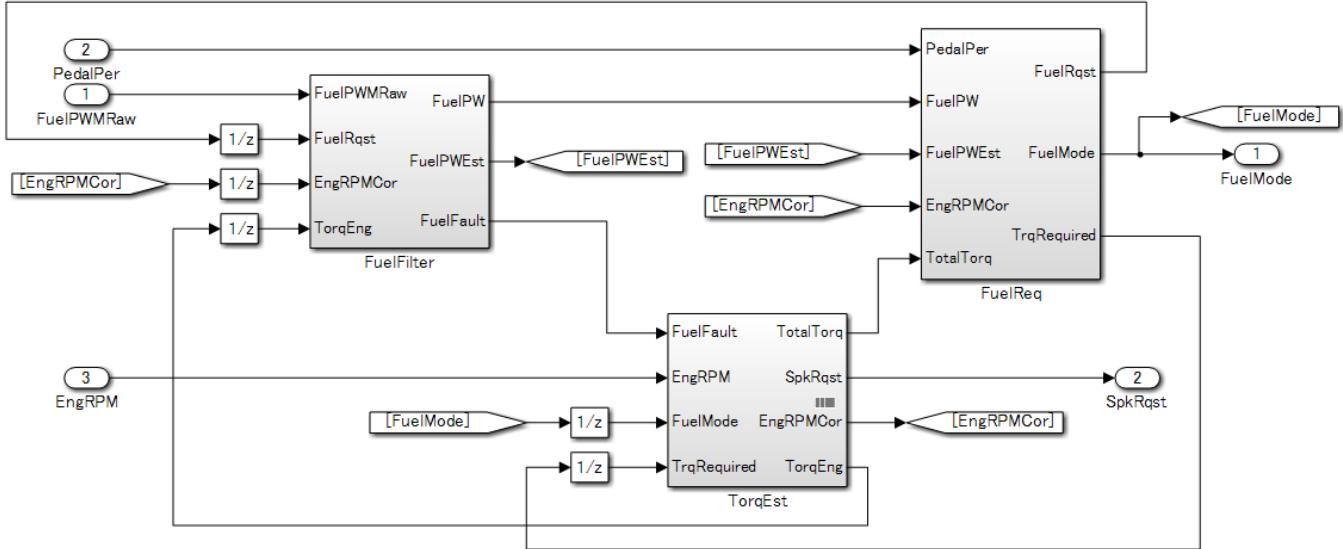
Exception

Using Goto and From blocks to create buses or connect signals to a Merge block.

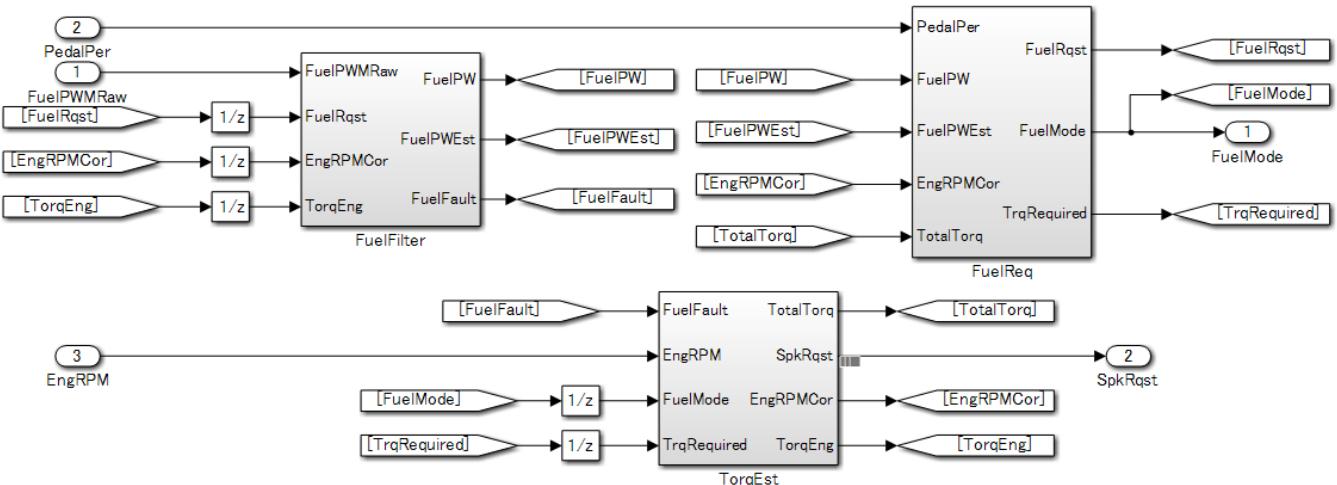
Custom Parameter

Not Applicable

Example – Correct



Example – Incorrect

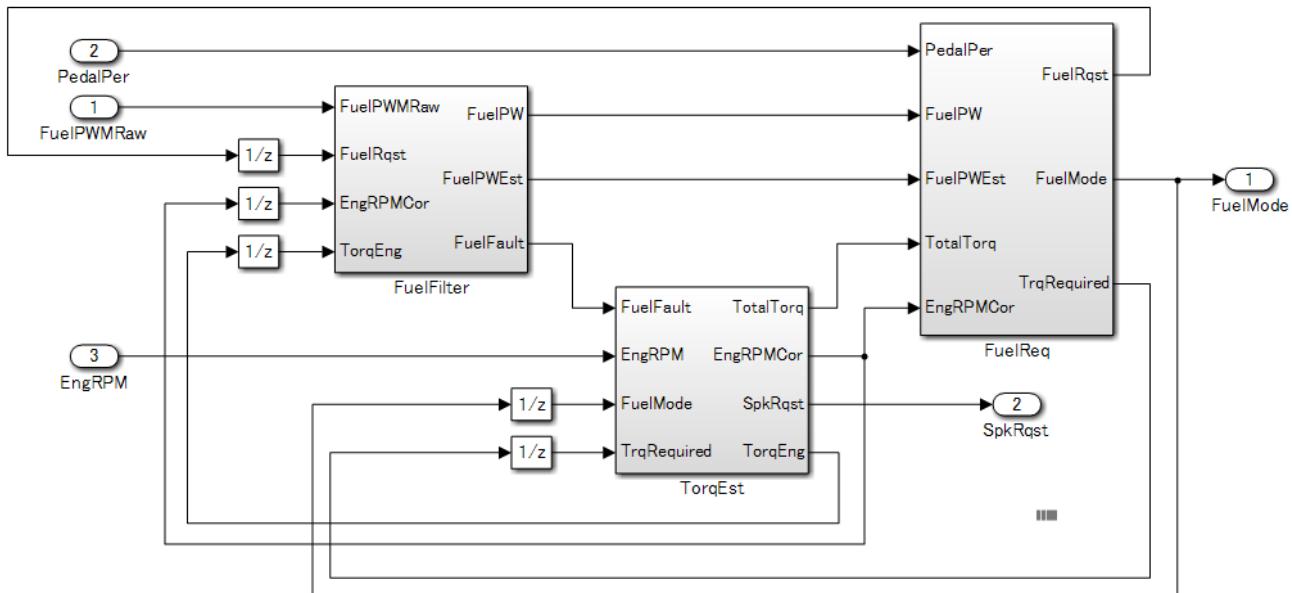


Sub ID b

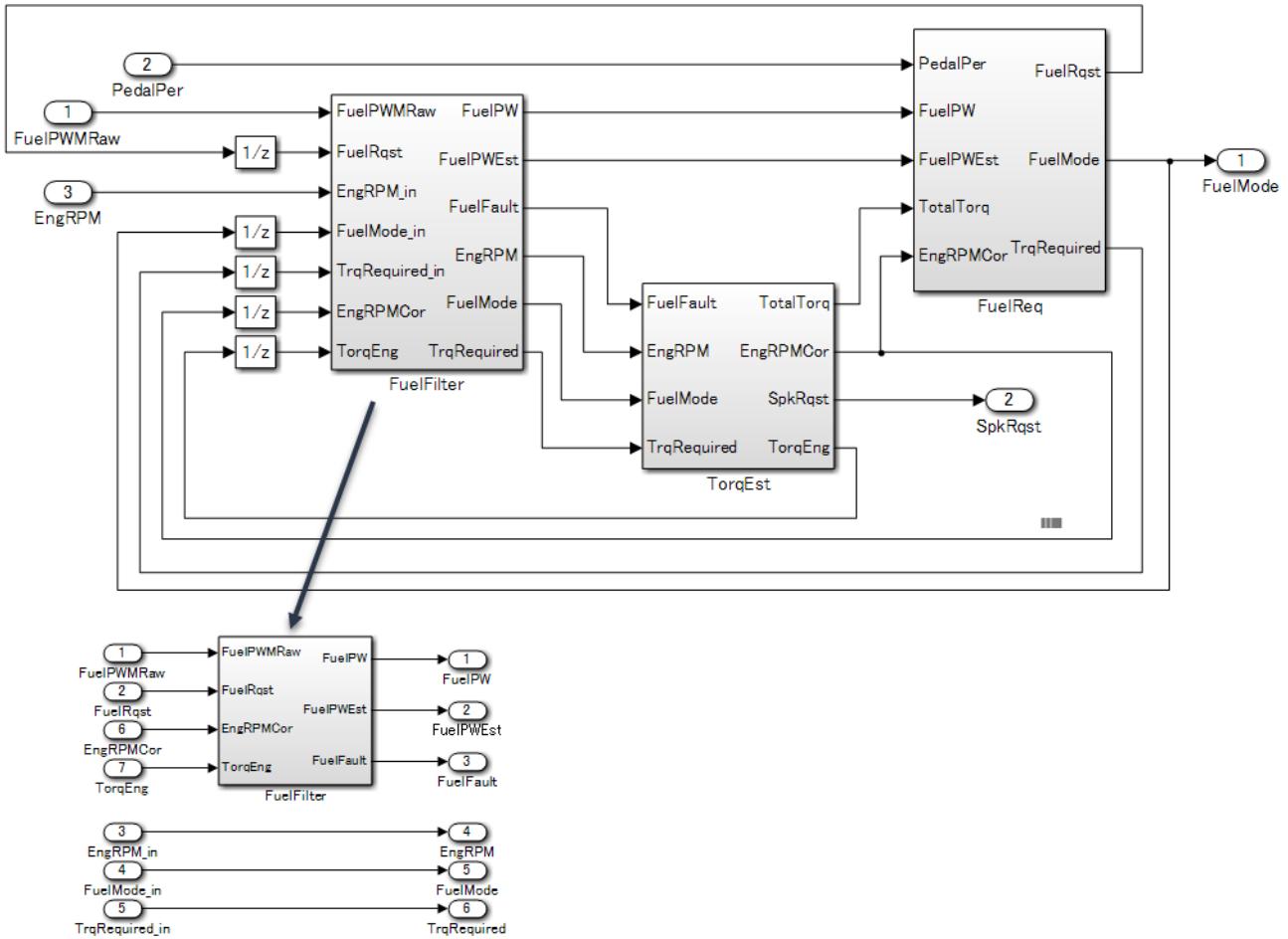
Signals that are not used within a structural subsystem shall be input to a structural subsystem. These signals shall not be output to other structural subsystems or basic blocks.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Signals that are not used in the subsystem are connected to avoid crossing of signal lines.



Rationale

Sub ID a:

- Clarifies structural subsystem connections and execution order.

Sub ID b:

- Eliminating unnecessary connections clarifies the relationship between connections.
- Deviation from the rule can cause confusion due to unused input/output signals.

Verification

Model Advisor check: "Check connections between structural subsystems" (Simulink Check)

Last Changed

R2020a

See Also

- “Signal Basics”
- “Explore Types of Subsystems”

Version History

Introduced in R2020a

jc_0602: Consistency in model element names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

These names shall match when they are directly connected by using signal lines.

- Import block name
- Outport block name
- Structural subsystem input port label name
- Structural subsystem output port label name
- From block tag name
- Goto block tag name
- Signal line signal name

Exceptions

A signal line that connects to one of the following subsystem types can have a name that differs from that of the subsystem port label:

- Subsystems linked to a library
- Reusable subsystems

When a combination of Import, Outport, and other blocks have the same name, use a suffix or prefix for the Import and Outport blocks. Any prefix or suffix can be used for ports, but they must be consistent. For example, the Import block uses “in” and Outport block uses “out”.

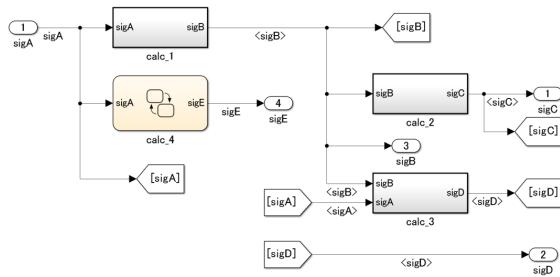
Note Import and Outport blocks must have different names and signal names.

Custom Parameter

Not Applicable

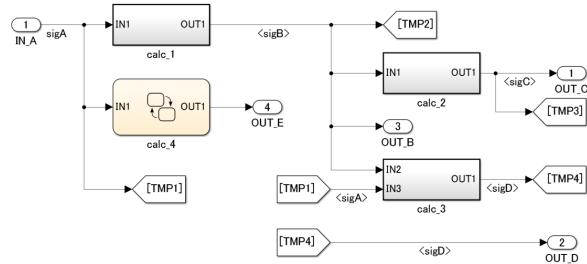
Example — Correct

Names of model elements that connect directly to signal lines are consistent.



Example — Incorrect

Inconsistent names for model elements that connect directly to signal lines.



Rationale

Sub ID a:

- Prevent misconnected signal lines.
- Readability is impaired.
- Deviation from the rule can make it difficult to maintain the integrity of the model and code.

Verification

Model Advisor check: "Check for consistency in model element names" (Simulink Check)

Last Changed

R2020a

See Also

- “Manage Signal Lines”

- “Explore Types of Subsystems”

Version History

Introduced in R2020a

jc_0281: Trigger signal names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a1/a2/a3/a4, b1/b2/b3/b4

MATLAB Versions

All

Rule

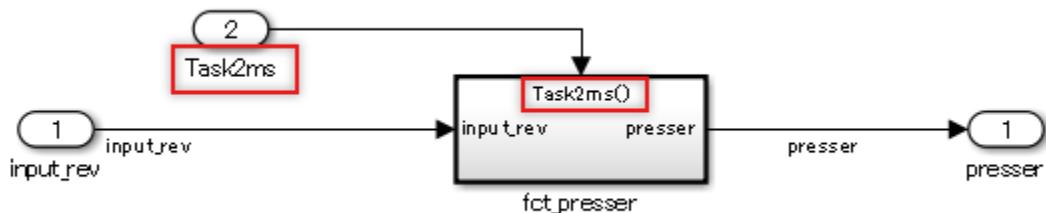
Sub ID a1

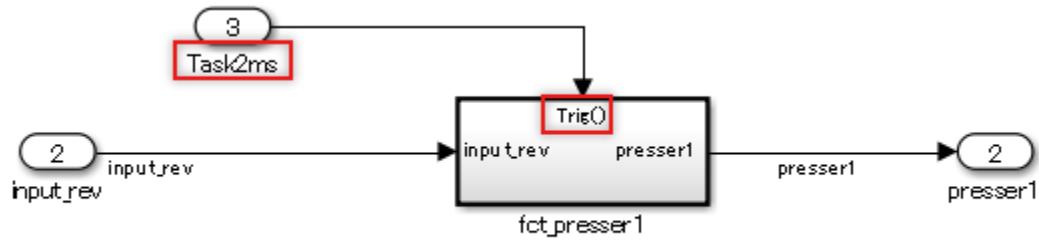
The name of the conditional input block at the destination shall include the name of the block at the origin of the trigger signal

Custom Parameter

Not Applicable

Example — Correct

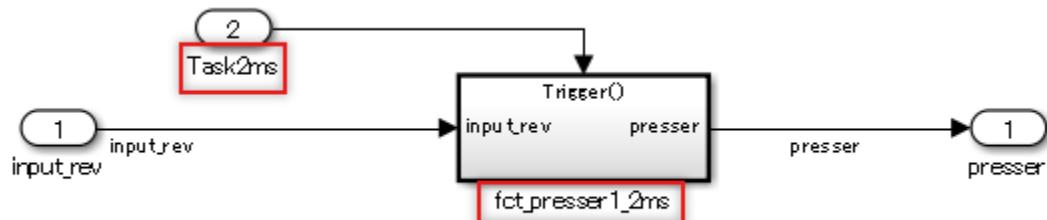
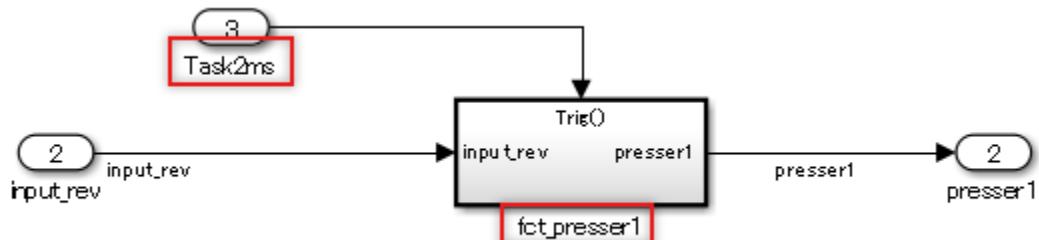


Example — Incorrect**Sub ID a2**

The name of the conditional subsystem at the destination shall include the name of the block at the origin of the trigger signal.

Custom Parameter

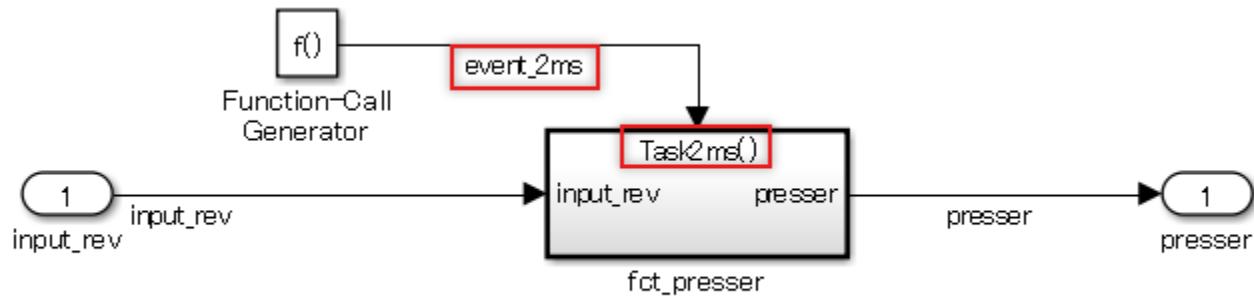
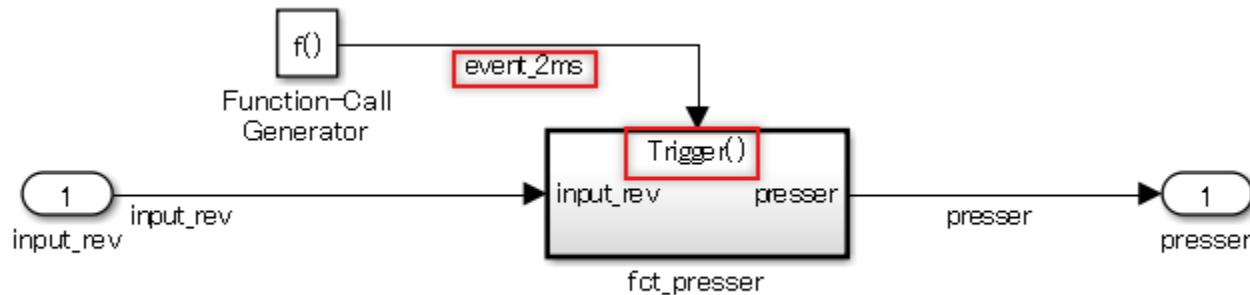
Not Applicable

Example — Correct**Example — Incorrect****Sub ID a3**

The name of the conditional input block at the destination shall include the name of the trigger signal.

Custom Parameter

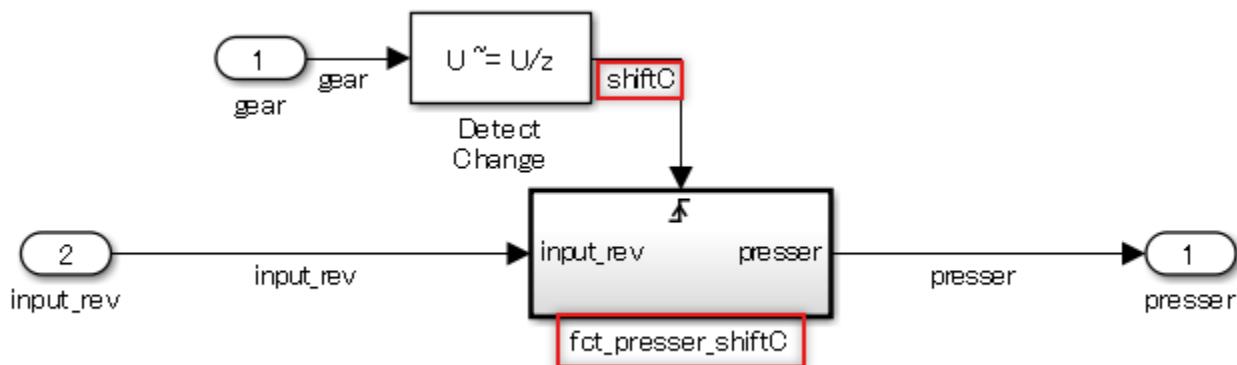
Not Applicable

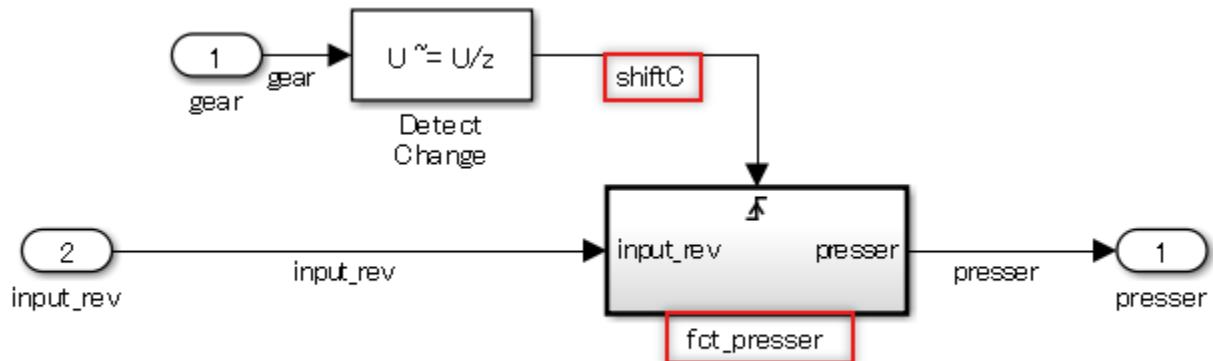
Example — Correct**Example — Incorrect****Sub ID a4**

The name of the conditional subsystem at the destination shall include the name of the trigger signal.

Custom Parameter

Not Applicable

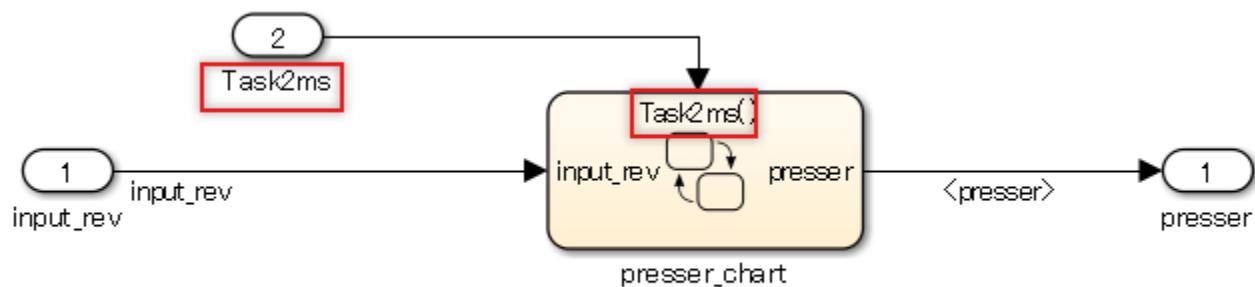
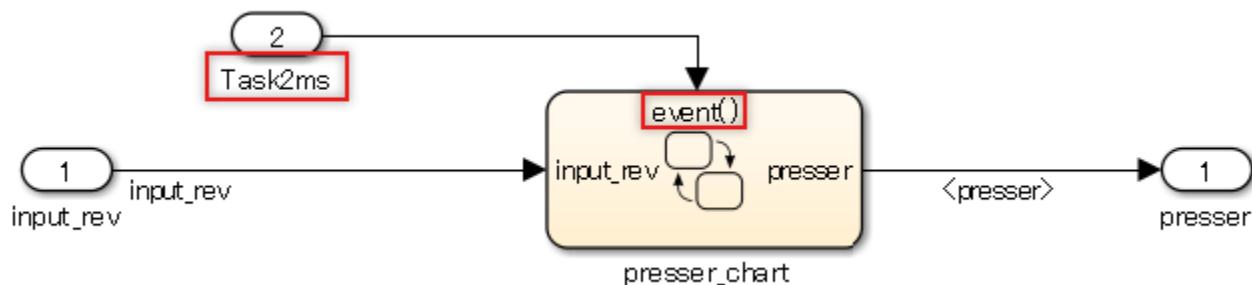
Example — Correct

Example — Incorrect**Sub ID b1**

The name of the Stateflow block event at the destination shall include the name of the block at the origin of the trigger signal.

Custom Parameter

Not Applicable

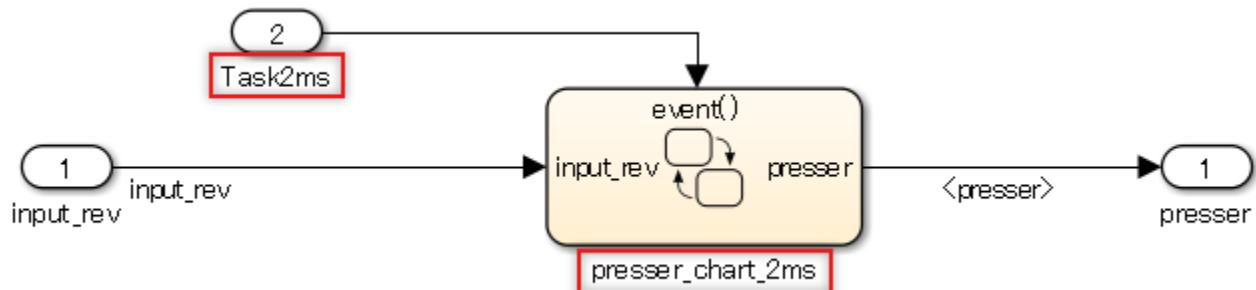
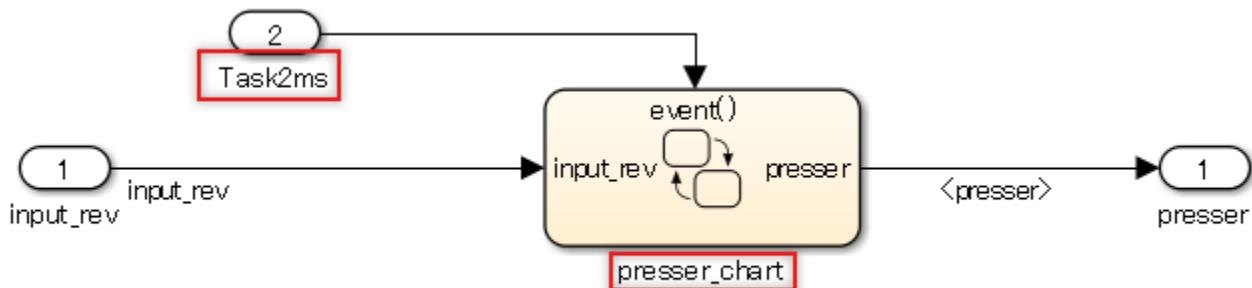
Example — Correct**Example — Incorrect**

Sub ID b2

The name of Stateflow Chart at the destination shall include the name of the block at the origin of the trigger signal.

Custom Parameter

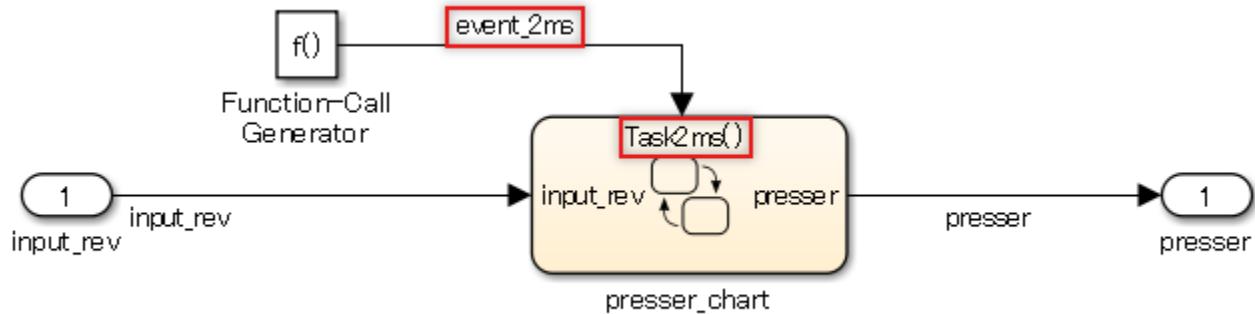
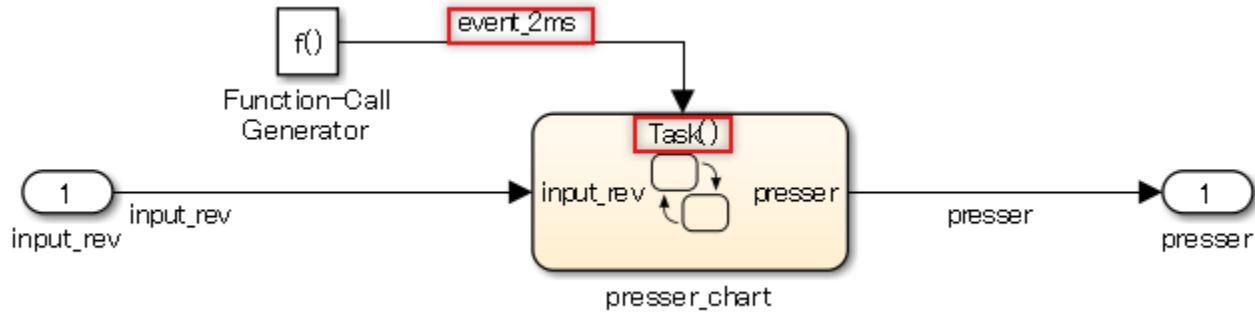
Not Applicable

Example — Correct**Example — Incorrect****Sub ID b3**

The name of the Stateflow block event at the destination shall include the name of the trigger signal.

Custom Parameter

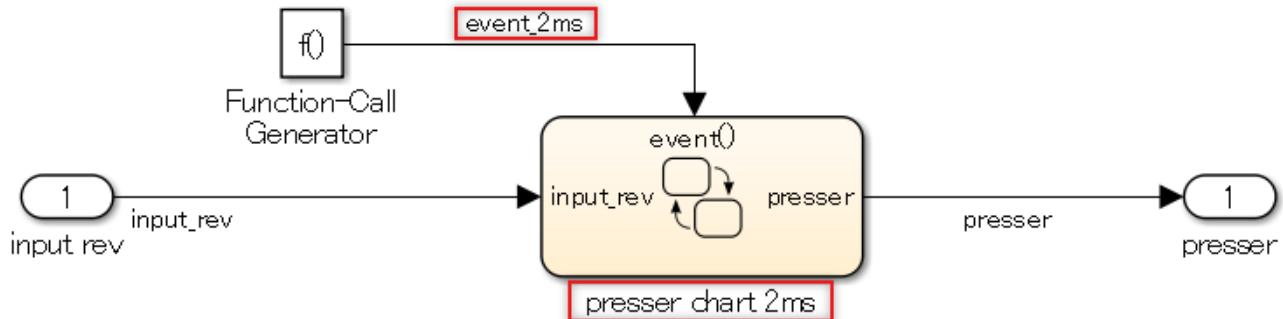
Not Applicable

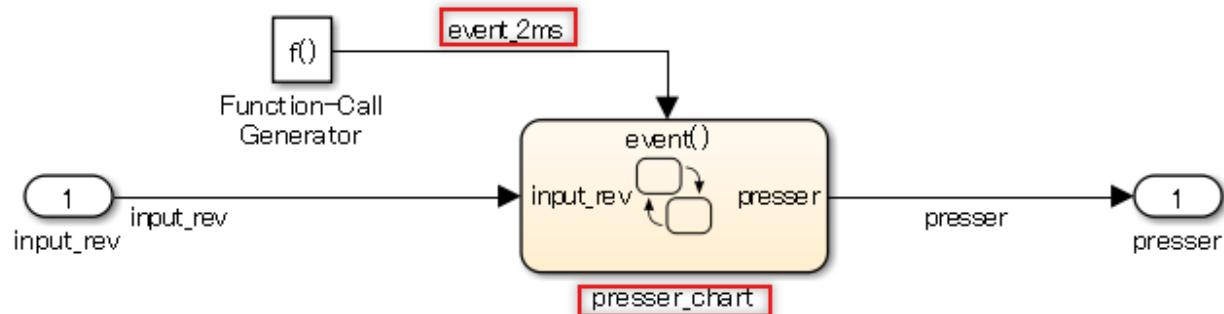
Example — Correct**Example — Incorrect****Sub ID b4**

The name of the trigger signal and the Stateflow Chart name at the destination must include the same name. The name of the Chart block at the destination shall include the name of the trigger signal.

Custom Parameter

Not Applicable

Example — Correct

Example — Incorrect**Rationale**

Sub ID a1, a2, a3, a4, b1, b2, b3, b4:

- Reduces connection mistakes.
- Increases understanding of the relationship between the origin of the trigger signal and the destination.

Verification

Model Advisor check: “Check trigger signal names” (Simulink Check)

Last Changed

R2020a

See Also

- Sub ID a1, a2, a3, a4, see MISRA AC SLSF 026C
- “Signal Basics”
- “Use Events to Execute Charts” (Stateflow)
- “Explore Types of Subsystems”

Version History

Introduced in R2020a

db_0143: Usable block types in model hierarchy

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Model levels shall use only the block types that are defined for the layer type. Clearly defined layer types restrict the number of blocks that can be used.

Block restrictions:

- (R2011a and earlier) Enable block cannot be used at the root level of the model.
- Action ports are not permitted at the root level of a model.

Layer restrictions:

- Data flow layers that are used for basic blocks only.
- Other than data flow layers, layers can include blocks that are used for structural subsystems and all other layers.

Blocks that can be used for all layers include:

- Inport
- Outport
- Mux
- Demux
- Bus Selector
- Bus Creator
- Selector
- Ground

- Terminator
- From
- Goto
- Merge
- Unit Delay
- Rate Transition
- Data Type Conversion
- Data Store Memory
- If
- Switch Case
- Function-Call Generator
- Function-Call Split
- Import Shadow

Custom Parameter

Layer type

Block type

Rationale

Sub ID a:

- Readability is impaired when subsystems and basic blocks are used in the same layer.

Verification

Model Advisor check: "Check for mixing basic blocks and subsystems" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

db_0144: Use of subsystems

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

Blocks in a Simulink diagram shall be grouped together into subsystems based on functional decomposition of the algorithm, or portion thereof, represented in the diagram. Blocks can also be grouped together based on behavioral variants or timing.

Avoid grouping blocks into subsystems primarily for the purpose of saving space in the diagram. Each subsystem in the diagram should represent a unit of functionality that is required to accomplish the purpose of the model or submodel.

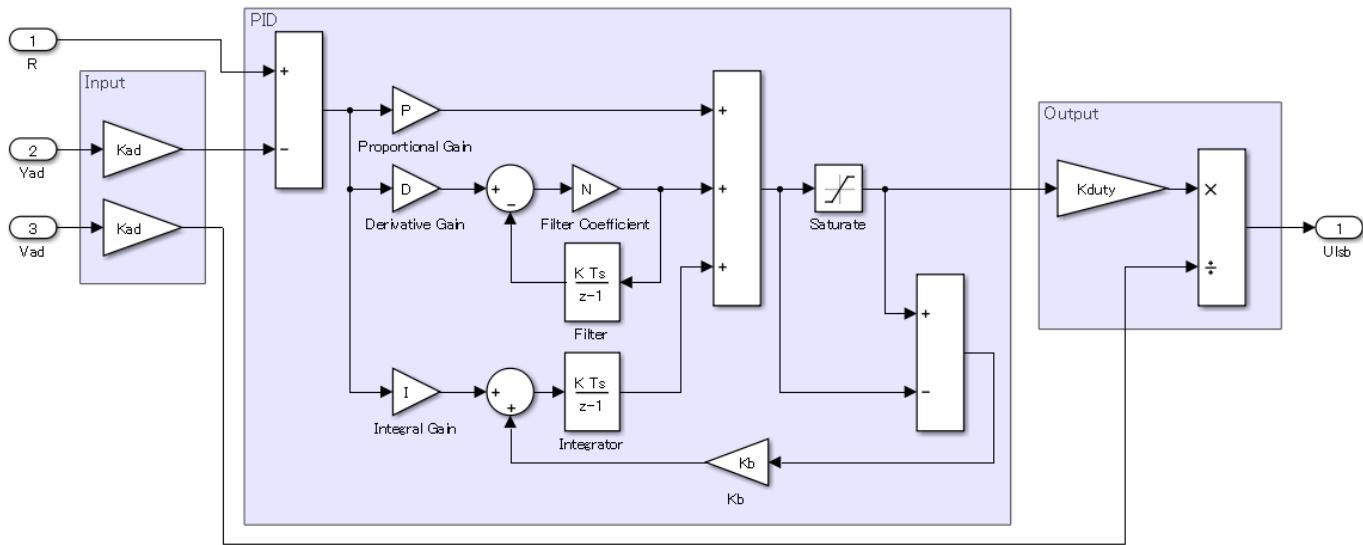
When implementing a subsystem to alleviate readability issues, use a virtual subsystem.

Custom Parameter

Not Applicable

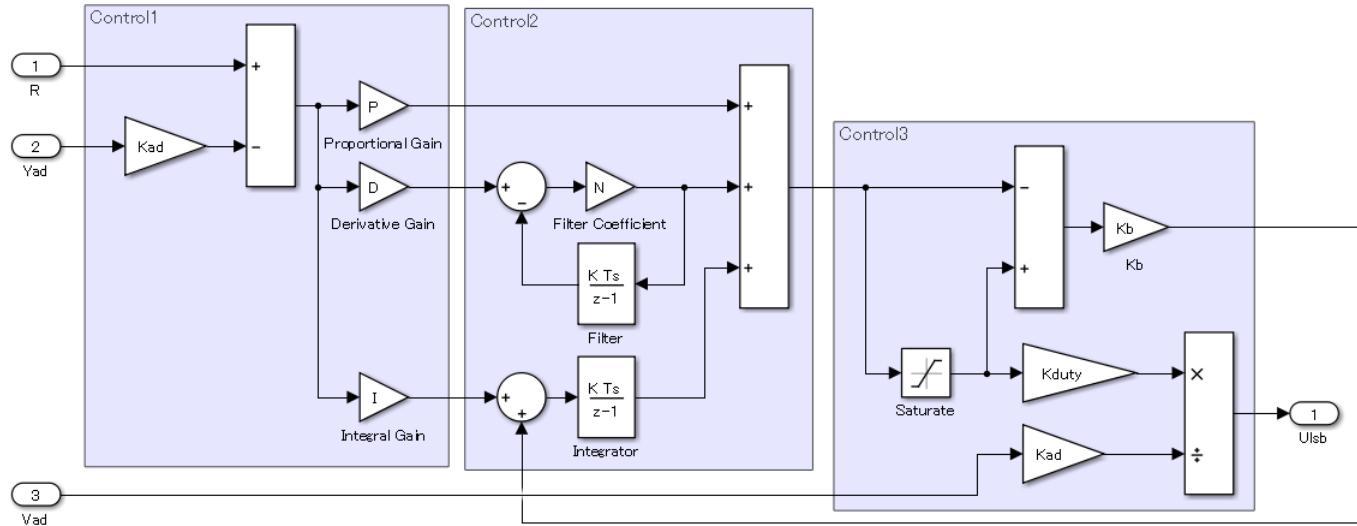
Example — Correct

Subsystems are divided by functional unit.



Example — Incorrect

Subsystems are not divided by functional unit.



Sub ID b

A virtual subsystem shall be used when processing order and code generation does not need to be taken into consideration.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Avoid grouping blocks into subsystems primarily for the purpose of saving space in the diagram.
- It can be difficult to reuse the subsystem.

Sub ID b:

- As atomic subsystems are considered a single process that influences processing order and code optimization, they can be misinterpreted when used other than as intended.

Verification

Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

See Also

- Subsystem

Version History

Introduced in R2020a

jc_0653: Delay block layout in feedback loops

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

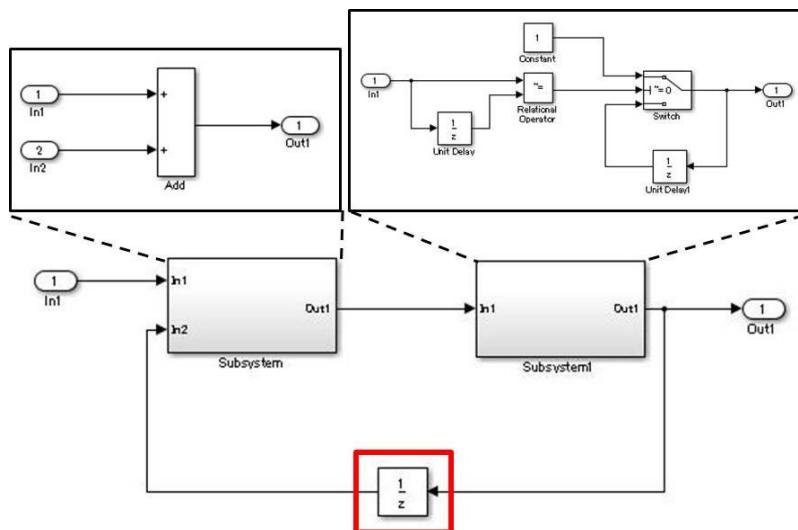
Delay block in feedback loops across subsystems shall reside in the hierarchy that describes the feedback loop.

Custom Parameter

Not Applicable

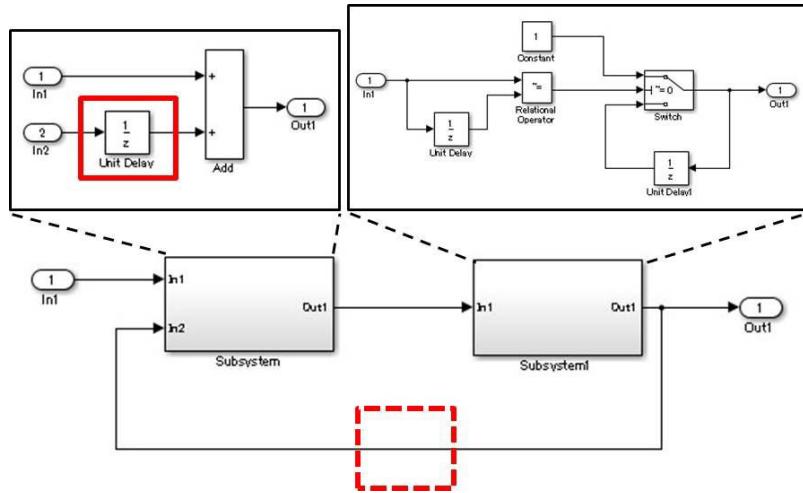
Example — Correct

Delay block resides in the hierarchy that describes the feedback loop.



Example — Incorrect

Delay block resides in a subsystem that is nested within the hierarchy which describes the feedback loop.



Rationale

Sub ID a:

- Prevents double placement of the Delay block.
- Clarifying the extent of diversion improves reusability.
- Improves testability; it is difficult to test a subsystem that contains a Delay block on its own because past values cannot be entered directly.

Verification

Model Advisor check: "Check for avoiding algebraic loops between subsystems" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

hd_0001: Prohibited Simulink sinks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

Control algorithm models shall be designed from discrete blocks.

Scope and Display blocks can be used in the model diagram.

These sink blocks shall not be used:

- To File
- To Workspace
- Stop Simulation

Consider using signal logging and the Viewers and Generators Manager for data logging and viewing requirements. (R2019b and later) To log and manage the signal, click the **Simulation** tab and, under the **Prepare** gallery, select the appropriate tool.

Custom Parameter

Not Applicable

Rationale

Sub ID a

- Improves readability and model simulation.
- Code generation may not be possible.

Verification

Model Advisor check: “Check for prohibited sink blocks” (Simulink Check)

Last Changed

R2020a

See Also

- “Viewers and Generators Manager”

Version History

Introduced in R2020a

Signal

na_0010: Usage of vector and bus signals

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b, c, d

MATLAB Versions

All

Rule

Sub ID a

Mux and Demux blocks shall be used when generating and decomposing vectors.

Custom Parameter

Not Applicable

Sub ID b

Mux block inputs shall be scalars and vectors.

Custom Parameter

Not Applicable

Sub ID c

Bus Creator and Bus Selector blocks shall be used when generating and decomposing buses.

Custom Parameter

Not Applicable

Sub ID d

Buses shall connect to blocks that support buses.

Custom Parameter

Not Applicable

Rationale

Sub IDs a, b, c, d:

- Prevents issues that are caused by combining vector and bus signals.

Verification

Model Advisor check: "Check usage of vector and bus signals" (Simulink Check)

Last Changed

R2021a

See Also

- "Signal Basics"
- "Signal Types"
- "Specify Bus Properties with Bus Objects"
- "Bus-Capable Blocks"

Version History

Introduced in R2020a

jc_0008: Definition of signal names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Signal names shall be defined for signal lines that output from important blocks. The signal name shall be provided once, at the origin of the signal line.

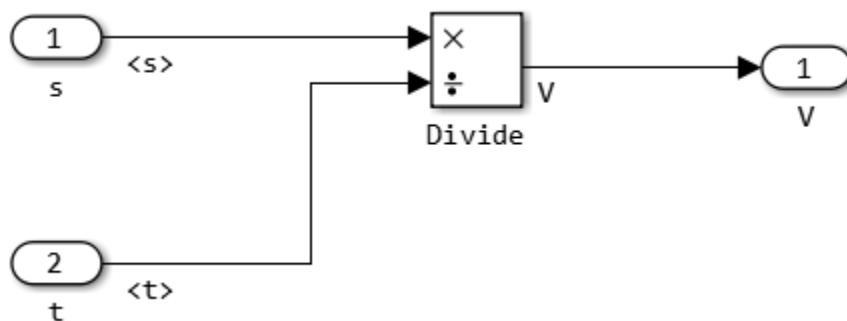
A label shall be used to display defined signal names.

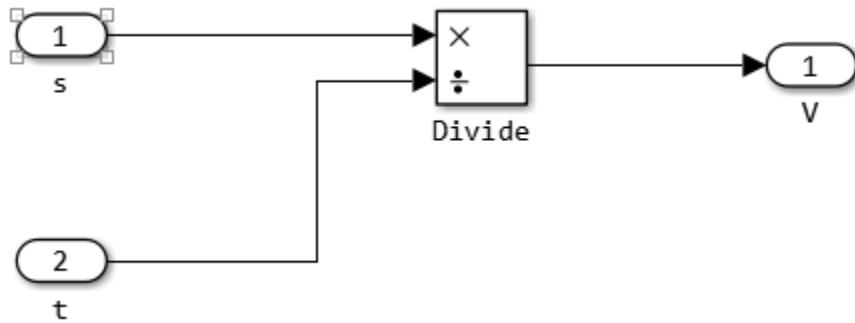
Note An important block is defined by the system input and output of meaningful results, not by its type.

Custom Parameter

Definition of an important block

Example — Correct



Example — Incorrect**Rationale**

Sub ID a:

- Defining the signal name and displaying the label for the output of meaningful results from important blocks improves the readability of the model.

Verification

Model Advisor check: "Check definition of signal labels" (Simulink Check)

Last Changed

R2020a

See Also

- "Signal Basics"
- "Explore Types of Subsystems"

Version History

Introduced in R2020a

jc_0009: Signal name propagation

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

When defining the signal name for a signal that extends across a hierarchy, signal property **Show propagated signals** shall be selected so that propagated signal names are displayed.

However, when one of the following conditions is met, do not select **Show propagated signals**:

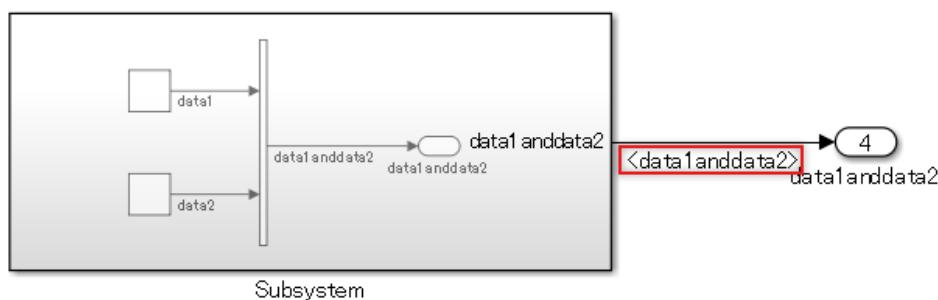
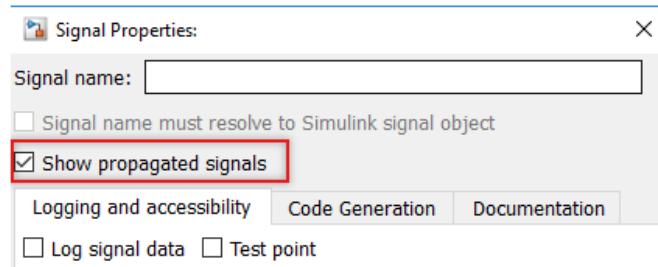
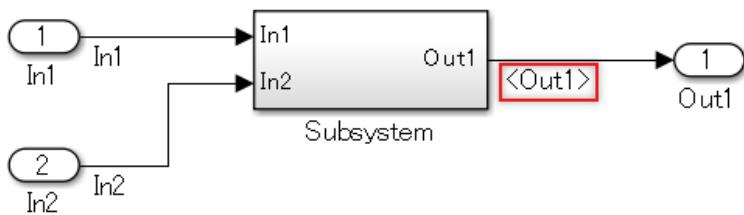
- In a subsystem with a library
- In subsystems where reusable functions are set

Custom Parameter

Not Applicable

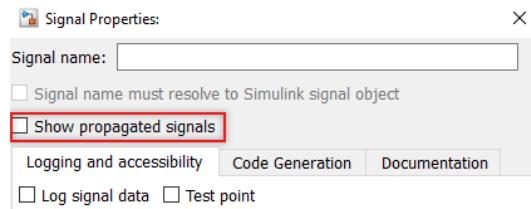
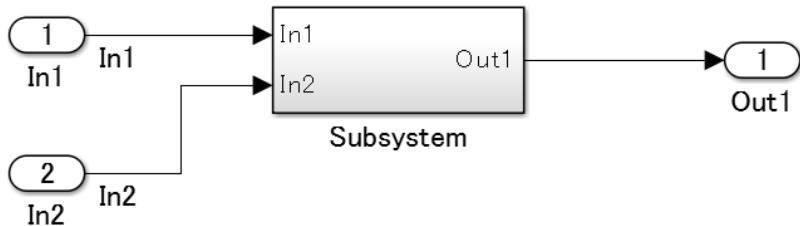
Example — Correct

Propagated signal names are displayed.

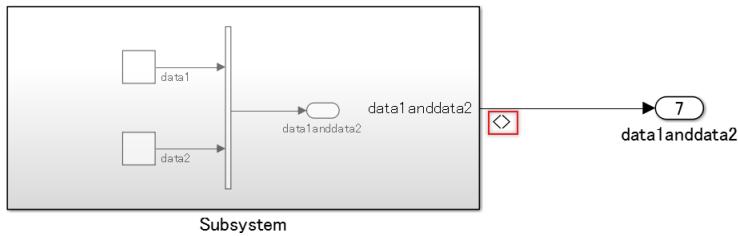


Example — Incorrect

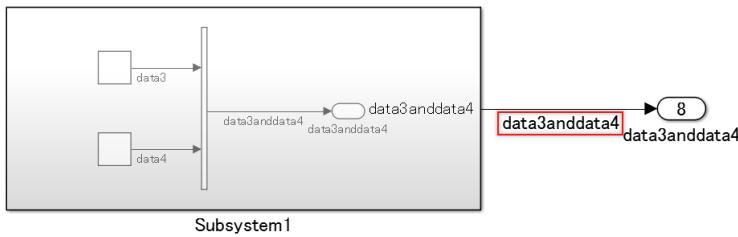
Show propagated signals is not selected, therefore signal names are not displayed.



Signals that connect to Bus Creator and Outport blocks do not have names, but **Show propagated signals** is selected for signals that connect to Subsystem and Outport blocks.



Signals that connect to Bus Creator and Outport blocks have names, but signals that connect to Subsystem and Outport blocks also have names.



Sub ID b

Signal property **Show propagated signals** shall be selected for these blocks so that propagated signal names of the signal output are displayed:

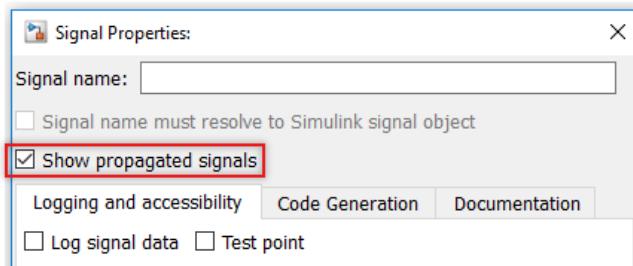
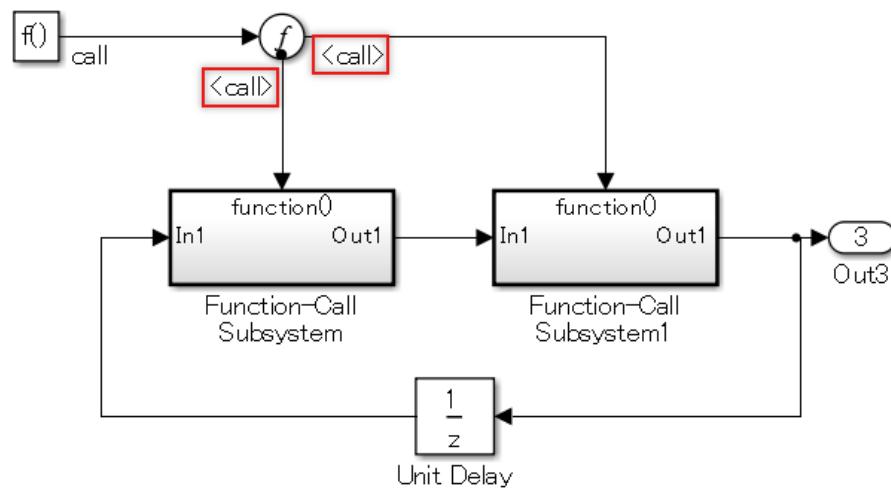
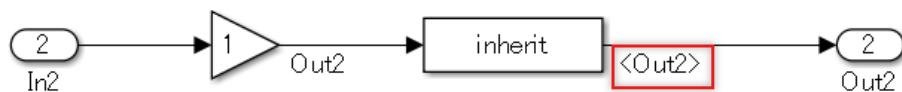
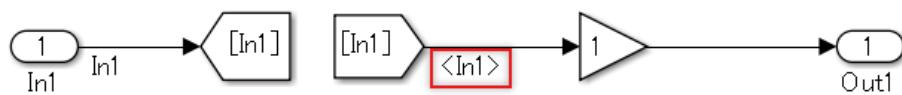
- From
- Signal Specification
- Function-Call Split[

Custom Parameter

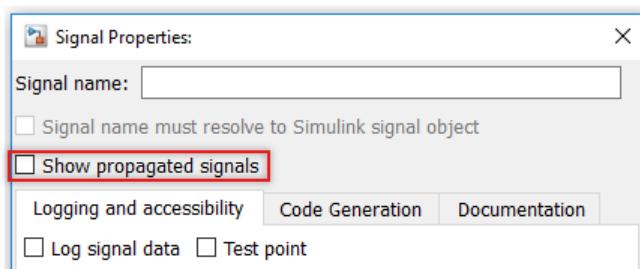
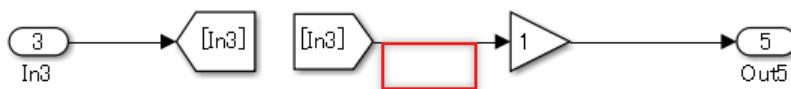
Not Applicable

Example — Correct

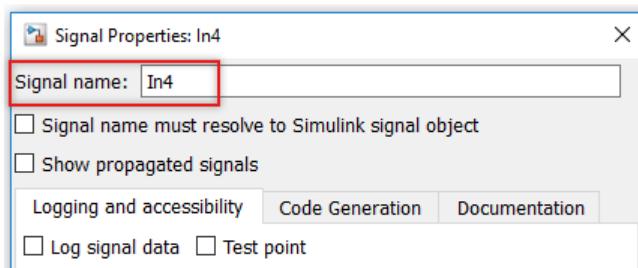
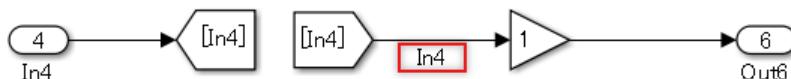
Propagated signal names are displayed.



Signals that connect to Import and Goto blocks do not have names, therefore **Show propagated signals** does not need to be selected.

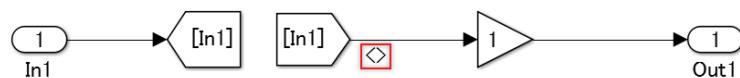


Signals that connect to Import and Goto blocks do not have names, therefore signals that connect to From and Gain blocks can be left unnamed.



Example — Incorrect

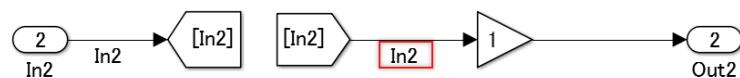
Signals that connect to Import and Goto blocks do not have names, but **Show propagated signals** is selected for signals that connect to From and Gain blocks.



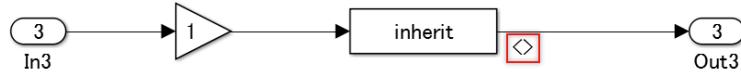
Regardless of whether signals are propagated, **Show propagated signals** is not selected



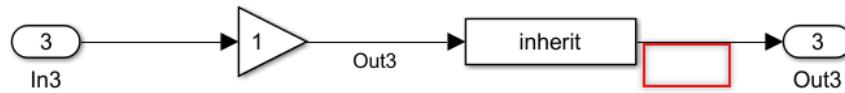
Signals that connect to Import and Goto blocks have names, but signals that connect to From and Gain blocks are named.



Signals that connect to Gain and Signal Specification blocks do not have names, but **Show propagated signals** is selected for signals that connect to Signal Specification and Outport blocks.



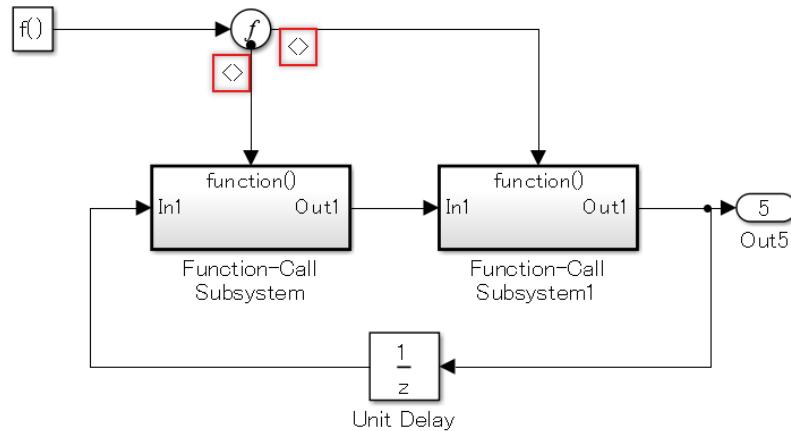
Regardless of whether signals are propagated, **Show propagated signals** is not selected.



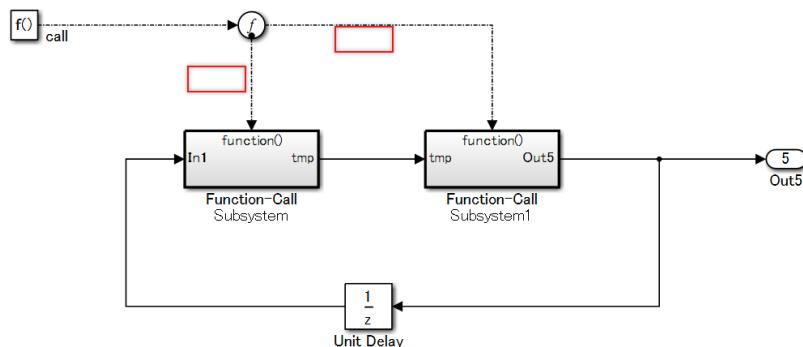
Signals that connect to Gain and Signal Specification blocks have names, but signals that connect to Signal Specification and Outport blocks also have names.



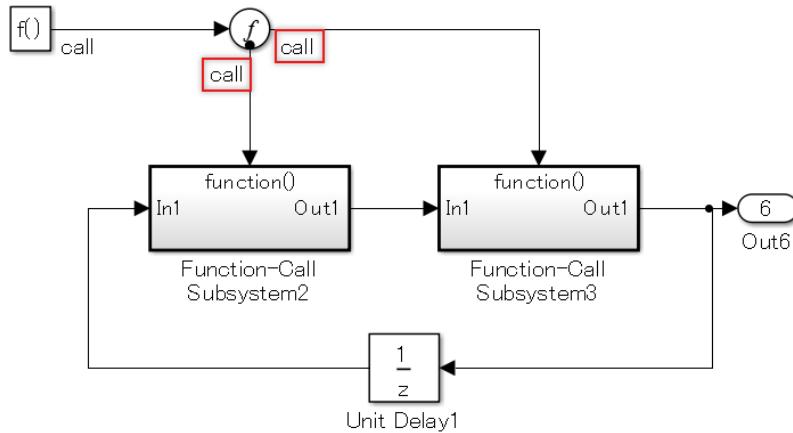
Signals that connect to Function-Call Generator and Function-Call Split blocks do not have names, but **Show propagated signals** is selected for signals that connect to Function-Call Split and Function-Call Subsystem blocks.



Regardless of whether signals are propagated, **Show propagated signals** is not selected.



Signals that connect to Function-Call Generator and Function-Call Split blocks have names and signals that connect to Function-Call Split and Function-Call Subsystem blocks are also named.

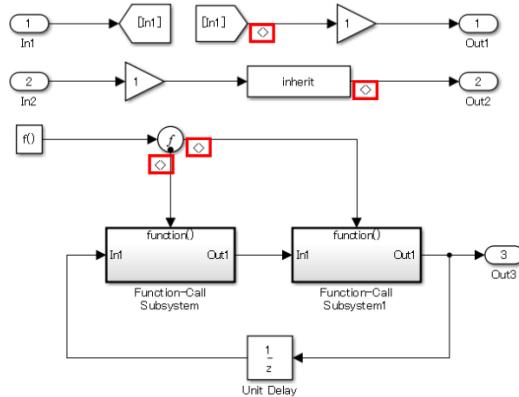


Sub ID c

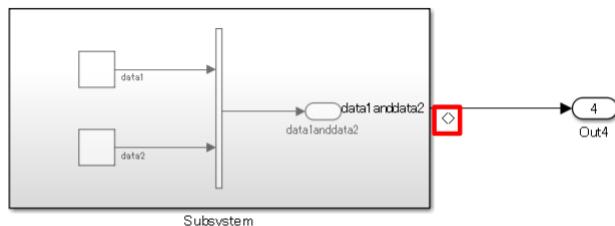
When propagated signal names does not exist **Show propagated signals** shall not be selected.

Example — Incorrect

Signals that connect to Bus Creator and Outport blocks do not have names, but **Show propagated signals** is selected for signals that connect to Subsystem and Outport blocks.



The Subsystem block is propagated empty.



Rationale

Sub IDs a, b:

- Prevents signal line connection errors.
- Prevents incorrect naming of signal lines.

Sub ID c:

- Standardizing the style of writing enhances readability.

Verification

Model Advisor check: “Check signal name propagation” (Simulink Check)

Last Changed

R2024b

See Also

- “Signal Basics”
- “Explore Types of Subsystems”

Version History

Introduced in R2020a

db_0097: Position of labels for signals and buses

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

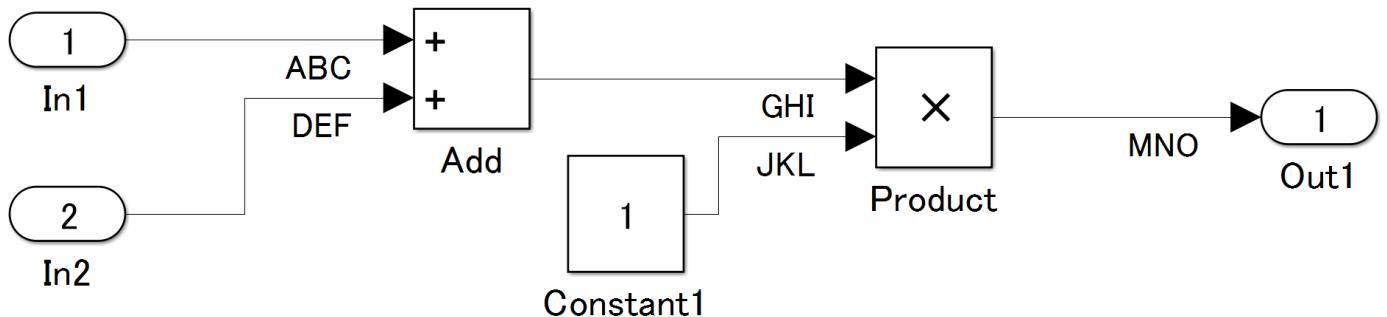
Signal line labels and bus labels shall be positioned below signal lines.

Custom Parameter

Not Applicable

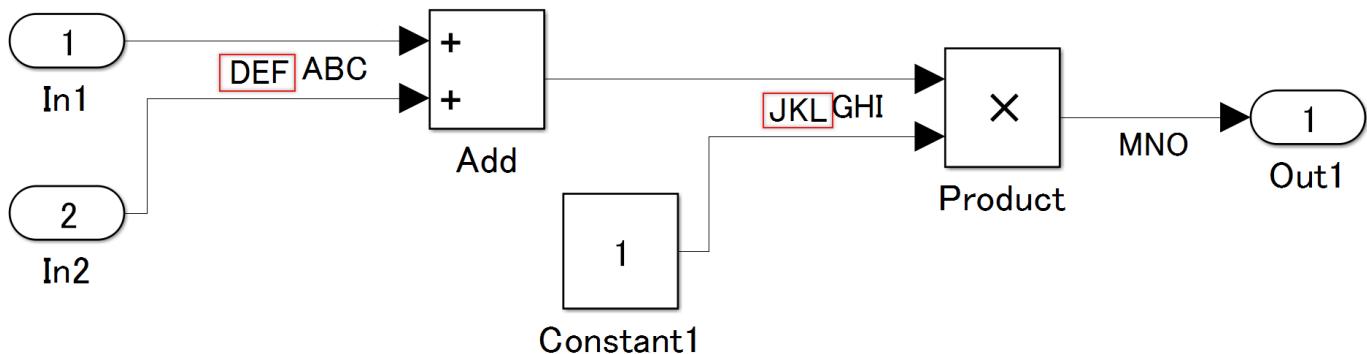
Example — Correct

Signal line labels and bus labels are below signal lines.



Example — Incorrect

Signal line labels and bus labels are above the signal line.



Sub ID b

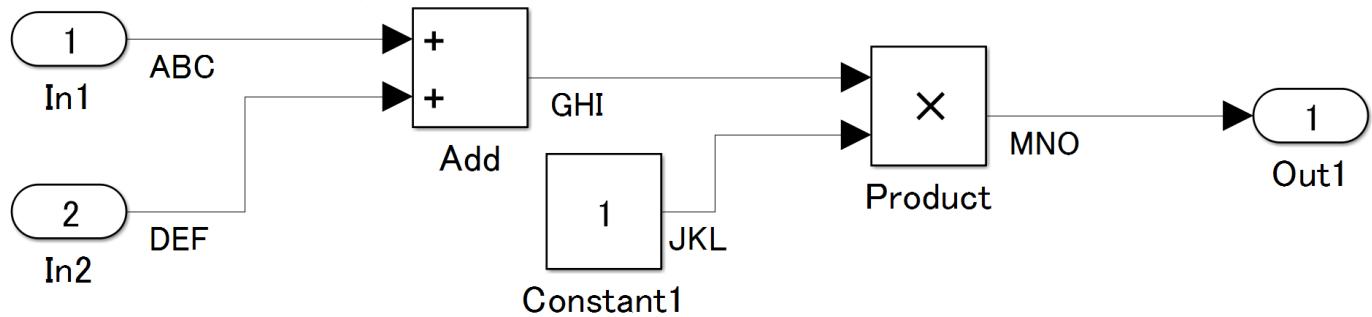
Signal line labels and bus labels shall be positioned at the origin of the connection.

Custom Parameter

Not Applicable

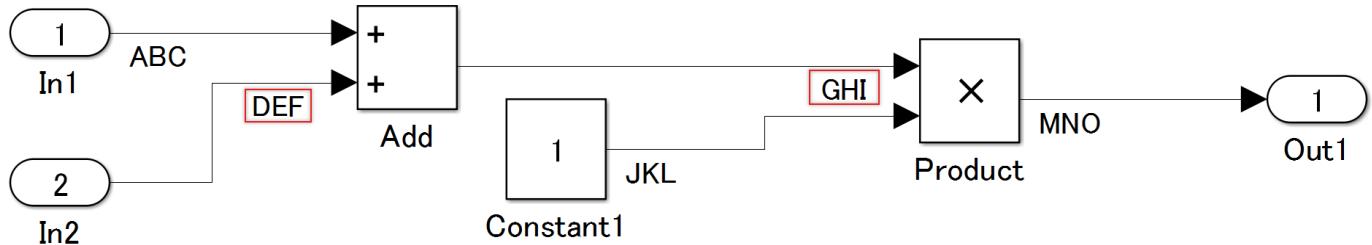
Example — Correct

Signal line labels and bus labels are positioned at the origin of the signal line connection.



Example — Incorrect

Signal line labels and bus labels are positioned at the destination of the signal line connection.



Rationale

Sub ID a, b:

- Consistent label position prevents confusion with corresponding labels, signal lines, and buses, which improves the readability of the model.

Verification

Model Advisor check: "Check position of signal labels" (Simulink Check)

Last Changed

R2024b

See Also

- “Signal Basics”
- “Signal Lines”
- “Simulink Bus Capabilities”

Version History

Introduced in R2020a

na_0008: Display of labels on signals

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

A label shall be displayed on the signal line originating from these blocks:

- Import
- From (see exception)
- Subsystem or Stateflow Chart (see exception)
- Bus Selector (the tool forces this to happen)
- Demux
- Selector
- Data Store Read (see exception)
- Constant (see exception)

Exception

When the signal label is visible in the originating block icon display, the signal does not need to have the label displayed *unless* the signal label is needed elsewhere due to a destination-based rule.

Custom Parameter

Not Applicable

Sub ID b

A label shall be displayed on a signal line that connects (either directly or by way of a basic block that performs a non-transformative operation) to these destination blocks:

- Outport
- Goto
- Data Store Write
- Bus Creator
- Mux
- Subsystem
- Stateflow Chart

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Improves readability, model simulation, and workflow.
- Code generation may not be possible.

Sub ID b:

- Improves readability, model simulation, and workflow.

Verification

Model Advisor check: “Check signal line labels” (Simulink Check)

Last Changed

R2020a

See Also

- “Signal Basics”
- “Manage Signal Lines”
- “Explore Types of Subsystems”

Version History

Introduced in R2020a

na_0009: Entry versus propagation of signal labels

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

When a label is displayed for a signal, the following rules define whether that label is created there (entered directly on the signal) or propagated from its true source (inherited from elsewhere in the model by using the < character).

Signal labels shall be entered for signals that originate from:

- The Import block at the root (top) level of a model
- Basic blocks that perform a transformative operation (For the purpose of interpreting this rule only, the Bus Creator, Mux, and Selector blocks also perform transformative operations.)

Signal labels shall be propagated for signals that originate from:

- Import block in a nested subsystem
- Basic blocks that perform a non-transformative operation
- Subsystem block or Stateflow Chart block.

Exceptions

When the nested subsystem is a library subsystem, a label can be entered on the signal coming from the Import block to accommodate reuse of the library block.

When the connection originates from the output of a library subsystem block, a new label can be entered on the signal to accommodate readability.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- The result of executing a MATLAB command is reflected in the code, which makes consistency between the model and code difficult to maintain.

Verification

Model Advisor check: "Check for propagated signal labels" (Simulink Check)

Last Changed

R2020a

See Also

- "Signal Basics"
- "Manage Signal Lines"
- "Signal Label Propagation"
- "Explore Types of Subsystems"

Version History

Introduced in R2020a

db_0110: Block parameters

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Block parameters shall not be used to describe:

- Operation expressions
- Data type conversion
- Selection of rows or columns
- MATLAB commands

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Operation expressions, data type conversion, or row or column selection become a magic number in generated code, which makes consistency between the model and code difficult to maintain. Adjusting parameters also becomes difficult.
- Describing the calculation formula within the block decreases readability.
- The result of executing a MATLAB command is reflected in the code, which makes consistency between the model and code difficult to maintain.

Verification

Model Advisor check: "Check usage of tunable parameters in blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

db_0112: Usage of index

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

Sub ID Recommendations

- NA-MAAB — a1/a2
- JMAAB — a1/a2

MATLAB Versions

All

Rule

Sub ID a1

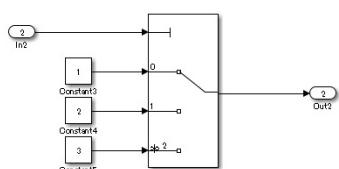
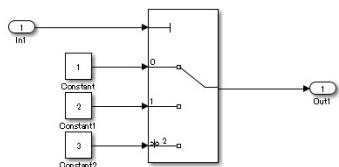
A vector signal shall use a zero-based index mode.

Custom Parameter

Not Applicable

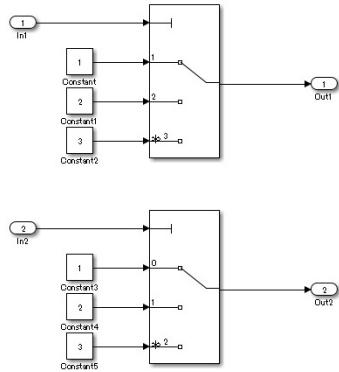
Example — Correct

A uniform zero-based index mode is used.



Example — Incorrect

A uniform index mode is not used.

**Sub ID a2**

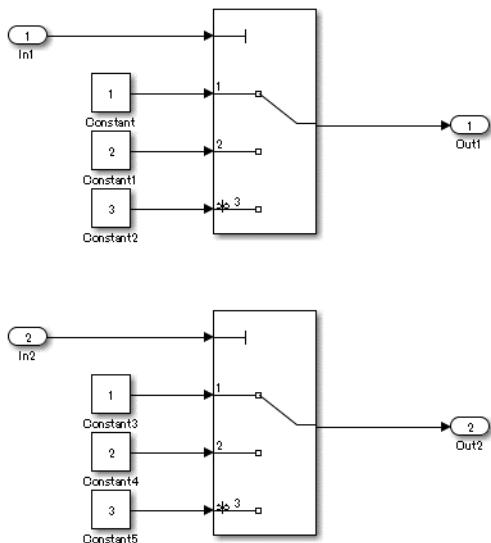
A vector signal shall use a one-based index mode.

Custom Parameter

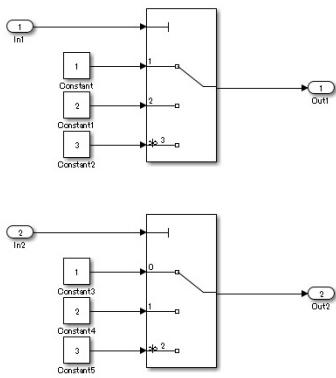
Not Applicable

Example — Correct

A uniform one-based index mode is used.

**Example — Incorrect**

A uniform index mode is not used (same as the incorrect example for sub ID a1).



Rationale

Sub IDs a1, a2

- Logic is easier to understand when using a uniform index mode.

Verification

Model Advisor check: "Check Indexing Mode" (Simulink Check)

Last Changed

R2020a

See Also

- Index Vector

Version History

Introduced in R2020a

jc_0645: Parameter definition for calibration

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

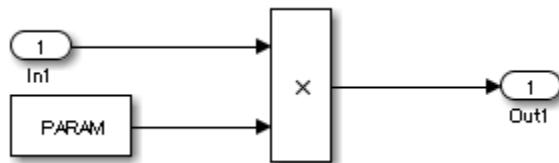
Block parameters that are targets of calibration shall be defined as named constants. Examples of parameters that are outside of the calibration target include:

- Initial value parameter 0
- Increment, decrement 1
- Arithmetic expressions

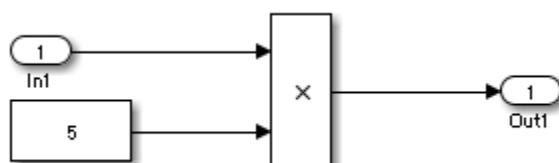
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- A literal constant in the model will propagate as a literal constant in the generated code, making calibration impossible.

Verification

Model Advisor check: "Check if tunable block parameters are defined as named constants" (Simulink Check)

Last Changed

R2020a

See Also

- "Set Block Parameter Values"

Version History

Introduced in R2020a

jc_0641: Sample time setting

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Block parameter **Sample time** shall be set to -1 (inherited).

Exceptions

- Import block
- Outport block
- Atomic subsystem
- Blocks with state variables, such as Unit Delay and Memory
- Signal conversion blocks, such as Data Type Conversion and Rate Transition
- Blocks that do not have external inputs, such as Constant
- Stateflow charts

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Discrepancies can occur in the processing of the model because of different simulation times.
- Maintainability of the model deteriorates when a specific sample time is set for each block individually.

Verification

Model Advisor check: “Check for sample time setting” (Simulink Check)

Last Changed

R2020a

See Also

- “What Is Sample Time?”
- “Explore Types of Subsystems”
- “Construct and Run a Stateflow Chart” (Stateflow)

Version History

Introduced in R2020a

jc_0643: Fixed-point setting

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

When block parameter **Data type** is a set to `fixdt` (fixed-point) and **Scaling** is set to `Slope` and `bias`, parameter **Bias** shall be set to `0`.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- When the bias in a model is not uniform:
 - Behavior of the model is impossible to determine by its appearance.
 - Unintended overflows and underflows occur.
 - Results in wasteful operation and deterioration of code efficiency/computing load.

Verification

Model Advisor check: “Check usage of fixed-point data type with non-zero bias” (Simulink Check)

Last Changed

R2020a

See Also

- “Specify Fixed-Point Data Types”
- “Specify Data Types Using Data Type Assistant”
- “Scaling, Precision, and Range”

Version History

Introduced in R2020a

jc_0644: Type setting

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

The data type shall not be set by using a block or Stateflow data when the data type is set by a data object.

Exceptions

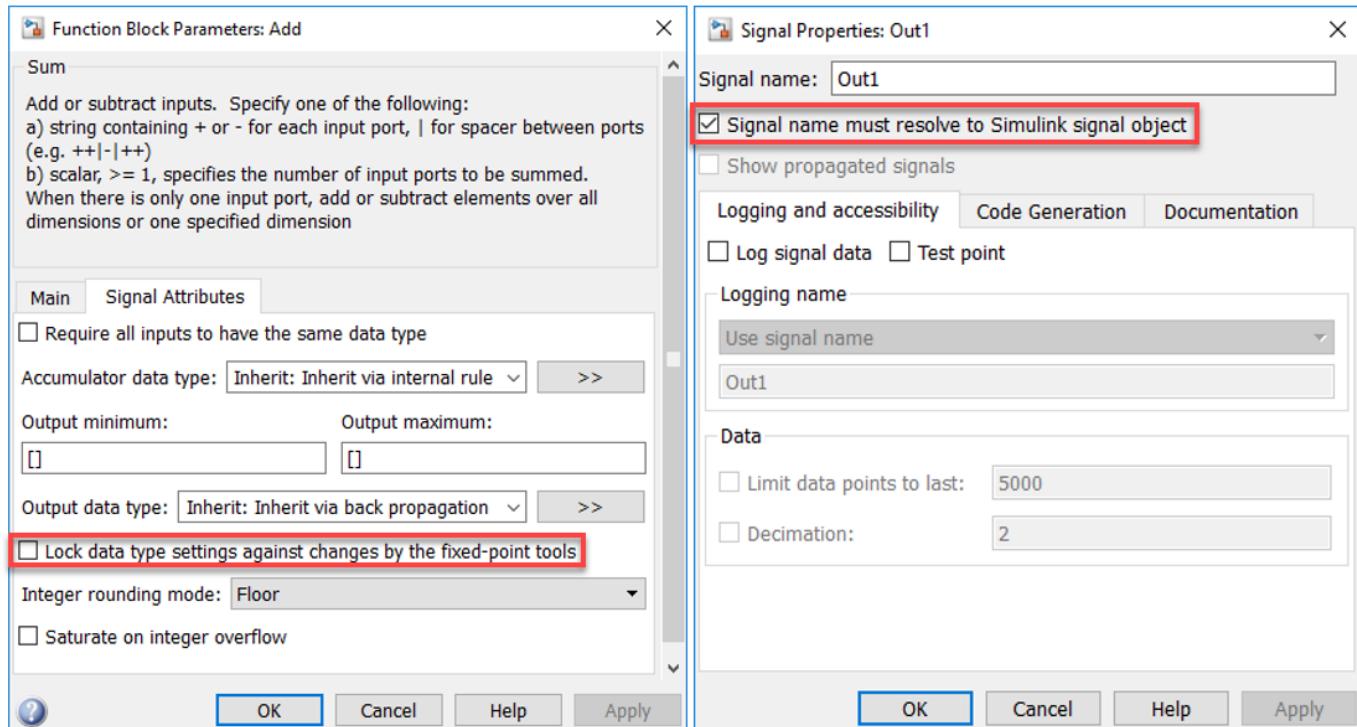
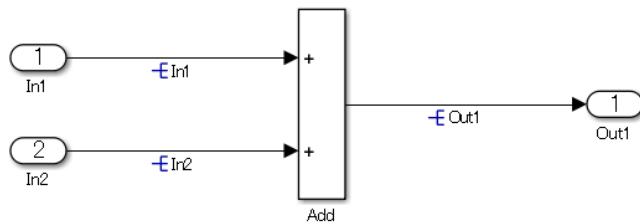
- Inside a reusable function
- Data Type Conversion block
- Data types set by using `fixdt`
- Boolean or double types

Custom Parameter

Not Applicable

Example — Correct

Type is set on the data object.



Rationale

Sub ID a:

- When the data type is set in a block and it differs from the type setting in the data object, it can be difficult to determine which setting is correct. This can impair readability.
- When the type is set in the block, maintainability is affected when the signal line type changes.

Exceptions

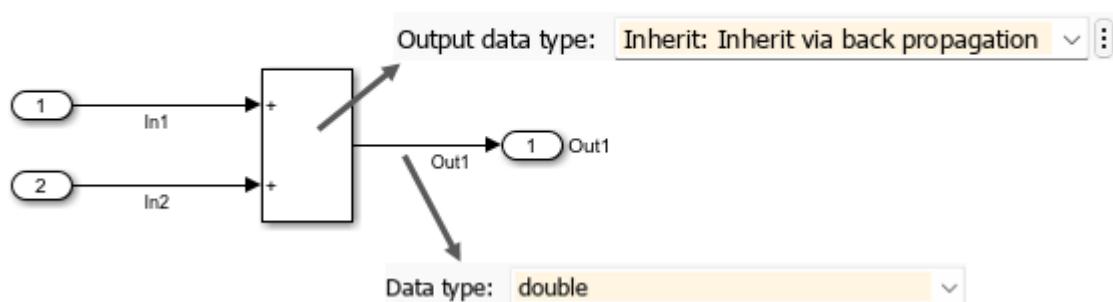
- When block structures are identical, differences between input/output data type can result in different C source code that is not reusable. For reusable functions, data types of input/output blocks should be specified at the subsystem level.
- The Data Type Conversion block is used to explicitly set the data type.
- When the data type is `fixdt` (fixed-point), data type must be set individually because each block can have different data points. In this scenario, it is impossible to use only the data object to set the data type.

- A block with a specific type set by default.

For example, **Output data type** is set to **boolean** by default. It is generally expected that the result of a logical operation **boolean**, eliminating the need to adjust inheritance settings. Given that the expected type is preconfigured in blocks set by default in this manner, readability and maintainability are not compromised.

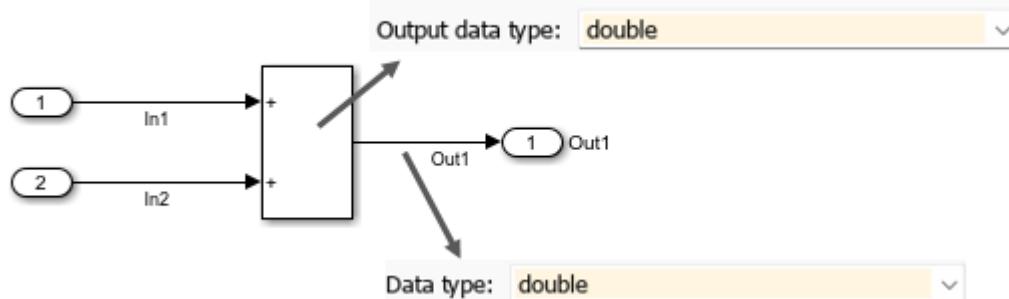
Example — Correct

Type is set for the data object but not the block.



Example — Incorrect

Type is set for both the data object and the block.



Verification

Model Advisor check: "Check type setting by data objects" (Simulink Check)

Last Changed

R2024b

See Also

- “About Data Types in Simulink”
- “Simulink Functions Overview”

Version History

Introduced in R2020a

Conditional Subsystem Relations

db_0146: Block layout in conditional subsystems

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

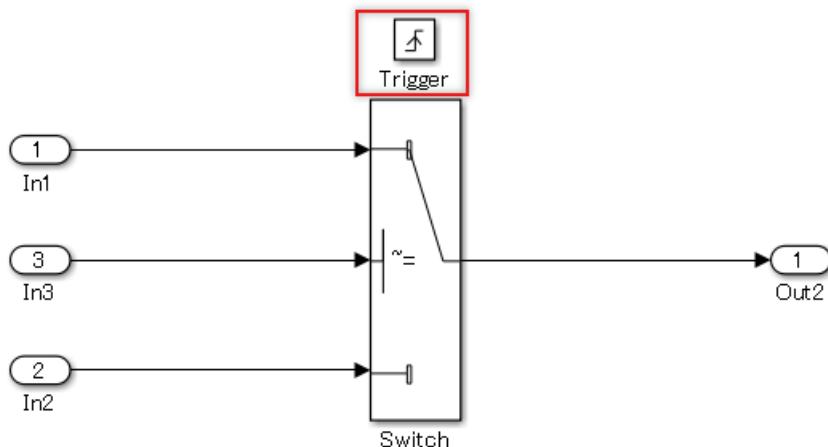
Sub ID a

Conditional input blocks shall be positioned at the top of the subsystem.

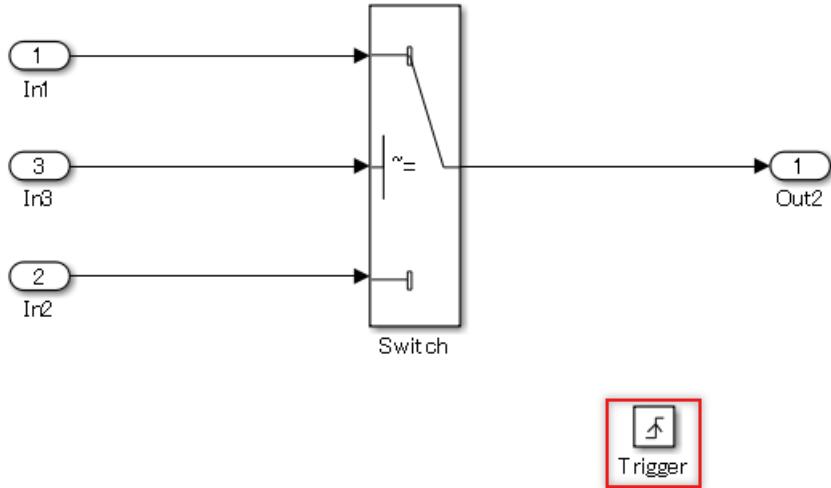
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Sub ID b

The position of these blocks shall be defined by the project:

- For Each
- For Iterator
- While Iterator

Custom Parameter

Location layout

Rationale

Sub IDs a, b:

- Unifying the internal and external layout of the conditional subsystem improves readability of the model.

Verification

Model Advisor check: "Check position of conditional blocks and iterator blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Conditionally Executed Subsystems Overview”

- “Explore Types of Subsystems”

Version History

Introduced in R2020a

jc_0640: Initial value settings for Outport blocks in conditional subsystems

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

The initial condition shall be defined on an Outport block when both of these conditions are met for a conditional subsystem:

- Includes a block with initial conditions (i.e. Constant and Delay blocks)
- Connects to Outport block

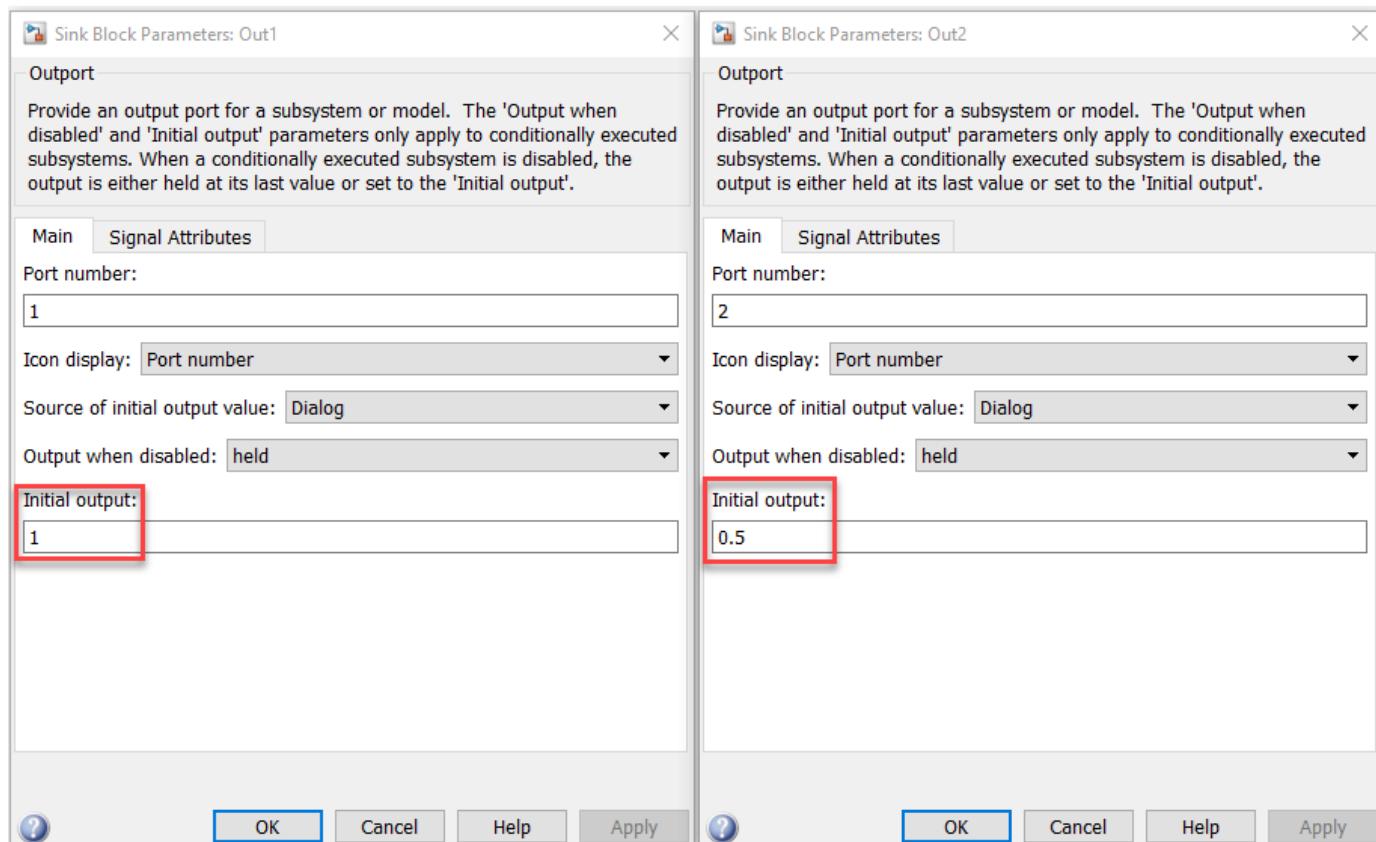
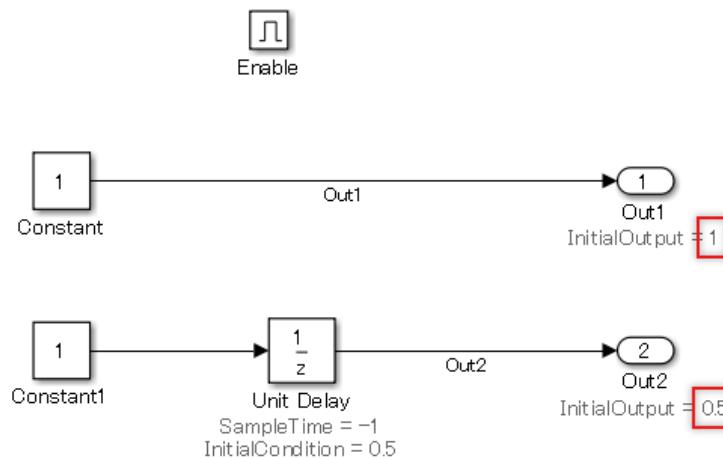
When the output signal from a conditional subsystem is connected to a Merge block, the initial condition shall be defined on the Merge block.

Custom Parameter

Not Applicable

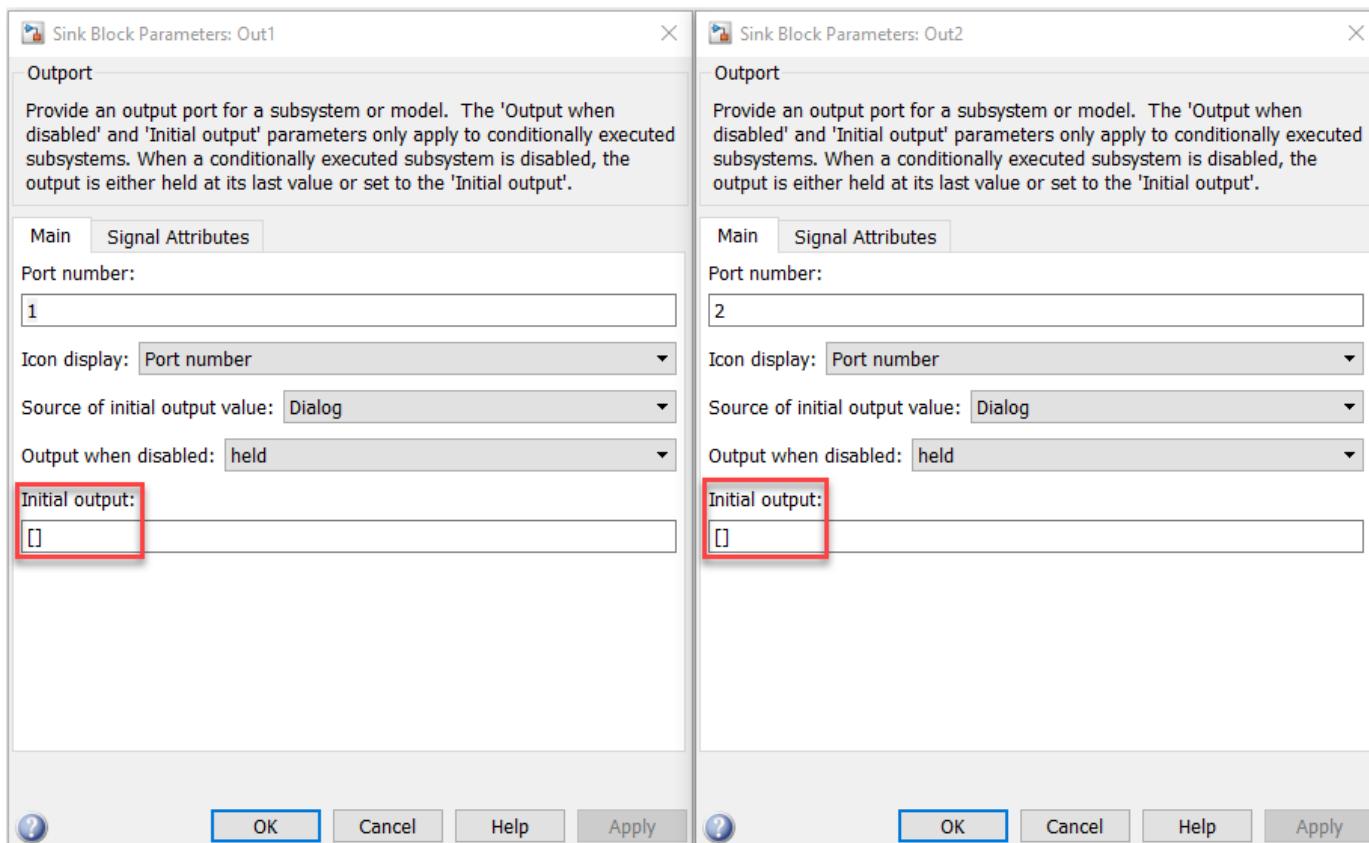
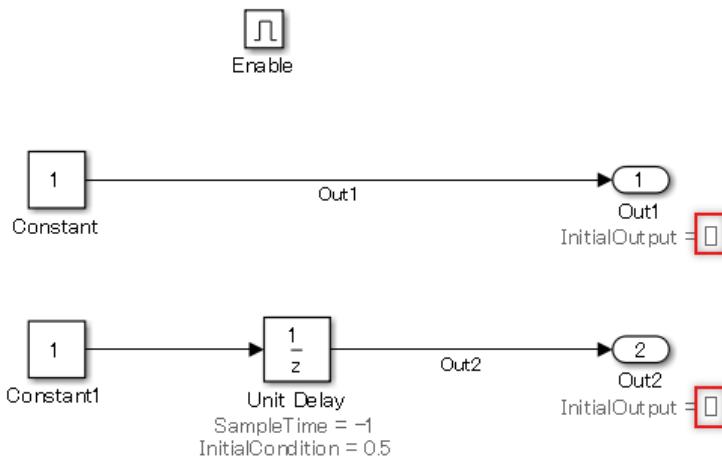
Example — Correct

The initial condition is defined.



Example — Incorrect

The initial condition is not defined.



Rationale

Sub ID a:

- The model may not behave as intended when the initial condition is unclear.

Verification

Model Advisor check: “Check undefined initial output for conditional subsystems” (Simulink Check)

Last Changed

R2020a

See Also

- “Explore Types of Subsystems”
- “Conditionally Executed Subsystems and Models”

Version History

Introduced in R2020a

jc_0659: Usage restrictions of signal lines input to Merge blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

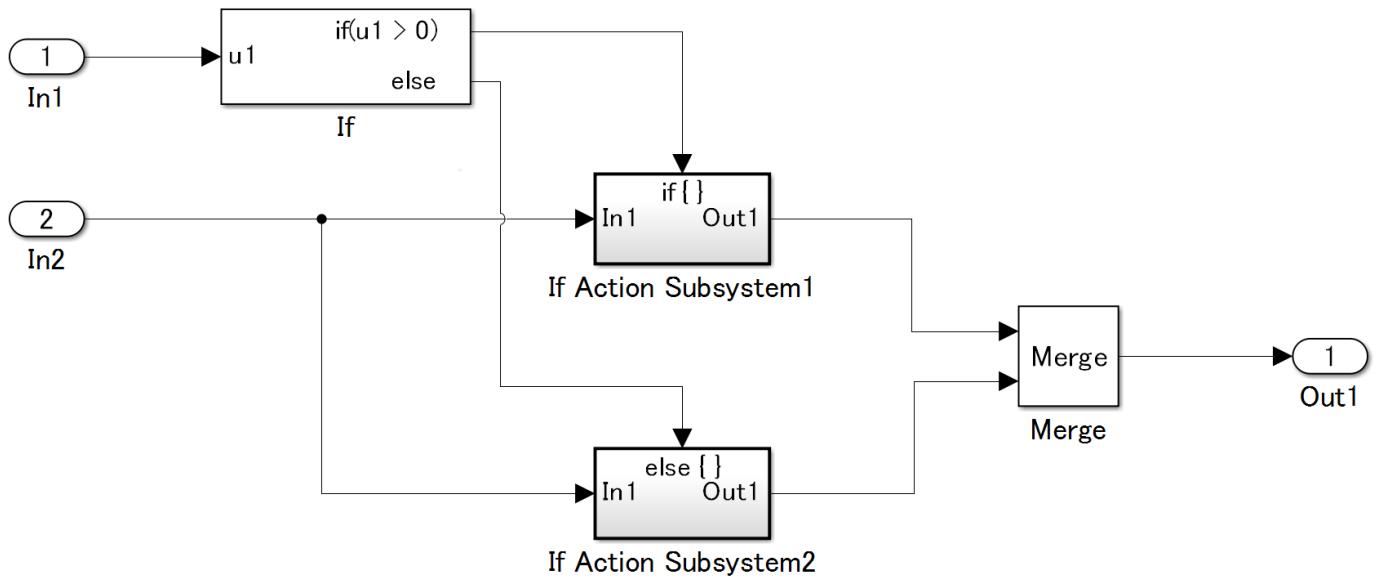
Sub ID a

Only conditional subsystem output signals shall input to Merge blocks.

Custom Parameter

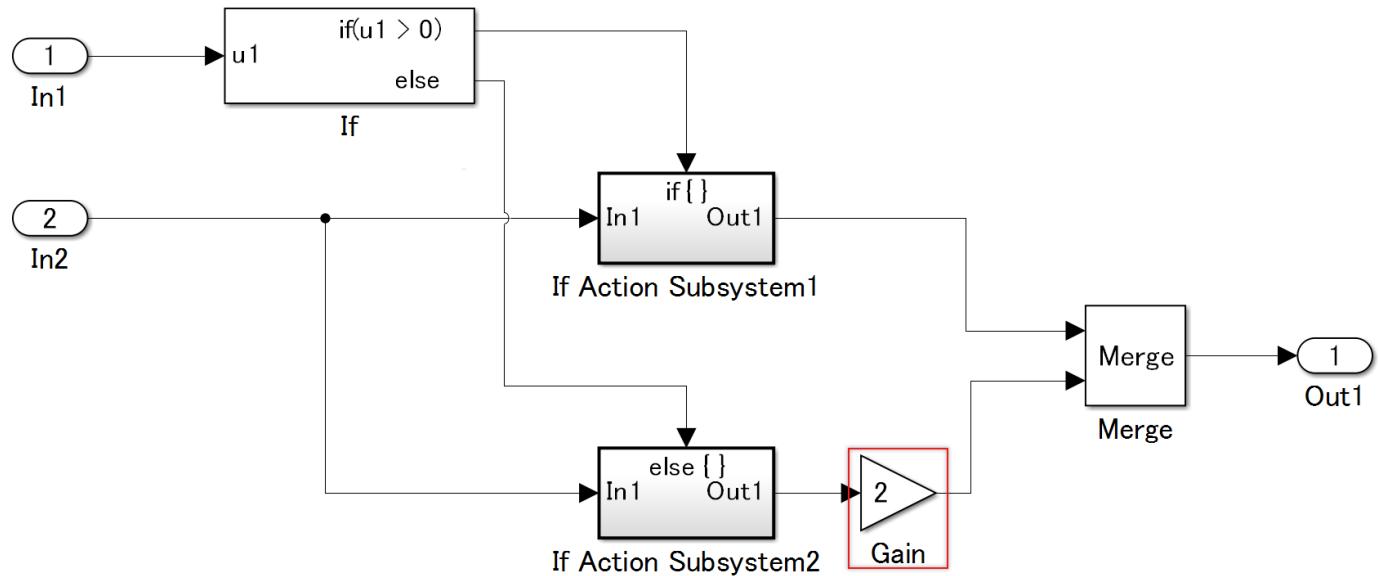
Not Applicable

Example — Correct



Example — Incorrect

A Gain block output signal is input to Merge.



Rationale

Sub ID a:

- Prevents the simulation from proceeding as intended.

Verification

Model Advisor check: "Check usage of Merge block" (Simulink Check)

Last Changed

R2020a

See Also

- "Explore Types of Subsystems"
- "Conditionally Executed Subsystems Overview"

Version History

Introduced in R2020a

na_0003: Usage of If blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

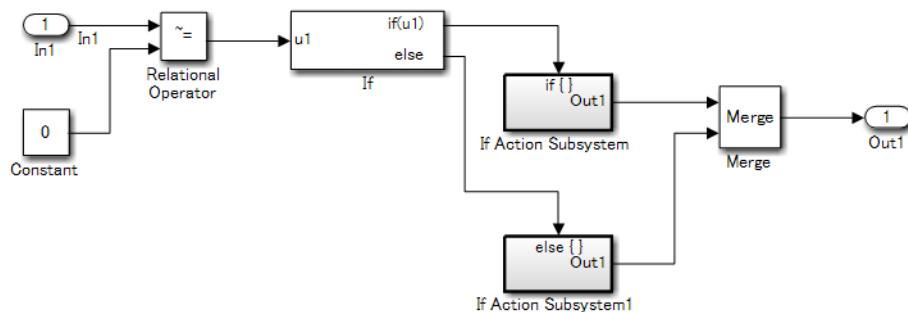
For the If block, the `if` expression and `elseif` expression shall be used only to define input signals.

Custom Parameter

Not Applicable

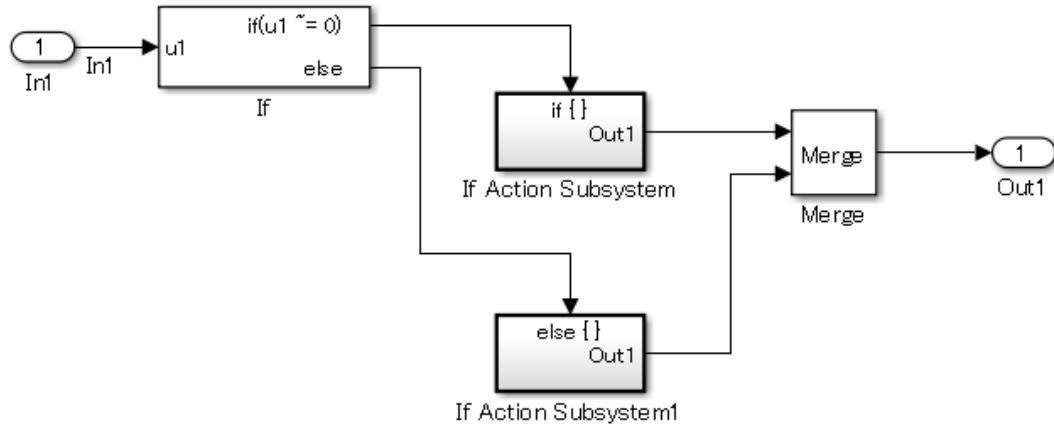
Example — Correct

The `if` expression only defines the input variables.



Example — Incorrect

The `if` expression defines a comparison operation.



Rationale

Sub ID a:

- Visual comprehension of control conditions is easier when logical operations are described outside of the If block.
- Describing logical operations outside of the If block allows verification to focus on the logical operation.

Verification

Model Advisor check: "Check logical expressions in If blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Loops and Conditional Statements”

Version History

Introduced in R2020a

jc_0656: Usage of Conditional Control blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

To make all actions in the conditions explicit, these block parameters shall be set as follows :

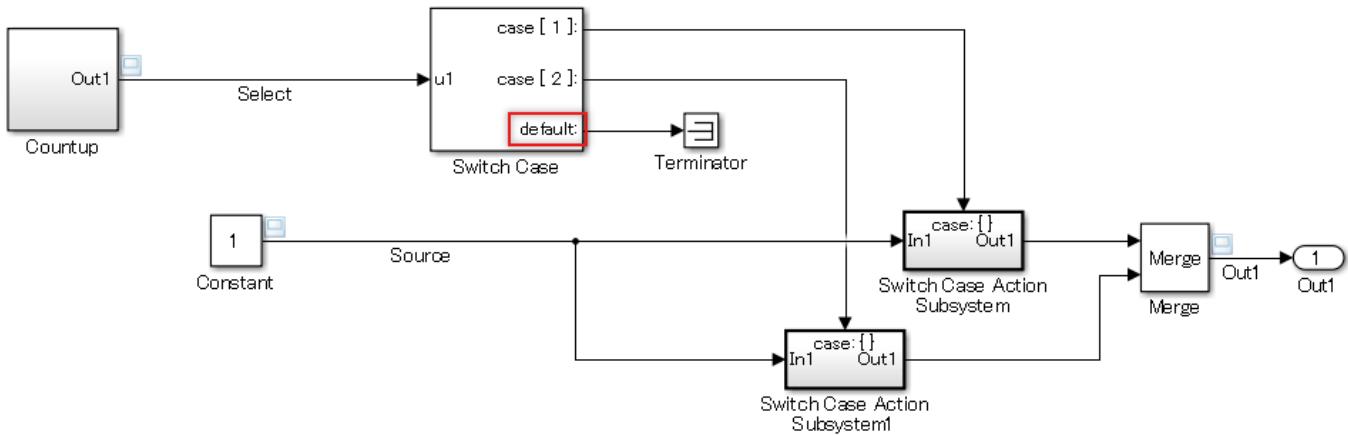
- For If blocks, select **Show else condition**
- For Switch Case blocks, select **Show default case**

Custom Parameter

Not Applicable

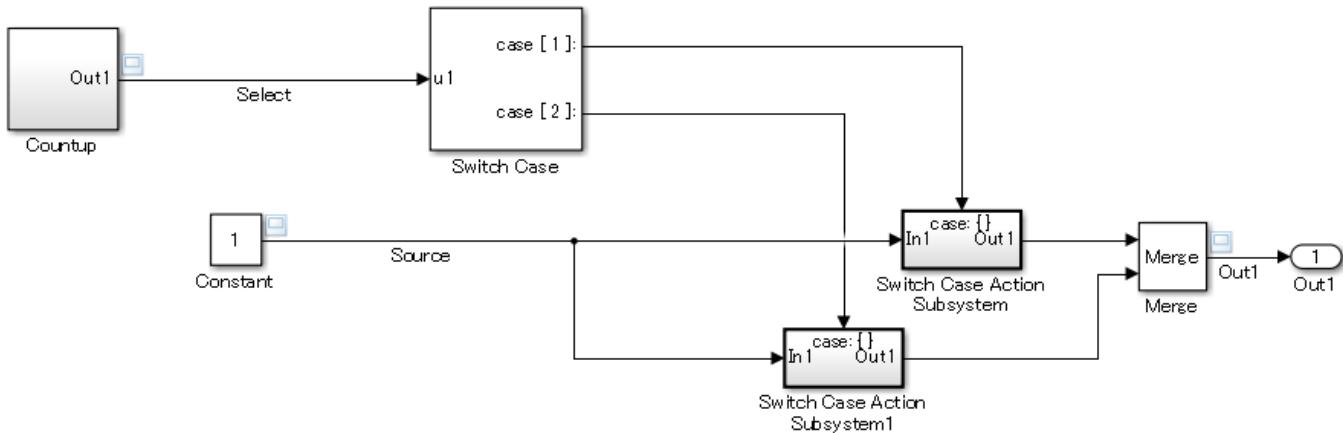
Example — Correct

Using default behavior in the Switch Case block.



Example — Incorrect

Not using default behavior in the Switch Case block.



Rationale

Sub ID a:

- Determining whether there is pointless processing or if something is missing from the design (such as a missing description) is easier when the processing of exceptions (`else`, `default`) is explicitly set in the model.

Verification

Model Advisor check: “Check default/else case in Switch Case blocks and If blocks” (Simulink Check)

Last Changed

R2020a

See Also

- “Conditionally Executed Subsystems Overview”
- “Edit Block Parameters”
- “Select Subsystem Execution”

Version History

Introduced in R2020a

jc_0657: Retention of output value based on conditional control flow blocks and Merge blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a2
- JMAAB — a1/a2

MATLAB Versions

All

Rule

Sub ID a1

Unused action ports shall connect to Terminator block when these conditions are met:

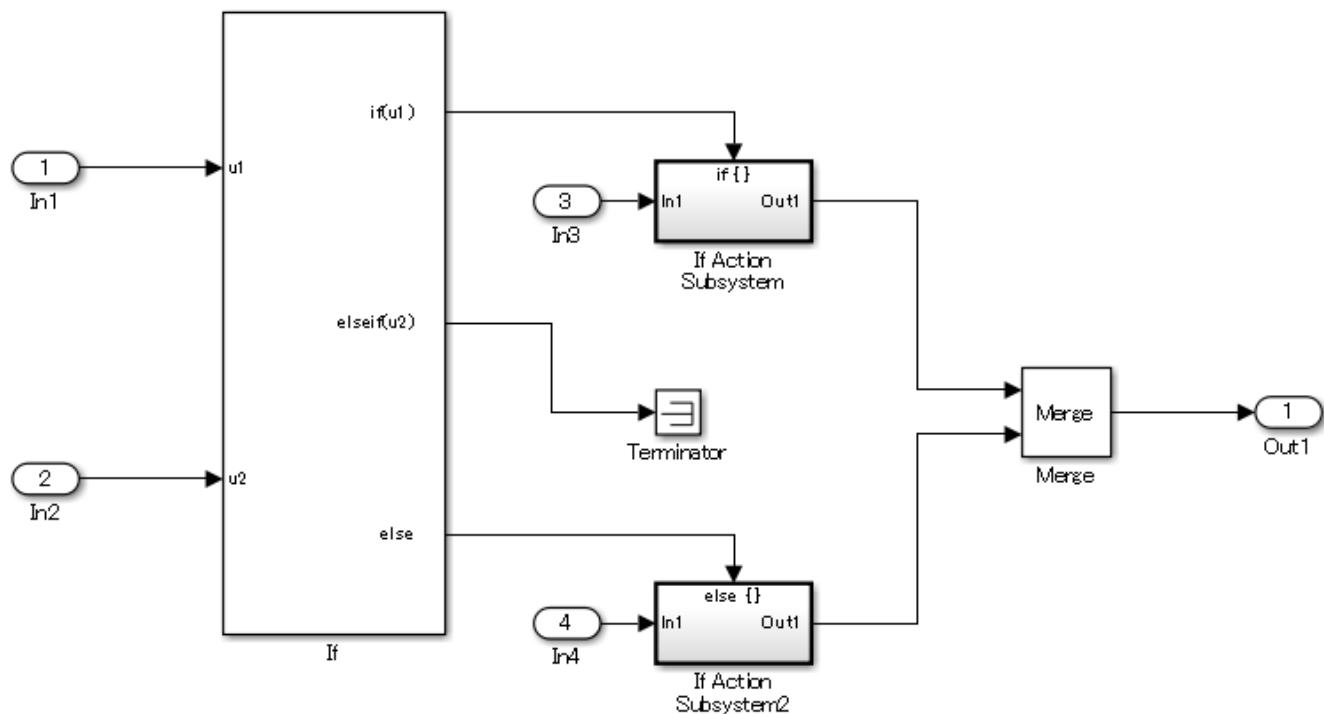
- Past value is retained
- Merge block and a conditional flow block, such as an If or Switch Case block, are used to switch functions.

Custom Parameter

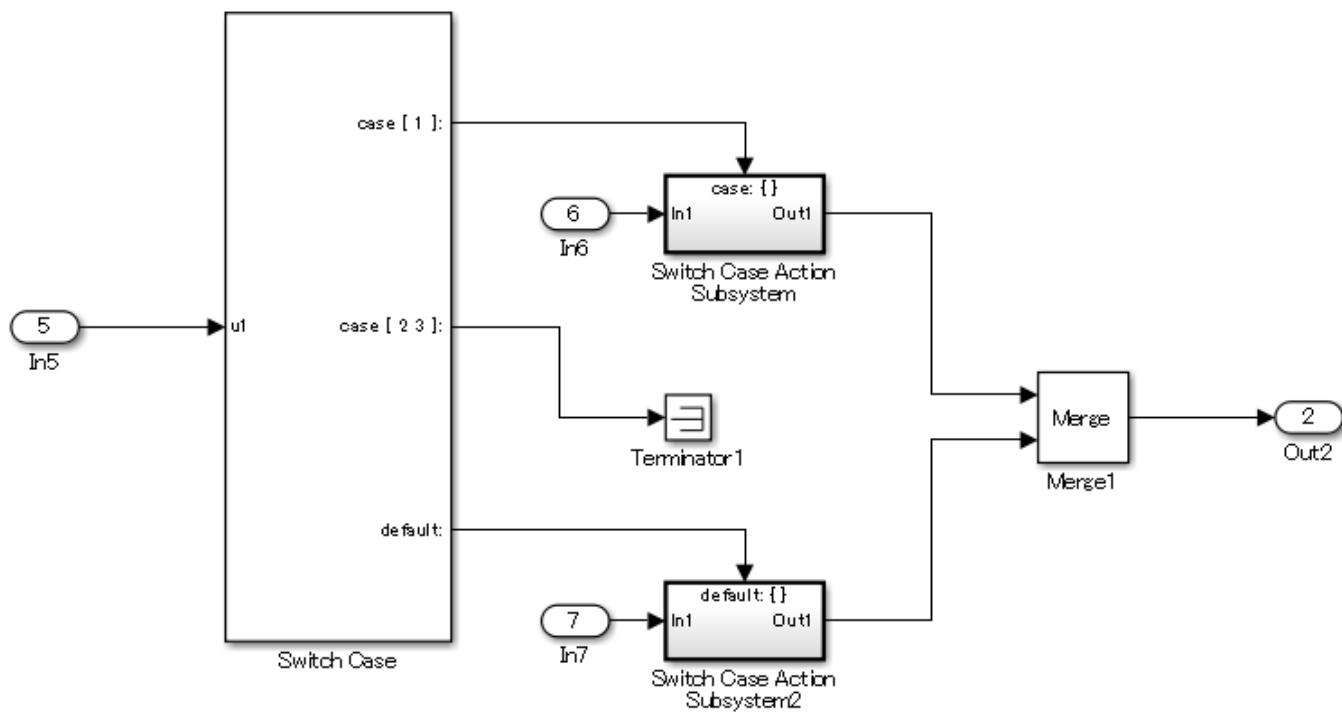
Not Applicable

Example — Correct

If block example

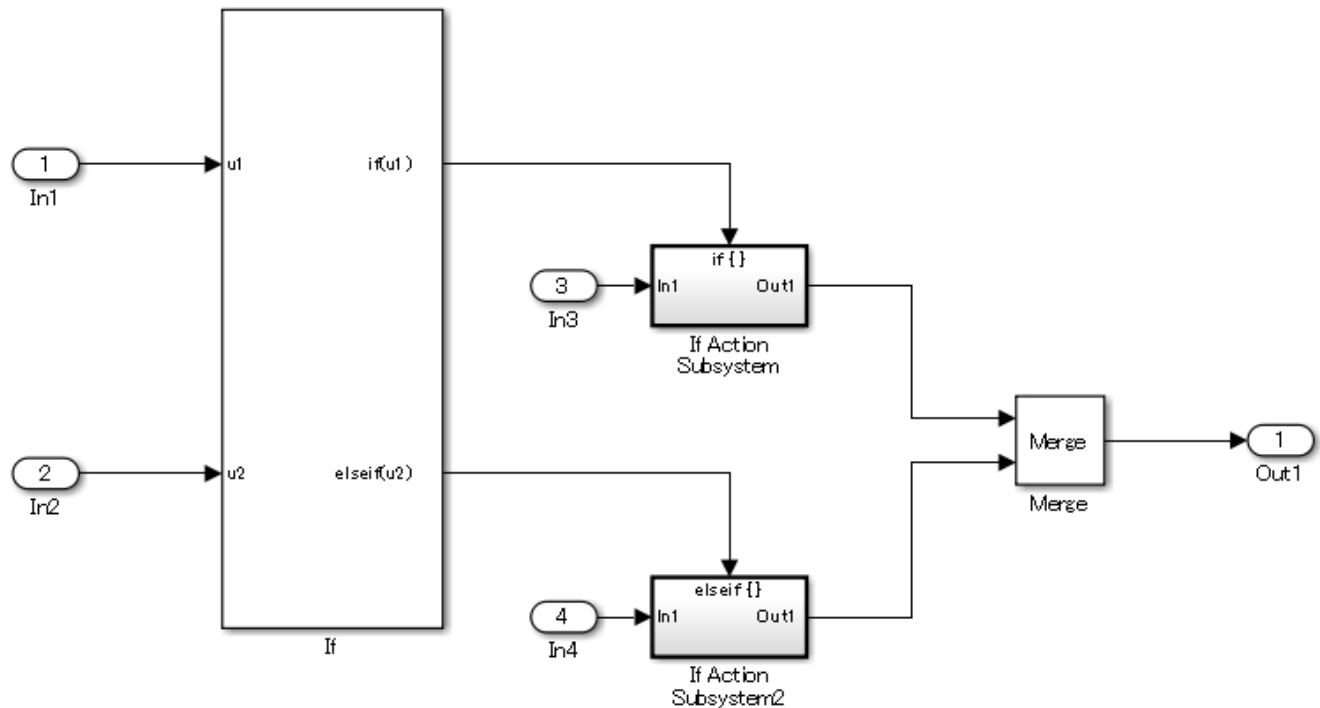


Switch Case block example

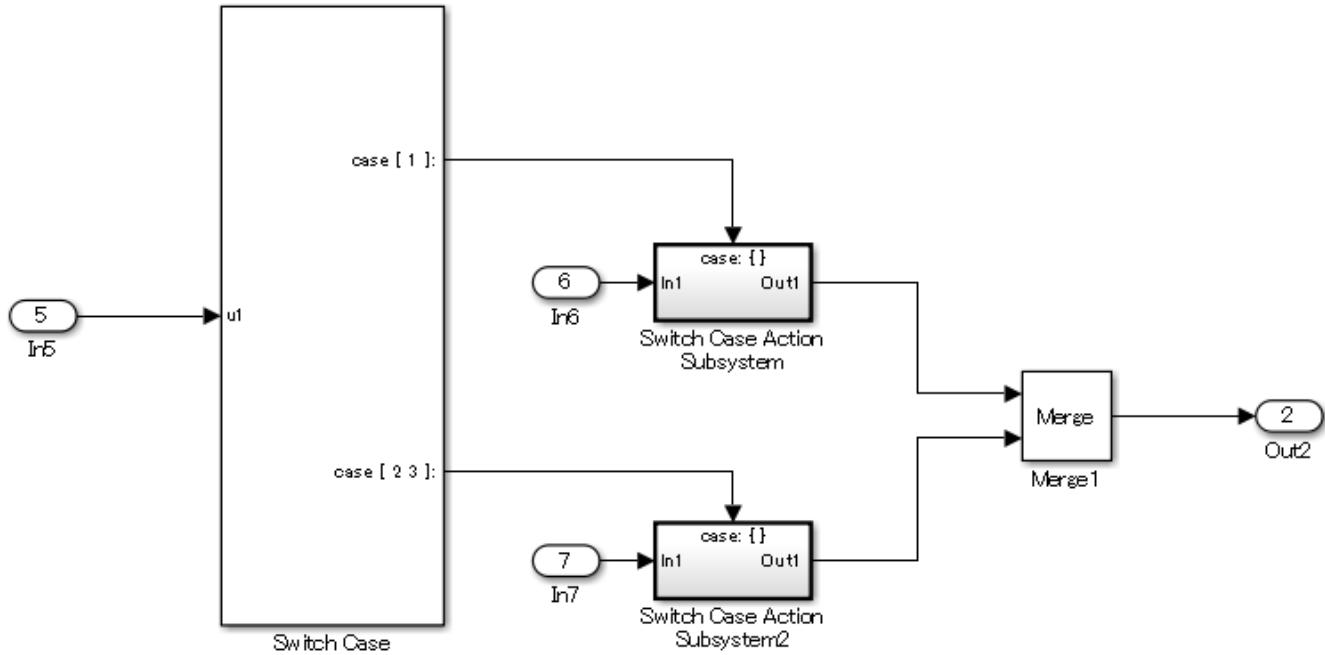


Example — Incorrect

If block example



Switch Case block example



Sub ID a2

A feedback loop using a Delay block shall be implemented when these conditions are met:

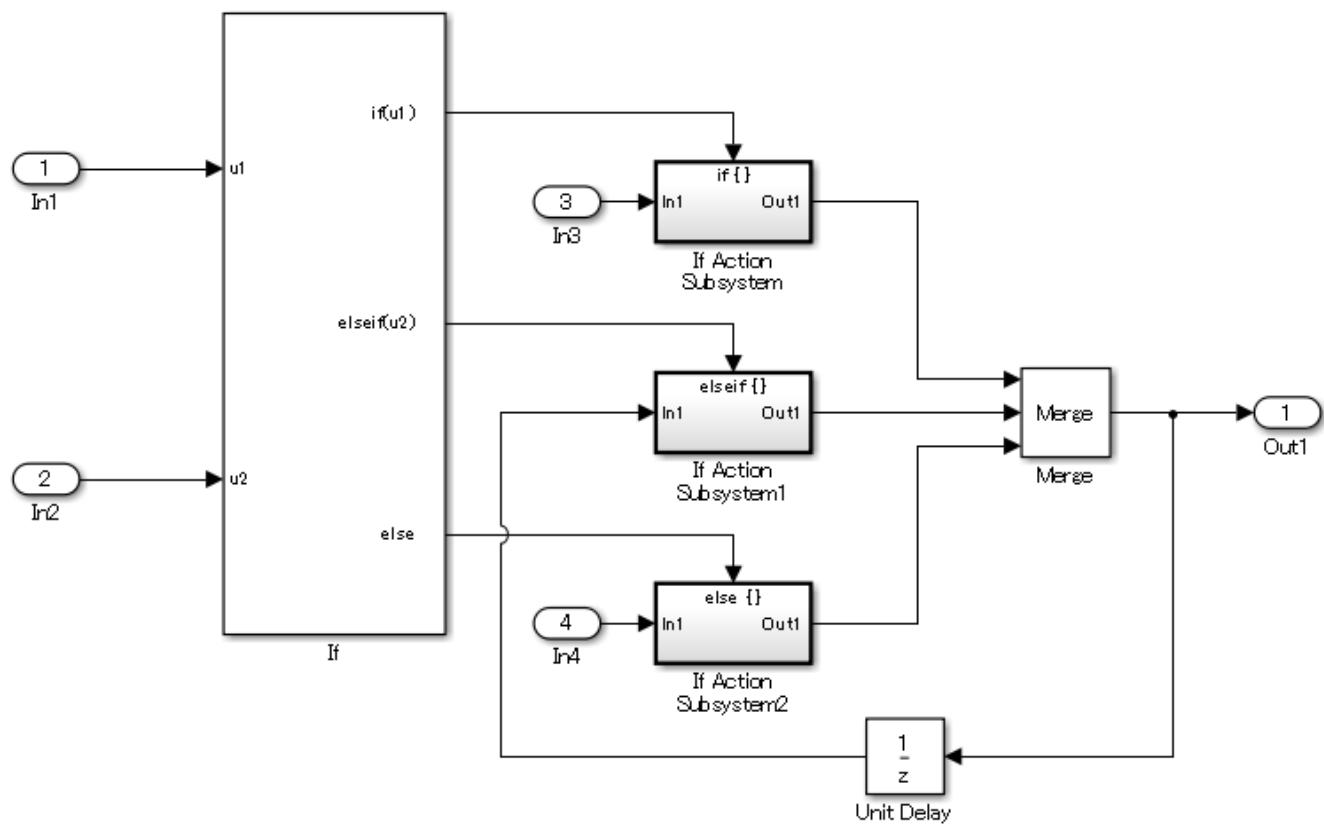
- Past value is retained
- Merge block and a conditional flow block, such as an If or Switch Case block, are used to switch functions.

Custom Parameter

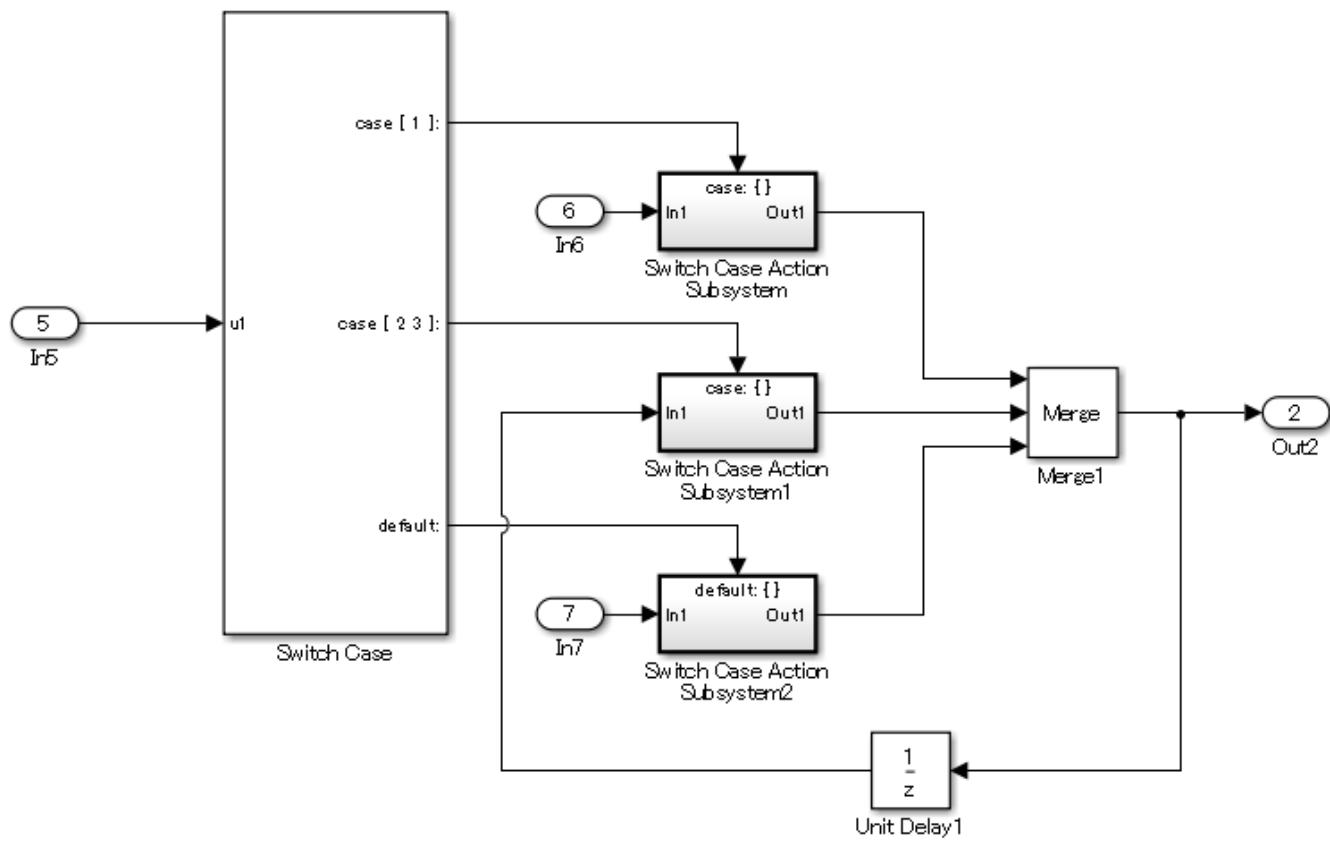
Not Applicable

Example — Correct

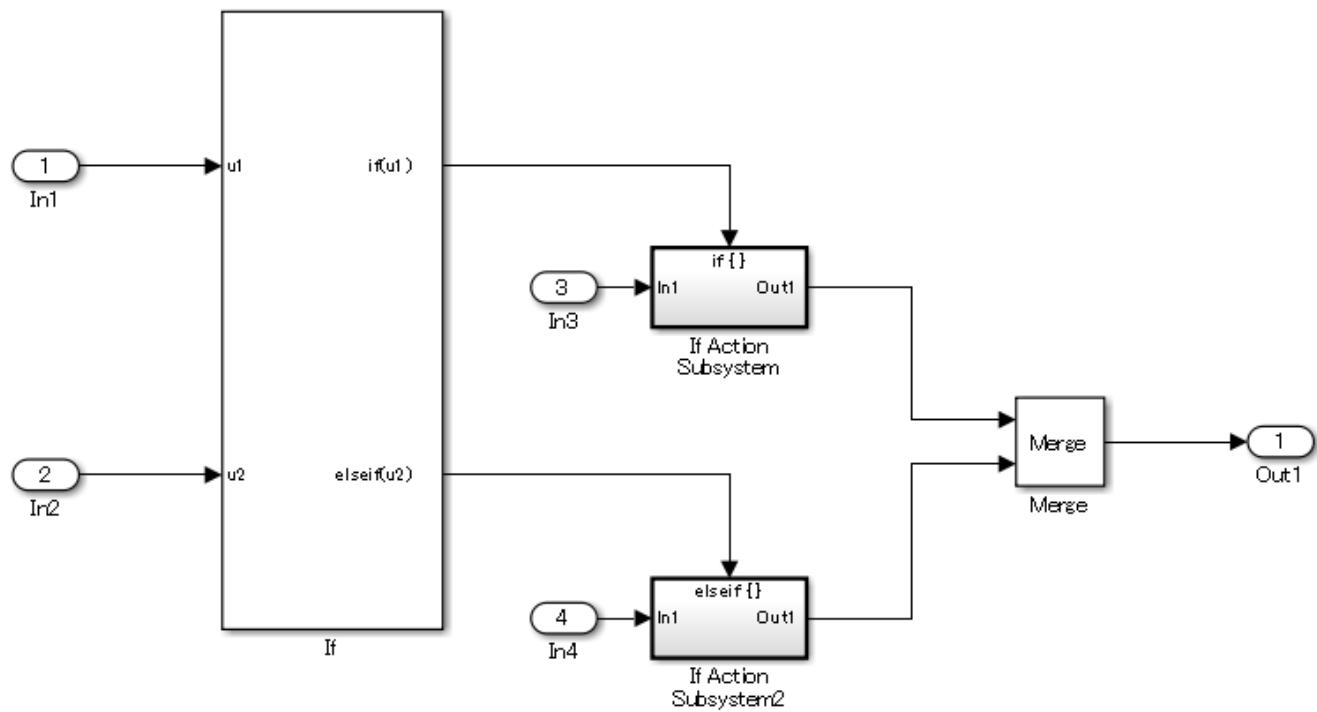
If block example



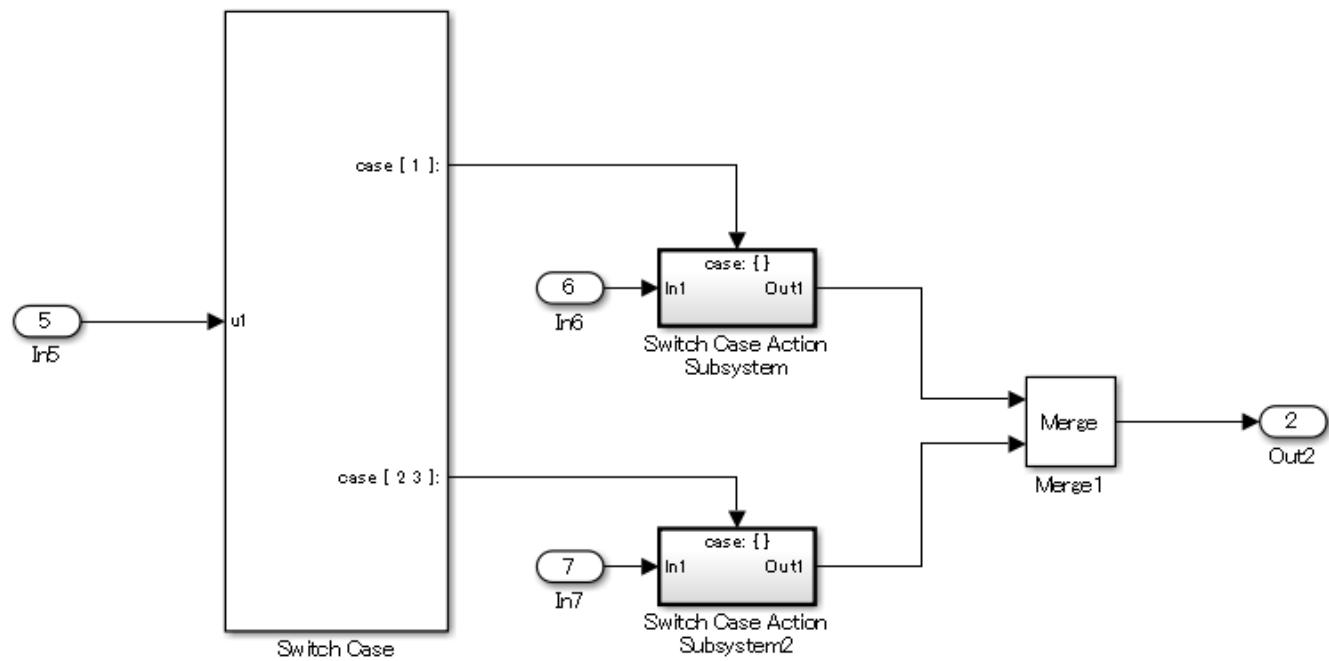
Switch Case block example

**Example — Incorrect**

If block example



Switch Case block example



Rationale

Sub ID a1:

- Improves code efficiency.
- Connections to Terminator block can be used when past values are held other than by the default (`else`).

Sub ID a2:

- Retaining past values is explicit.

Verification

Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

Version History

Introduced in R2020a

Operation Blocks

na_0002: Appropriate usage of basic logical and numerical operations

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

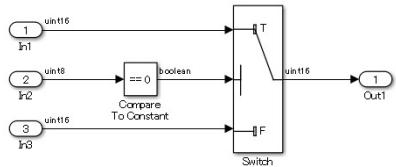
Logical signals shall not connect to blocks that operate on numerical signals.

Custom Parameter

Blocks receiving numerical signals

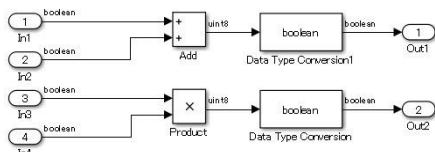
Example — Correct

Numerical values are compared to determine if they are equal.

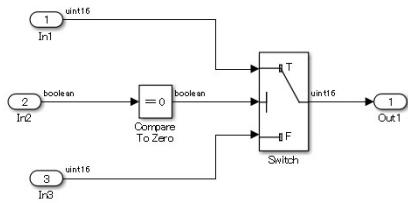


Example — Incorrect

A logical output is connected directly to the input of blocks that process numerical inputs.



A logical signal is compared with a numerical value.



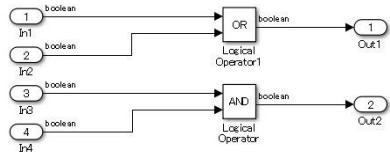
Sub ID b

Numerical signals shall not connect to blocks that operate on logical signals.

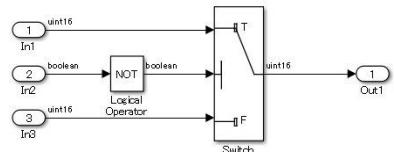
Custom Parameter

Blocks receiving logical signals

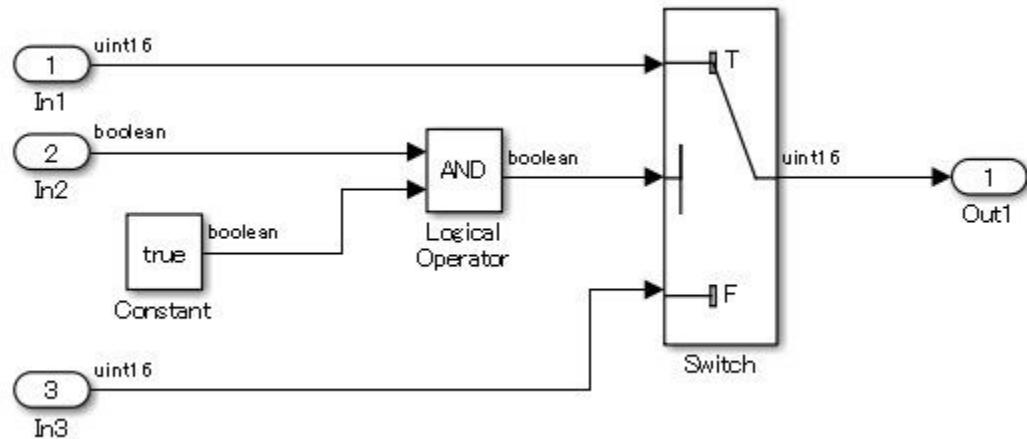
Example — Correct



Logical signal is inverted by using a logical operation.



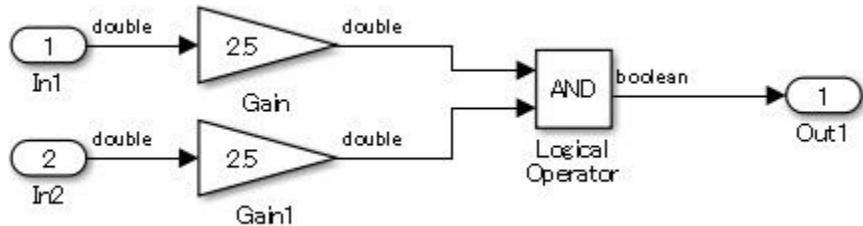
Logical signal is evaluated by using a logical operation.



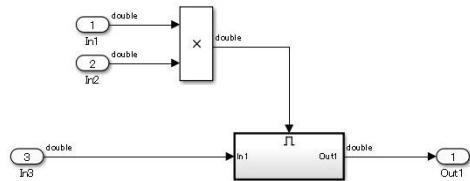
Example — Incorrect

Logical signal is inverted by using a logical operation.

A block that is used to perform logical operations is being used to perform numerical operations. A numerical output is connected to the input of blocks that process logical inputs.



A block that is used to perform numerical operations is being used to perform logical operations. Inputs other than logical values can be provided to the block. However, the Enable port block can receive only logical signals that have On/Off. The Product block performs logical operations when it connects the numerical operations result to a block that receives the logical value Enable port.



Rationale

Sub IDs a, b:

- When numerical and logical values are treated the same, the original intention becomes unclear and the next operation in the model can be incorrectly interpreted, further compounding the error.

Verification

Model Advisor check: "Check fundamental logical and numerical operations" (Simulink Check)

Last Changed

R2020a

See Also

- "Signal Basics"

Version History

Introduced in R2020a

jc_0121: Usage of add and subtraction blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

The icon shape of the add and subtraction Sum block shall be rectangular.

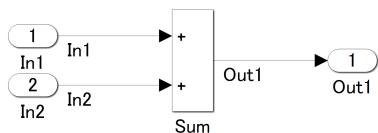
When used in a feedback loop, the *icon shape* can be round.

Custom Parameter

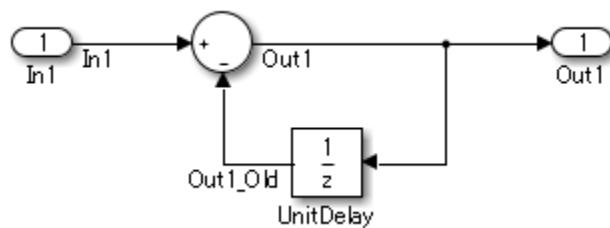
Not Applicable

Example — Correct

The icon shape of the add and subtraction Sum block shall be rectangular.

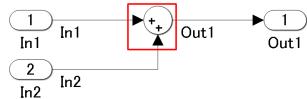


The second input to the add and subtraction Sum block is a feedback loop, so the *icon shape* is round.



Example — Incorrect

This is not a feedback loop, but the *icon shape* of the add and subtractionSum block is round.

**Sub ID b**

The + mark shall be used for the first input to the add and subtraction Sum block.

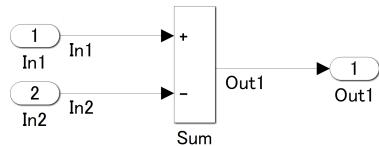
For a feedback loop, the first input can be set by using the - mark.

Custom Parameter

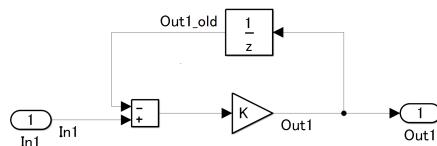
Not Applicable

Example — Correct

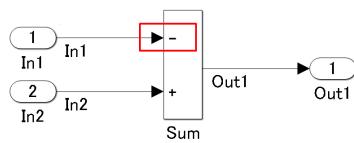
The + mark is used for the first input to the add and subtraction Sum block.



The second input to the add and subtraction Sum block is a feedback loop, so the - mark is used.

**Example — Incorrect**

The sign for the first input to the add and subtraction Sum block is the - mark.

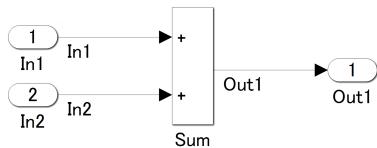
**Sub ID c**

The add and subtraction Sum block shall not have more than two inputs.

Custom Parameter

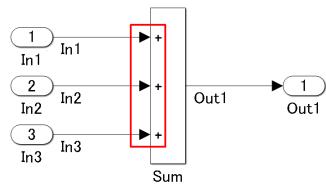
Not Applicable

Example — Correct



Example — Incorrect

The add and subtraction Sum block has three inputs.



Rationale

Sub ID a:

- Adherence to the guideline improves readability of the model.

Sub ID b:

- Readability of the control specification improves when the sign for the first input is consistent.

Sub ID c:

- The order of operations is clearly defined.

Verification

Model Advisor check: "Check usage of Sum blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0610: Operator order for multiplication and division block

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b

MATLAB Versions

All

Rule

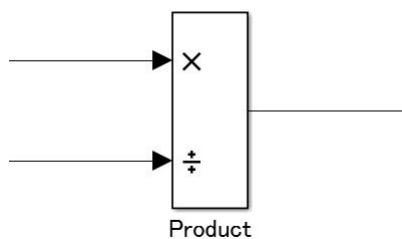
Sub ID a

The * mark shall be used for the first input to a multiplication and division Product block.

Custom Parameter

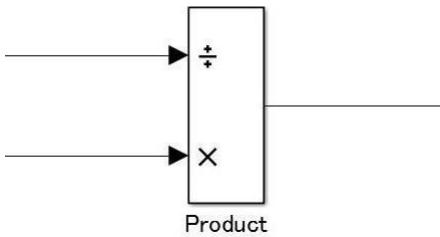
Not Applicable

Example — Correct



Example — Incorrect

The / mark is used for the first input.



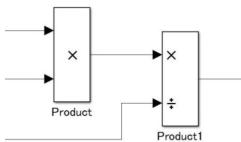
Sub ID b

The multiplication and division Product block shall not have more than two inputs.

Custom Parameter

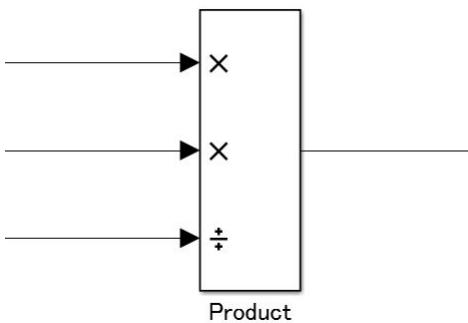
Not Applicable

Example — Correct



Example — Incorrect

The block has three inputs.



Rationale

Sub ID a:

- When checking the block, the input order of the expression and block is reversed, which impairs readability.
- For floating point numbers, the code is generated according to the operation order in the block -- $((1 \div 1\text{st input})) \times 2\text{nd input}$. However, if division is performed later, then the number of operations can be reduced.

Sub ID b:

- The order of operations is clearly defined.

Verification

Model Advisor check: "Check operator order of Product blocks" (Simulink Check)

Last Changed

R2020a

See Also

- "Multiply and Divide Inputs Using the Product Block"

Version History

Introduced in R2020a

jc_0611: Input sign for multiplication and division blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

When using fixed-point values as the input to the multiplication and division Product block, the sign of the data type shall be the same for all input signals.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- A utility function is created for each least significant bit (LSB) when fixed-point code is generated. Unification of data type signs can reduce the number of utility functions.

Verification

Model Advisor check: "Check signs of input signals in product blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Multiply and Divide Inputs Using the Product Block”

Version History

Introduced in R2020a

jc_0794: Division in Simulink

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

When using division, implementation of the algorithm shall avoid division by zero.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Deviation from the rule can cause unintended operation and code generation results.

Model Advisor Check

“Check for division by zero in Simulink” (Simulink Check)

Last Changed

R2021a

See Also

- Multiply “Multiply and Divide Inputs Using the Product Block”

Version History

Introduced in R2020a

jc_0805: Numerical operation block inputs

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1/a2, b, c1/c2, d, e, f1/f2, g, h, i, j
- JMAAB — a1/a2, b, c1/c2, d, e, f1/f2, g, h, i, j

MATLAB Versions

All

Rule

Sub ID a1

When using and Abs block with signed integer types, the input shall not be the most negative value.

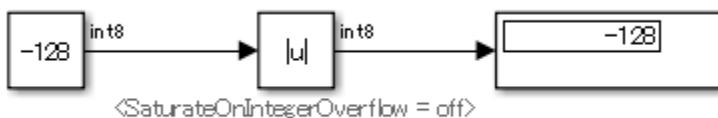
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect

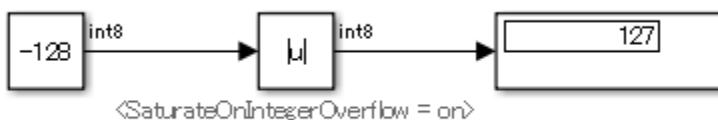


Sub ID a2

Abs block parameter **Saturation on Integer Overflow** shall be selected.

Custom Parameter

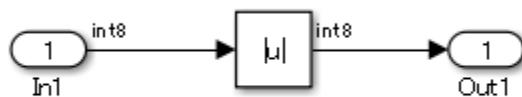
Not Applicable

Example — Correct**Example — Incorrect****Sub ID b**

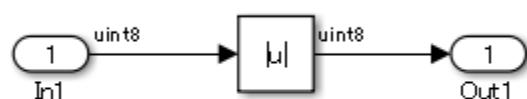
Input to the Abs block shall not be unsigned integer types or fixed-point types.

Custom Parameter

Not Applicable

Example — Correct

....

Example — Incorrect

....

Sub ID c1

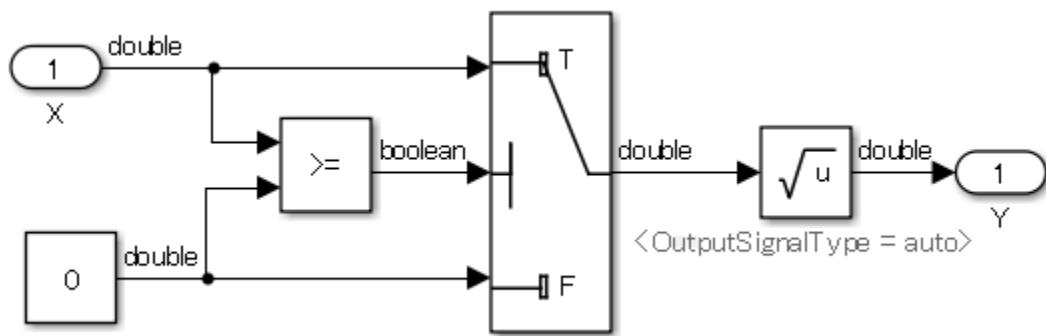
Input to the Sqrt block shall not be a negative value.

Custom Parameter

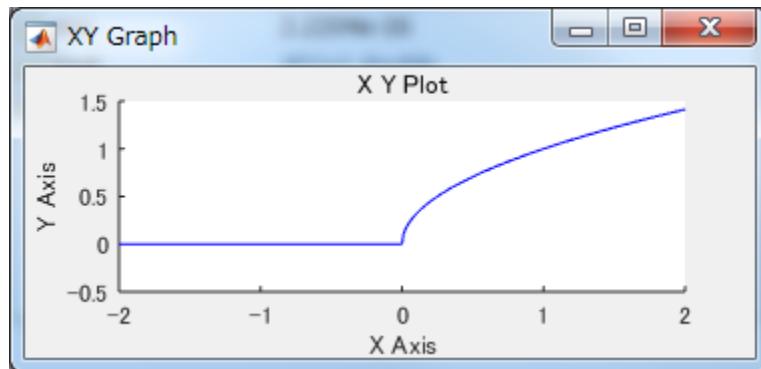
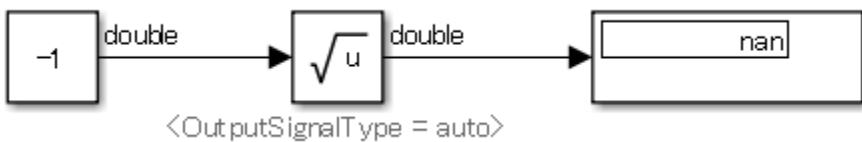
Not Applicable

Example — Correct

Negative number is saturated with 0.



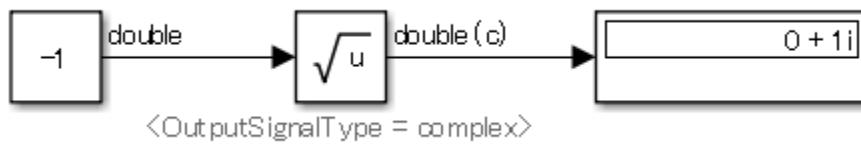
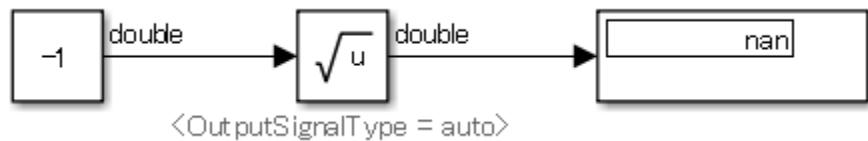
Simulation result

**Example — Incorrect****Sub ID c2**

Sqrt block parameter **Output Signal Type** shall be set to **complex**.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Sub ID d**

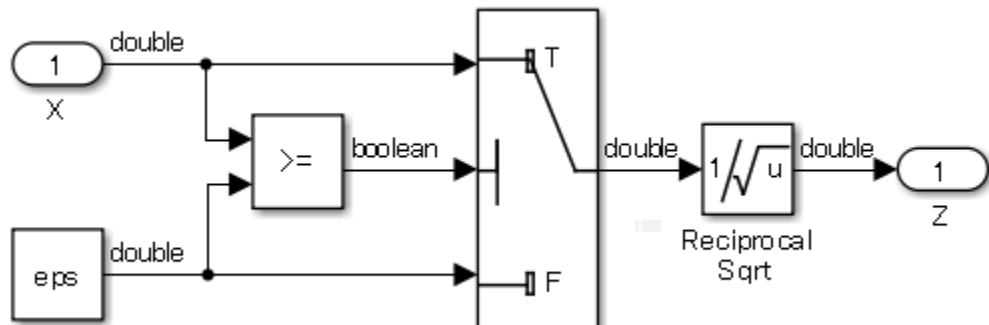
Input to the Reciprocal Sqrt block shall not be less than zero.

Custom Parameter

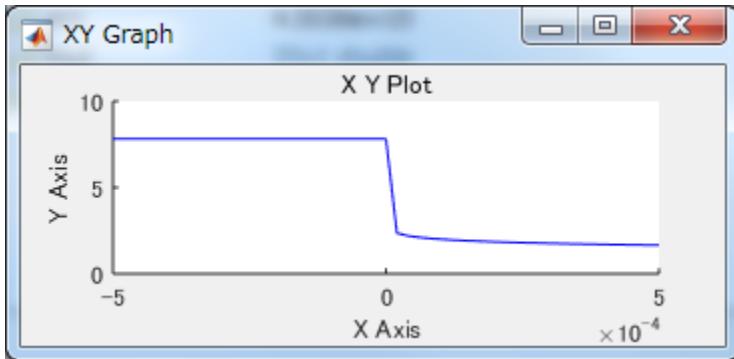
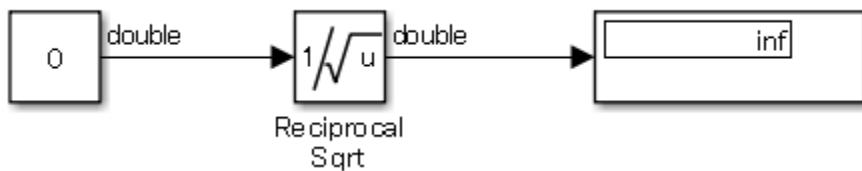
Not Applicable

Example — Correct

Less than eps saturated with eps.



Simulation result: Plot as Y=log10(Z)

**Example — Incorrect****Sub ID e**

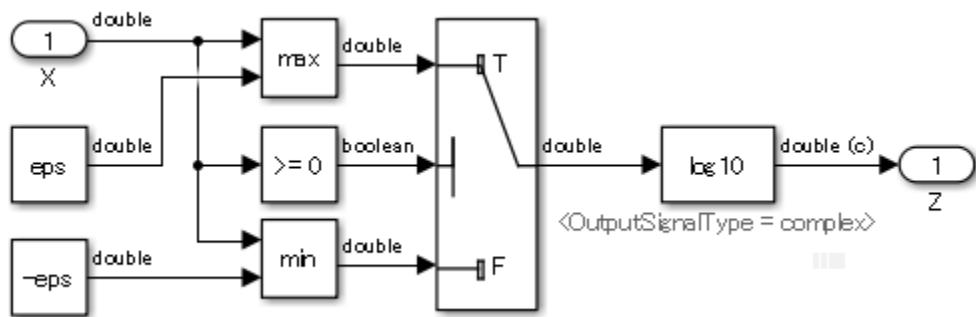
When Math Function block parameter **Function** is set to **log** or **log10**, the input to the block shall not be zero.

Custom Parameter

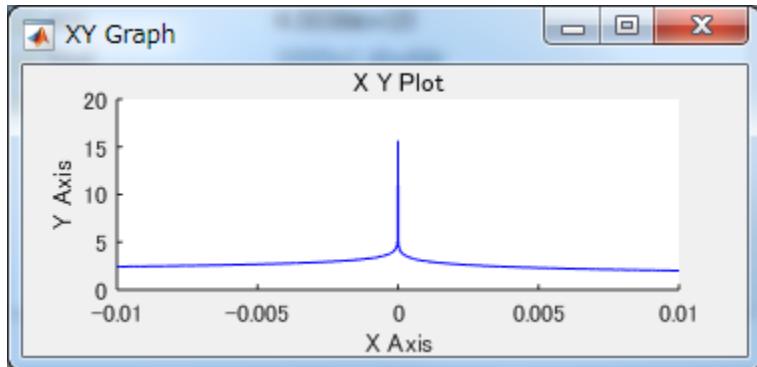
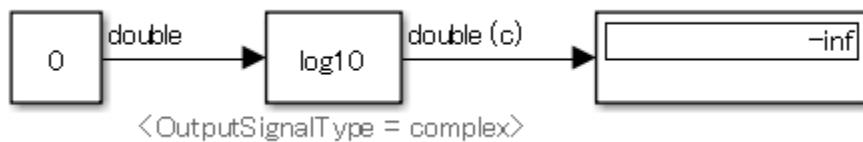
Not Applicable

Example — Correct

Replace within $\pm \text{eps}$ with $\pm \text{eps}$



Simulation result: Plot as $Y = |Z|$

**Example — Incorrect****Sub ID f1**

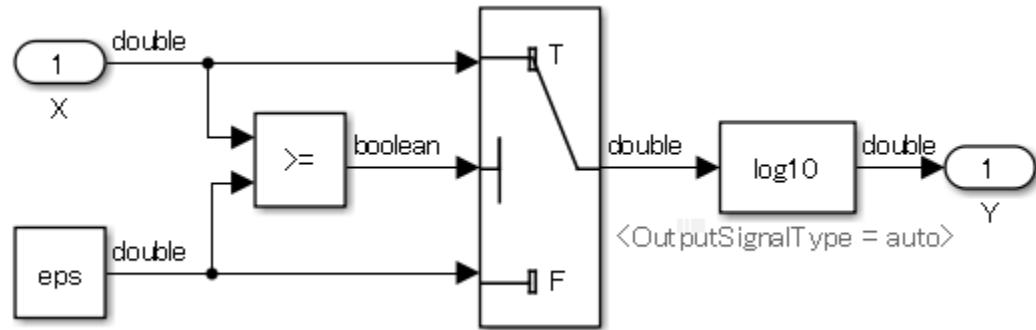
When Math Function block parameter **Function** is set to **log** or **log10**, the input to the block shall not be a negative number.

Custom Parameter

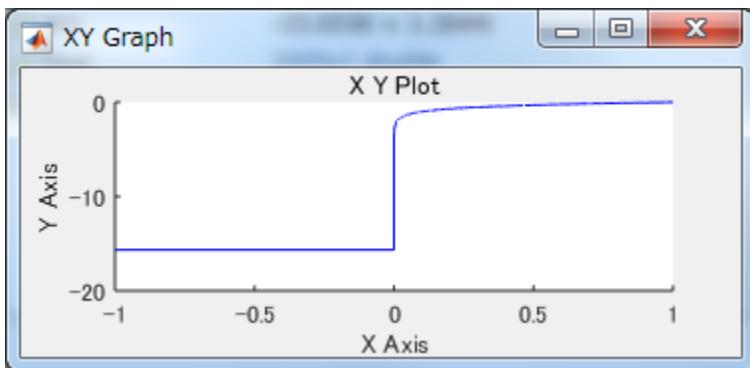
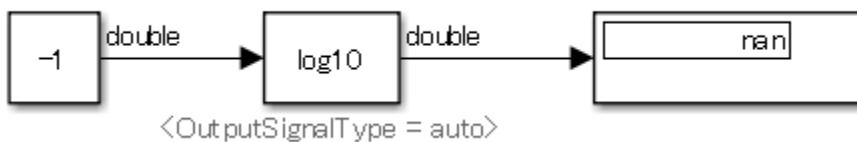
Not Applicable

Example — Correct

When the input is less than eps, the value is saturated to eps. Less than eps saturated with eps.



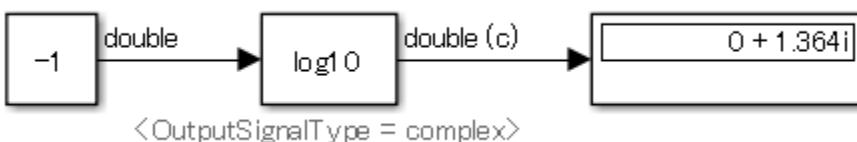
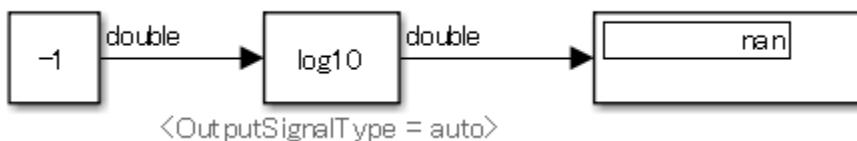
Simulation result

**Example — Incorrect****Sub ID f2**

When Math Function block parameter **Function** is set to **log** or **log10**, block parameter **Output Signal Type** shall be set to **complex**.

Custom Parameter

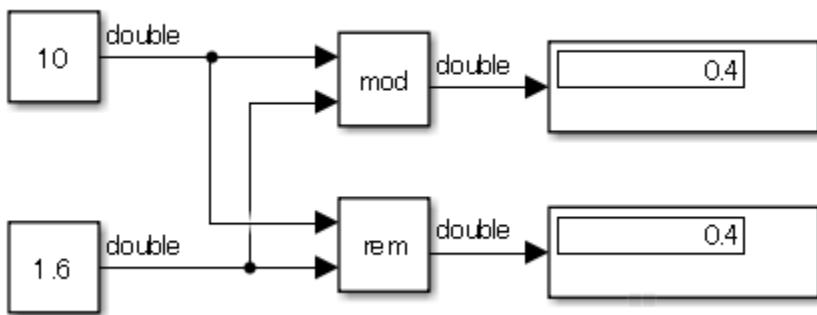
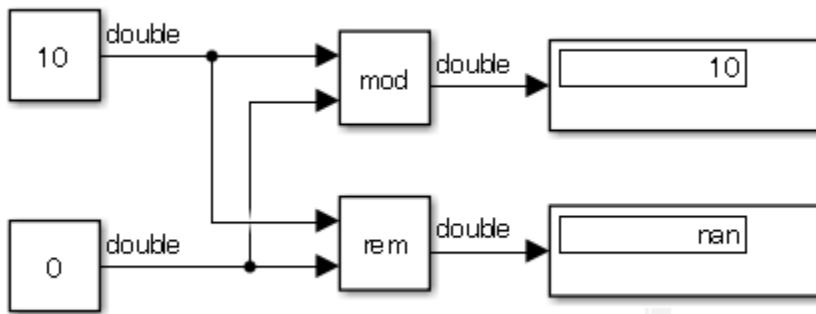
Not Applicable

Example — Correct**Example — Incorrect****Sub ID g**

When Math Function block parameter **Function** is set to **mod** or **rem**, the second argument input shall not be zero.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Sub ID h**

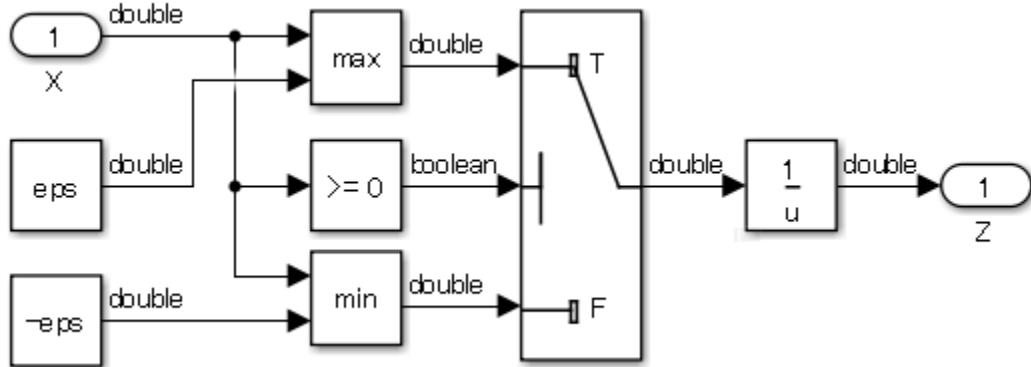
When Math Function block parameter **Function** is set to **reciprocal**, the input to the block shall not be zero.

Custom Parameter

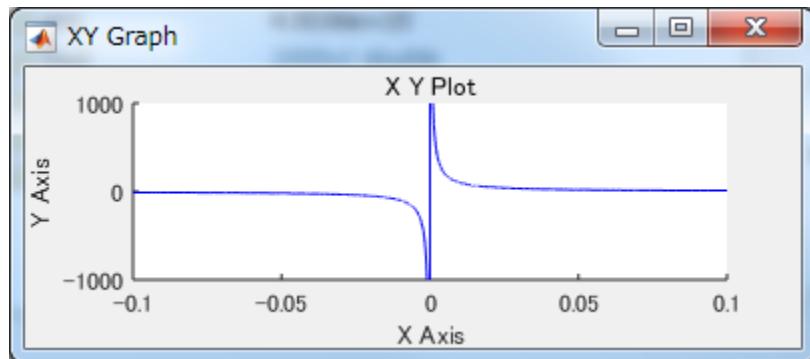
Not Applicable

Example — Correct

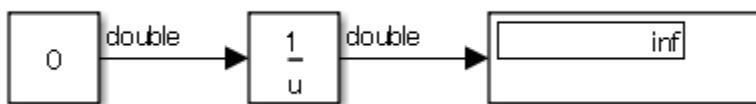
Replace within $\pm \text{eps}$ with $\pm \text{eps}$



Simulation result: Simulation results is not inf, but since it is close to zero, the change in the output value is significant.



Example — Incorrect



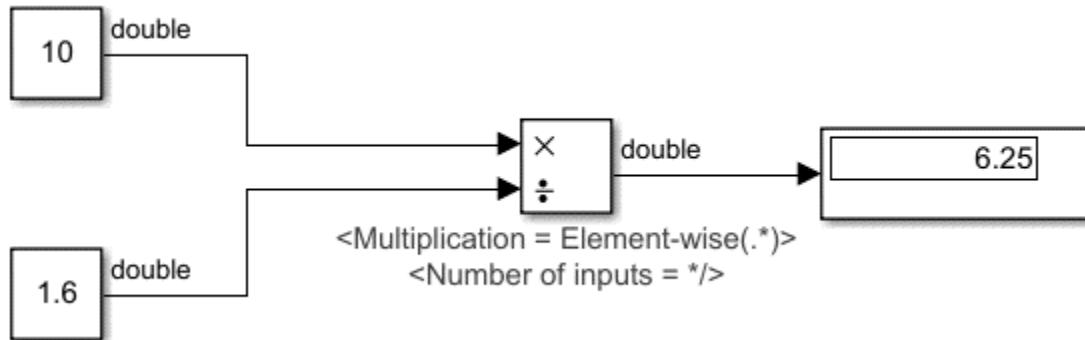
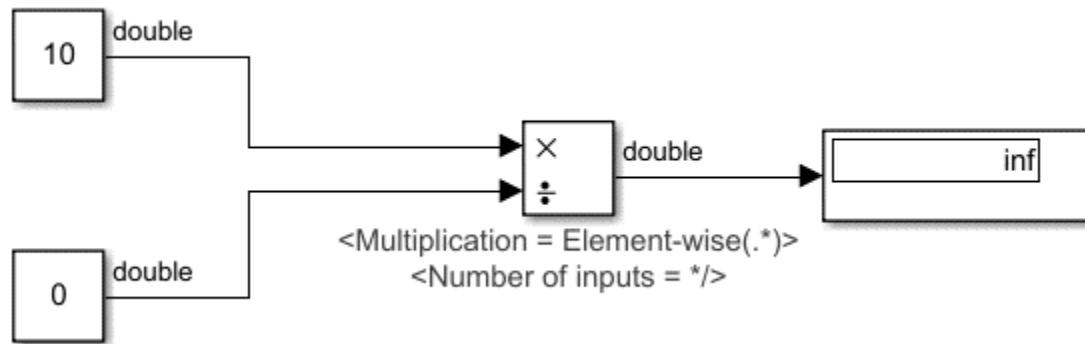
Sub ID i

When Product block parameter **Multiplication** is set to **Element-wise (.*)**, the divisor input shall not be zero.

Note To specify a divisor input, set Product block parameter **Number of inputs** to ***/**.

Custom Parameter

Not Applicable

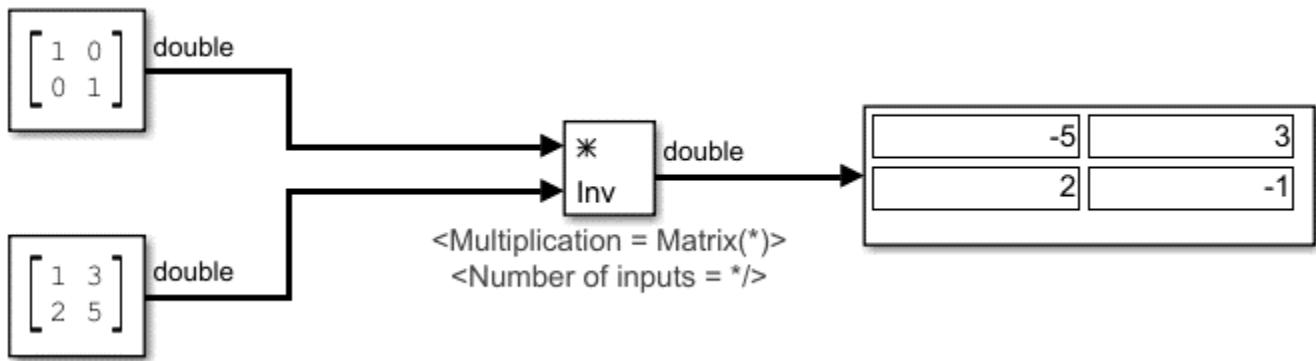
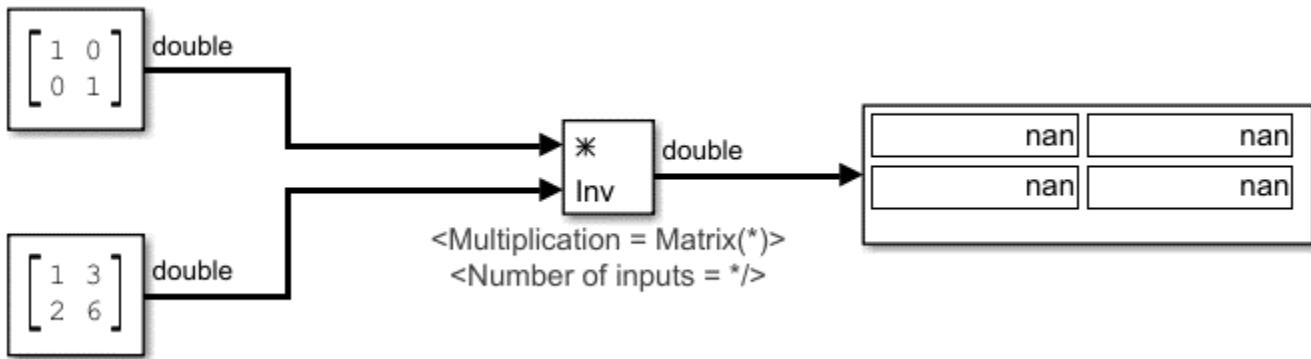
Example — Correct**Example — Incorrect****Sub ID j**

When Product block parameter **Multiplication** is set to **Matrix(*)**, the divisor input shall not be set to a singular matrix.

Note To specify a divisor input, set Product block parameter **Number of inputs** to ***/**.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Rationale**

Sub IDs a1, c1, d, e, f1, g, h, i, j:

- The result of entering an invalid value is implementation dependent. Deviation from the rules can result in unintended behavior.

Sub ID a2:

- Correct settings prevent unintended behavior that can result from using invalid values.

Sub ID b:

- The block can become optimized out of the generated code, resulting in a block that you cannot trace to the generated code.

Sub IDs c2, f2:

- Correct settings prevent unintended behavior that can result from using negative values.

Verification

Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

See Also

- “Specify Block Properties”
- “Control Data Types of Signals”

Version History

Introduced in R2020a

jc_0622: Usage of Fcn blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

When a Fcn block has operators with different priorities, parentheses shall be used to specify the priority order.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- When operators have different priorities and the computation order is not clearly specified by using parentheses, readability is impaired and can be misinterpreted. This can result in unintended behavior.

Verification

Model Advisor check: "Check for parentheses in Fcn block expressions" (Simulink Check)

Last Changed

R2020a

See Also

- “Algebraic Loop Concepts”

Version History

Introduced in R2020a

jc_0621: Usage of Logical Operator blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

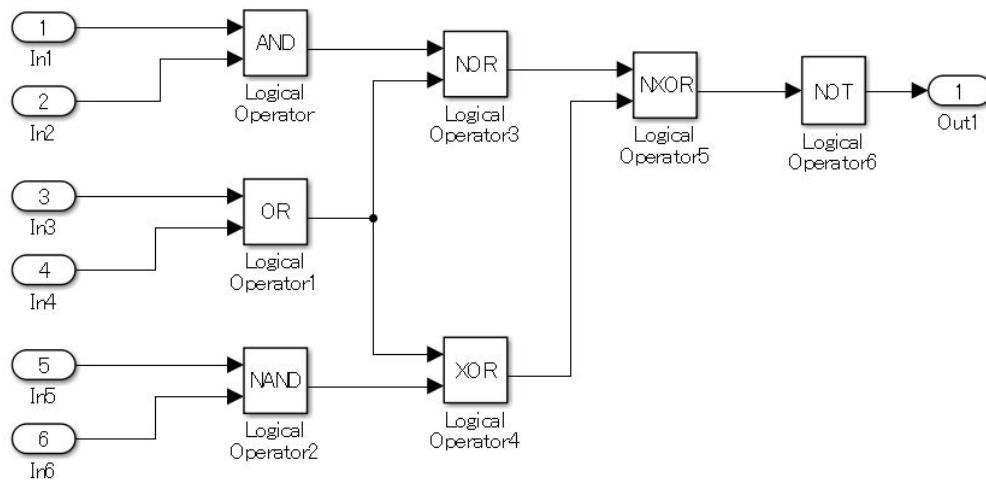
Sub ID a

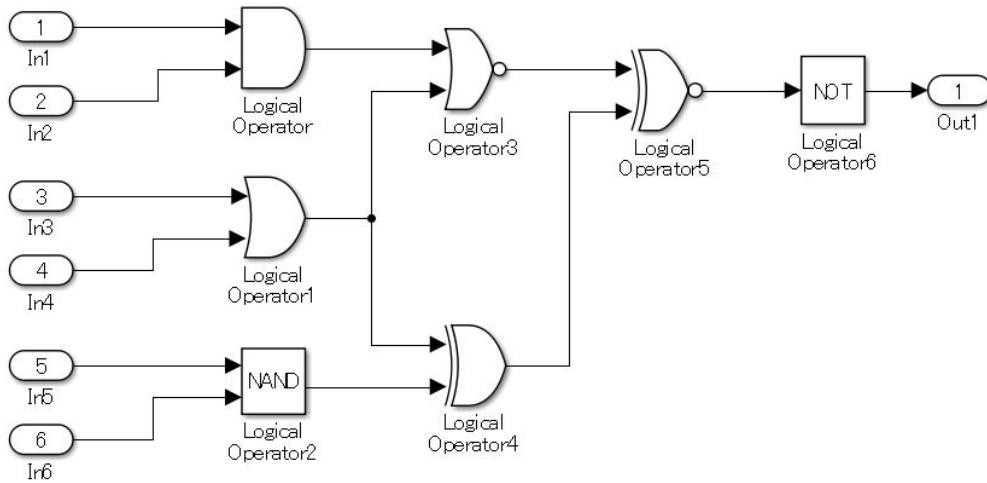
The **icon shape** for the Logical Operator block shall be set to rectangular.

Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect**Rationale**

Sub ID a:

- When describing the same function, using a consistent expression improves readability. Since "characteristics" shapes are similar, the risk of misinterpretation is greater than with rectangular shapes.

Verification

Model Advisor check: "Check icon shape of Logical Operator blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0131: Usage of Relational Operator blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

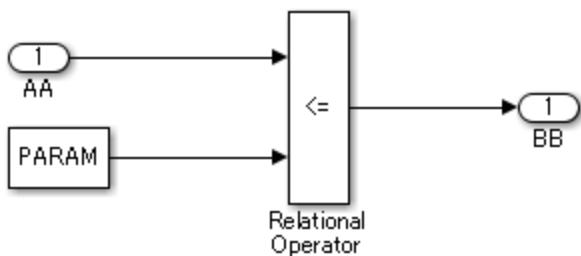
Sub ID a

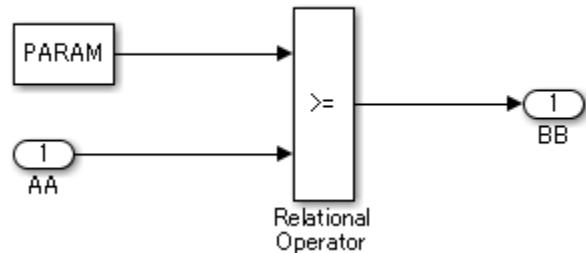
When using a Relational Operator block for comparison of signals and constants, the second (bottom) input shall be used as the constant input.

Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect**Rationale**

Sub ID a:

- Using constant values and the same comparison method reduces misinterpretation of the model.

Verification

Model Advisor check: "Check usage of Relational Operator blocks" (Simulink Check)

Last Changed

R2020a

See Also

- "Relational and Logical Operators" (Embedded Coder)

Version History

Introduced in R2020a

jc_0800: Comparing floating-point types in Simulink

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

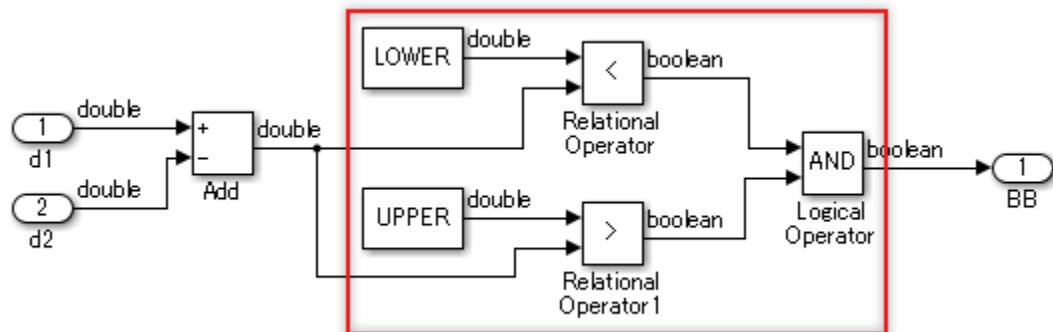
Sub ID a

Equivalence comparison operators ($==$, $\sim=$) shall not be used on floating-point data types.

Custom Parameter

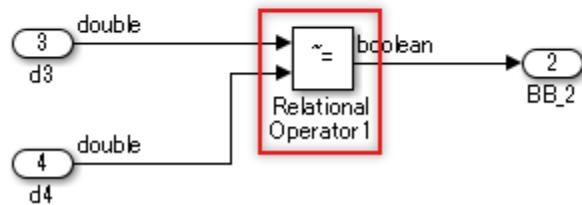
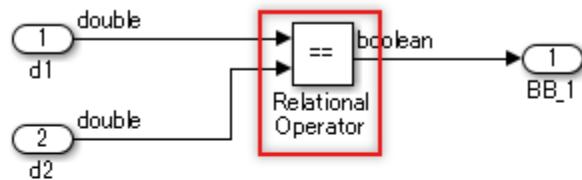
Not Applicable

Example — Correct



Example — Incorrect

Uses equivalence comparison operators $==$ and $\sim=$ on the floating-point data type.



Rationale

Sub ID a:

- Due to the characteristics of the floating-point, since the error is included in the value, the result of the equivalence comparison operation may be false when it was expected to be true.

Verification

Model Advisor check: "Check comparison of floating point types in Simulink" (Simulink Check)

Last Changed

R2020a

See Also

- "Validate a Floating-Point Embedded Model"

Version History

Introduced in R2020a

jc_0626: Usage of Lookup Table blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

The Lookup Table Dynamic block parameter **Lookup Method** shall be set to **Interpolation – Use End Values**.

Custom Parameter

Not Applicable

Sub ID b

These n-D Lookup Table block parameters shall be set as follows:

- Set **Interpolation Method** to **Linear point-slope** or **Linear Lagrange**.
- Set **Extrapolation Method** to **Clip**.
- Select **Use last table value for inputs at or above last breakpoint**.

Custom Parameter

Not Applicable

Rationale

Sub IDs a, b:

- When an unexpected value is entered for the Lookup Table block, the output is determined by using the extrapolation method and can become an impossible value or cause the Lookup Table output to overflow.

Verification

Model Advisor check: “Check usage of Lookup Tables” (Simulink Check)

Last Changed

R2020a

See Also

- “Methods for Approximating Function Values”

Version History

Introduced in R2020a

jc_0623: Usage of continuous-time Delay blocks and discrete-time Delay blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Unit Delay or Delay blocks shall be used in a discrete model or subsystem.

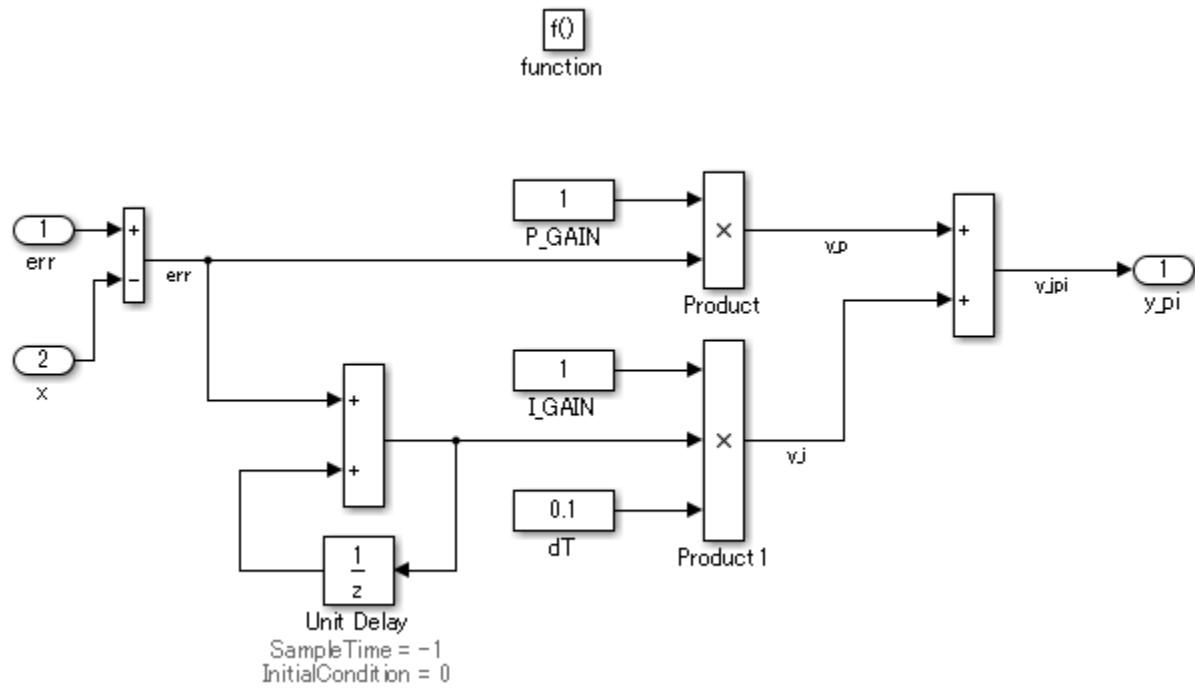
A Memory block shall be used in a continuous type model or subsystem.

Custom Parameter

Not Applicable

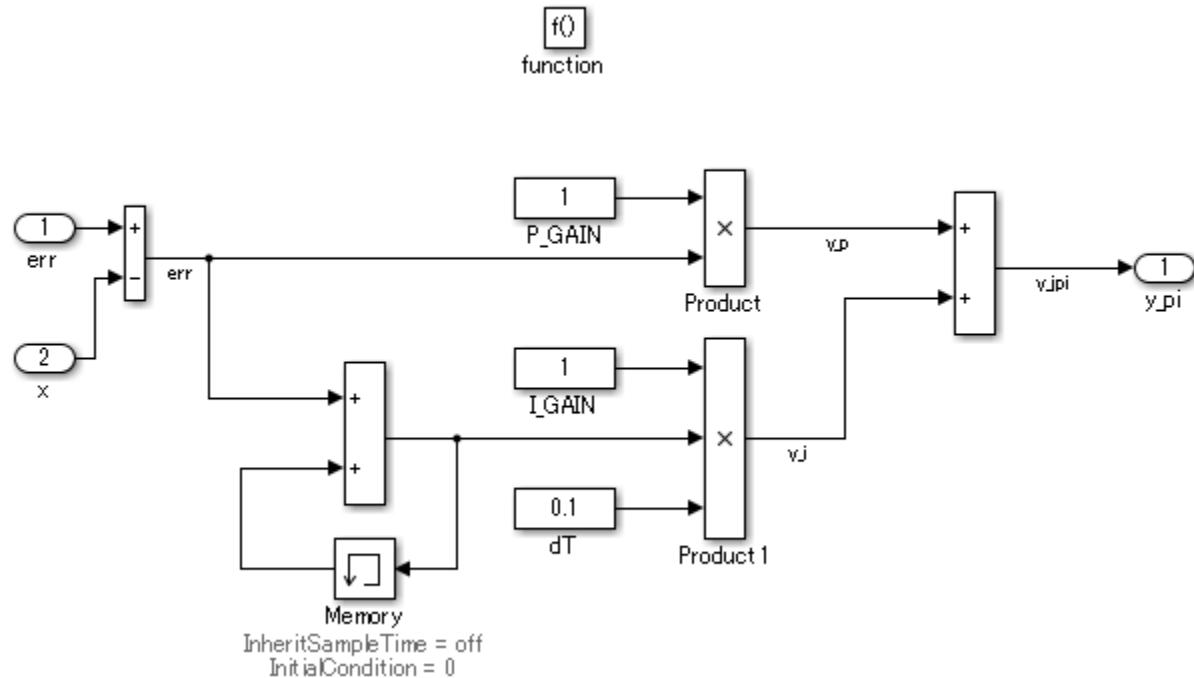
Example — Correct

A Unit Delay block is used in the discrete type model.



Example — Incorrect

A Memory block is used in the discrete type model.



Rationale

Sub ID a:

- Adherence to the rule improves readability of the model.

Verification

Model Advisor check: "Check usage of Memory and Unit Delay blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Explore Types of Subsystems”
- “Model a Continuous System”
- “Discrete and Continuous Resettable Subsystems”

Version History

Introduced in R2020a

jc_0624: Usage of Tapped Delay blocks/Delay blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b

MATLAB Versions

All

Rule

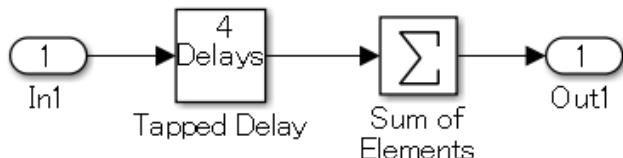
Sub ID a

When holding previous past values, Tapped Delay block shall be used to create a vector signal from all held values.

Custom Parameter

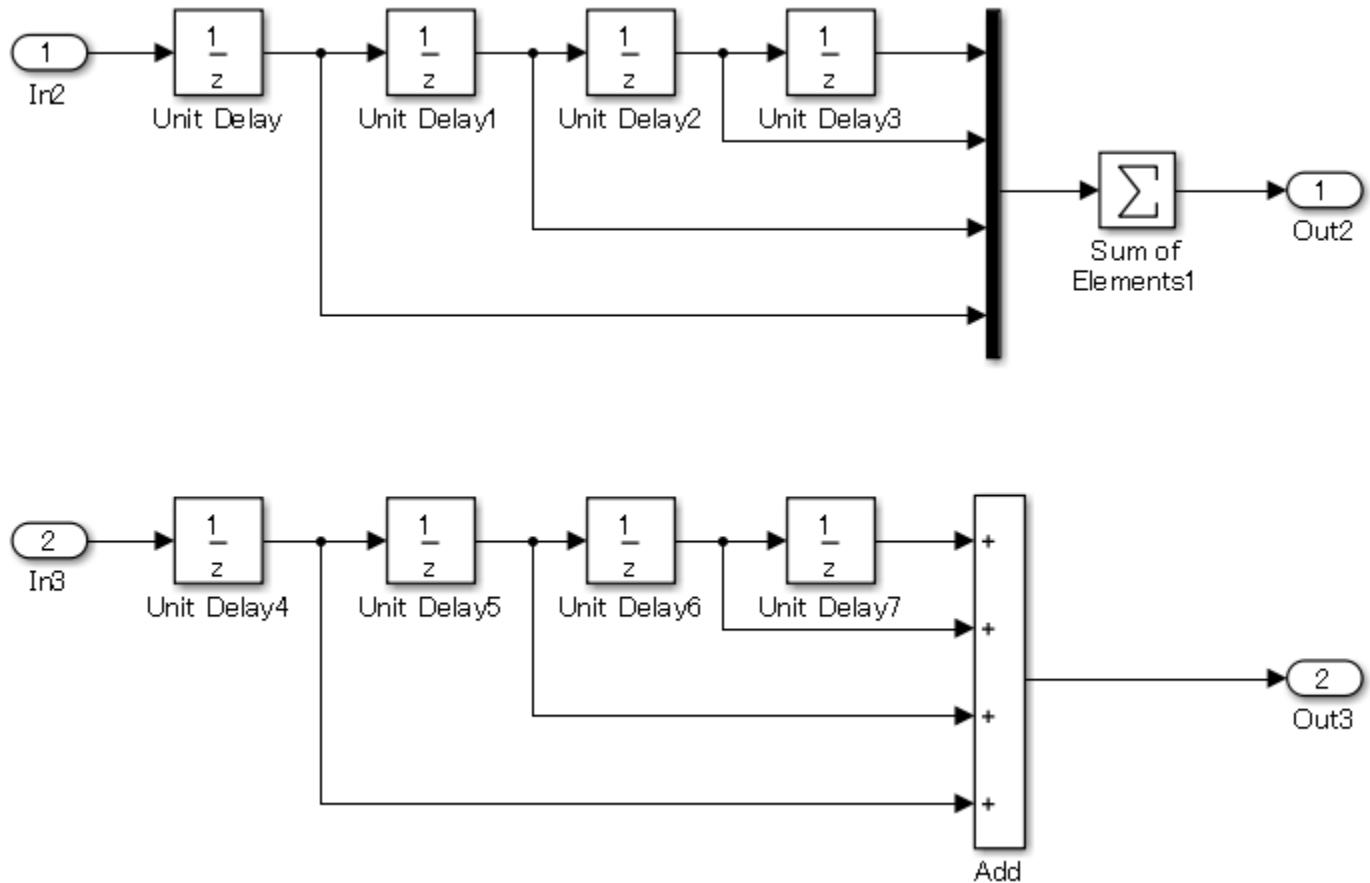
Not Applicable

Example — Correct



Example — Incorrect

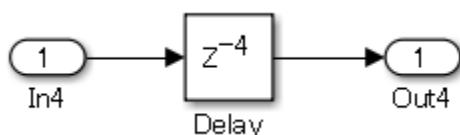
The Tapped Delay block is not used.

**Sub ID b**

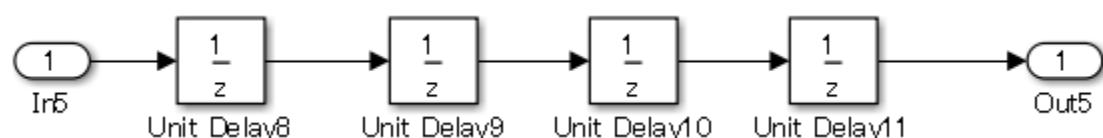
When holding past values, Delay block shall be used to obtain the oldest value only

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

The Delay block is not used.



Rationale

Sub ID a:

- Tapped Delay is set with arrays that hold past values, which improves code readability to assist code efficiency.

Sub ID b:

- Improves model readability and code efficiency.

Verification

Model Advisor check: "Check for cascaded Unit Delay blocks" (Simulink Check)

Last Changed

R2020a

See Also

- "Signal Lines"

Version History

Introduced in R2020a

jc_0627: Usage of Discrete-Time Integrator blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

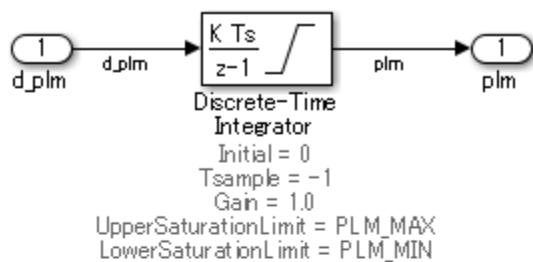
Sub ID a

Discrete-Time Integrator block parameters **Upper saturation limit** and **Lower saturation limit** shall be defined.

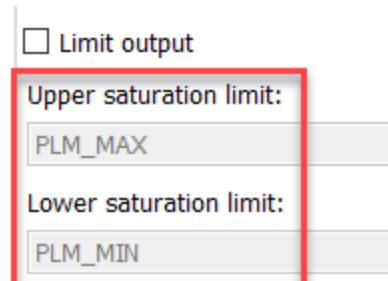
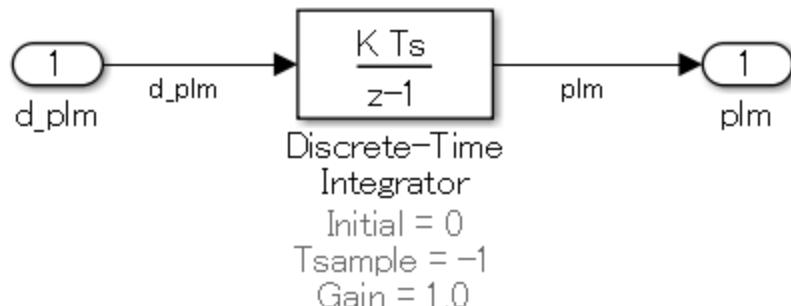
Custom Parameter

Not Applicable

Example — Correct



<input checked="" type="checkbox"/> Limit output
Upper saturation limit:
PLM_MAX
Lower saturation limit:
PLM_MIN

Example — Incorrect

Note When Discrete-Time Integrator block parameters **Upper saturation limit** and **Lower saturation limit**, are defined with parameter objects such as `Simulink.Parameter`, the parameter **Data type** should be set to `auto`. Simulation errors occur when **Data type** is set to a value other than `auto`, `single`, or `double`.

Rationale

Sub ID a:

- Avoids block output overflow and prevents other computation blocks that use the output of this block from producing unexpected results.

Verification

Model Advisor check: "Check usage of Discrete-Time Integrator block" (Simulink Check)

Last Changed

R2024b

See Also

- `Simulink.Parameter`

Version History

Introduced in R2020a

jc_0628: Usage of Saturation blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Saturation and Saturation Dynamic blocks shall be used to limit physical quantity.

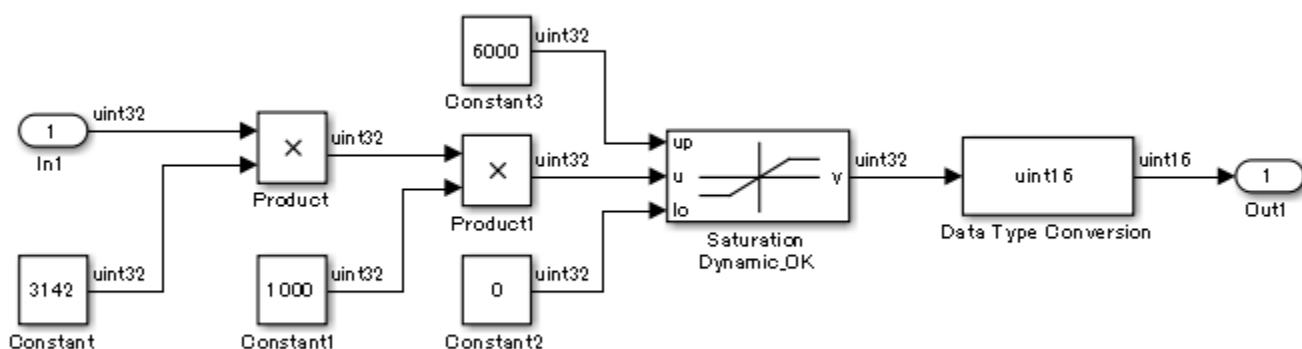
Type conversion shall not be used. Block parameters **Upper limit** and **Lower limit** shall not be set.

Custom Parameter

Not Applicable

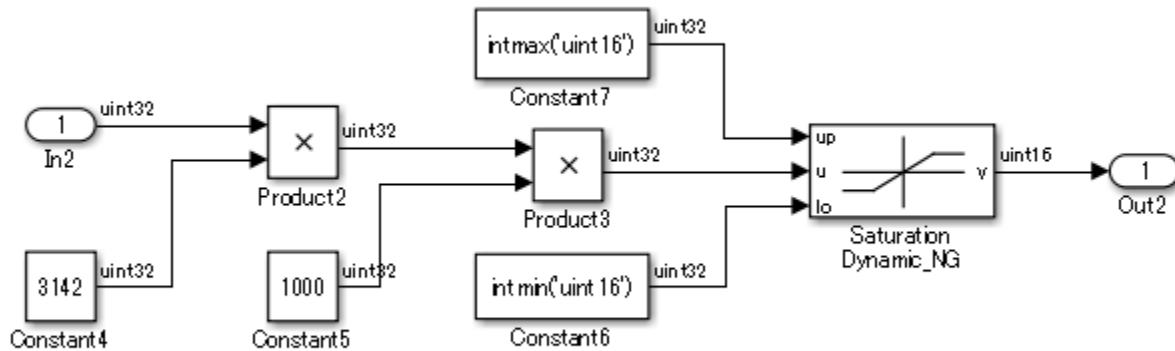
Example — Correct

The Saturation Dynamic block is limiting physical quality.



Example — Incorrect

The Saturation Dynamic block is not being used to limit physical quantity. Type conversion is being used. The upper and lower limits for the data type maximum and minimum values are set.



Rationale

Sub ID a:

- Consistent use of Saturation blocks improve maintainability of the model.

Verification

Model Advisor check: "Check usage of the Saturation blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0651: Implementing a type conversion

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

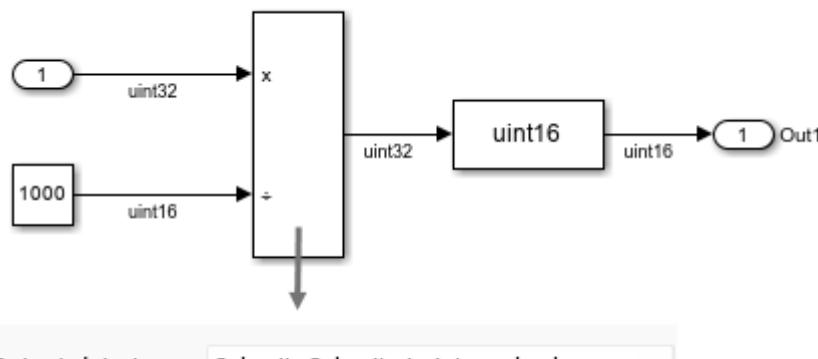
The **Output data type** parameter of the output signal of a numerical operation block shall be set to **Inherit:.. A Data Type Conversion** shall be used when changing the data type of the block output signal.

Custom Parameter

Not Applicable

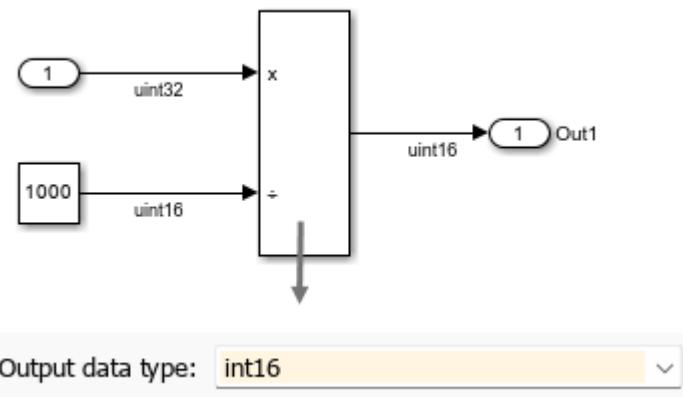
Example — Correct

The **Output data type** parameter of the Divide block is set to **Inherit: Inherit via internal rule**.



Example — Incorrect

The **Output data type** parameter of the Divide block is set to **int16**.



Rationale

Sub ID a:

- Dividing the math operations and type cast can help to clarify the order of execution and data type for each expression.

Verification

Model Advisor check: "Check Output data type of operation blocks" (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2020a

Other Blocks

db_0042: Usage of Import and Outport blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

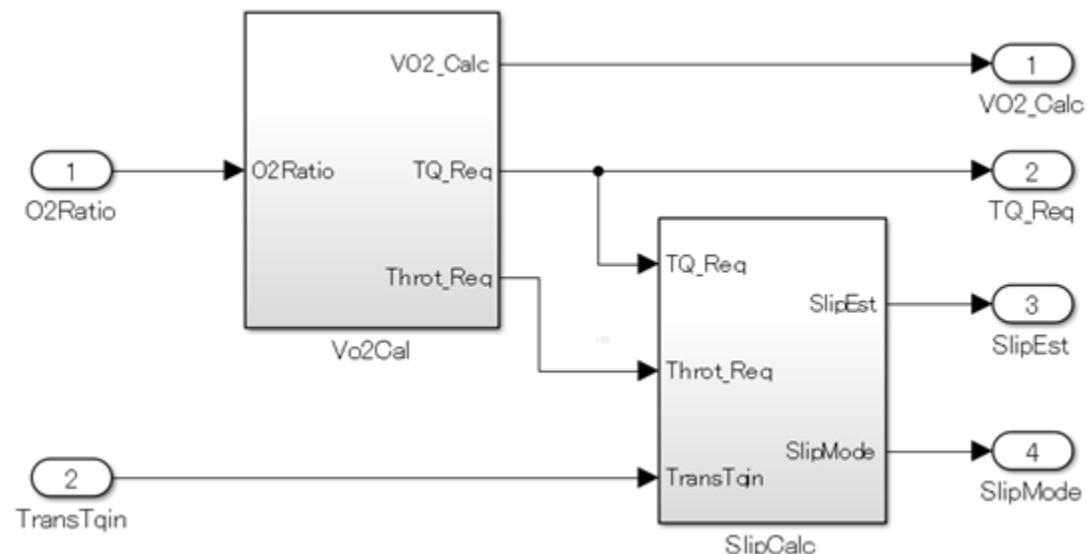
The Import block shall be positioned on the left side of the diagram, but can be moved to prevent the crossing of signals.

Custom Parameter

Not Applicable

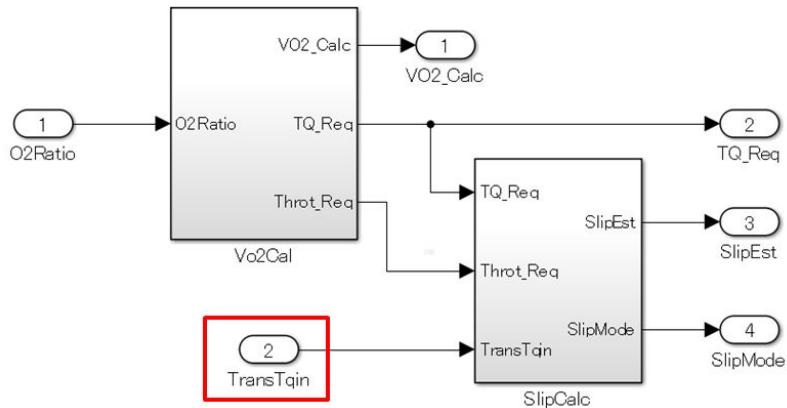
Example — Correct

Import block is positioned on the left side of the diagram.



Example — Incorrect

Import block is not positioned on the left side of the diagram.

**Sub ID b**

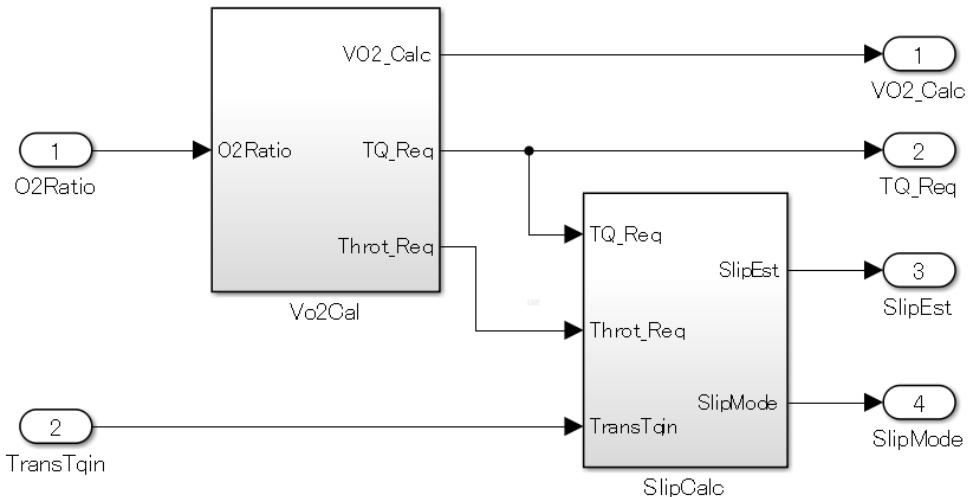
The Outport block shall be positioned on the right side of the diagram, but can be moved to prevent the crossing of signals.

Custom Parameter

Not Applicable

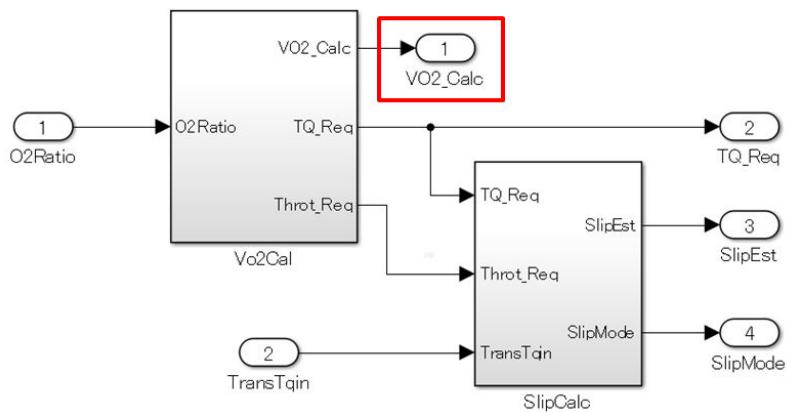
Example — Correct

Outport block is positioned on the right side of the diagram.



Example — Incorrect

Outport block is not positioned on the right side of the diagram.

**Sub ID c**

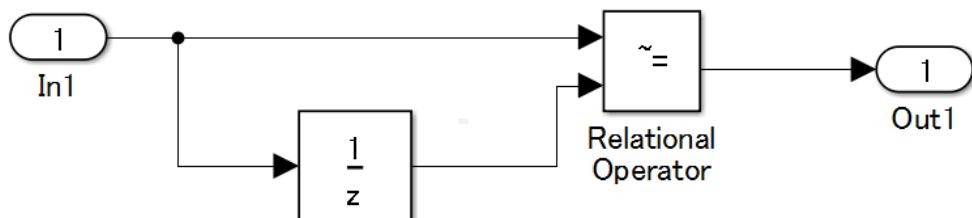
Duplicate Import blocks shall be prohibited.

Custom Parameter

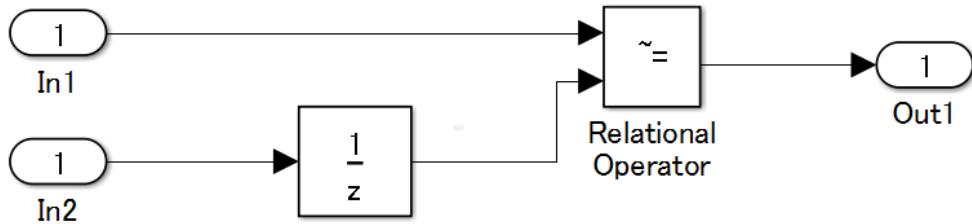
Not Applicable

Example — Correct

One Import block is used.

**Example — Incorrect**

Import blocks is duplicated.



Rationale

Sub IDs a, b, c:

- Defined operation rules improve readability.

Verification

Model Advisor check: "Check position of Import and Outport blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0081: Import and Outport block icon display

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

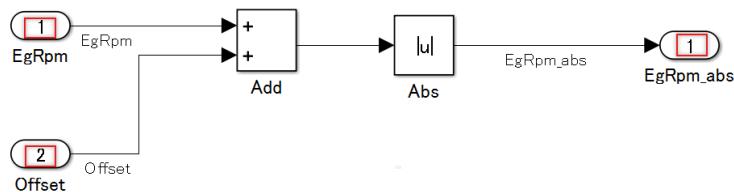
Sub ID a

For Import and Outport blocks, parameter **Icon Display** shall be set to **Port number**.

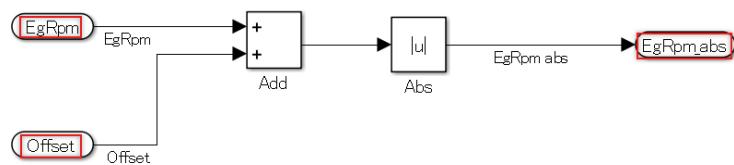
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- Improves readability by displaying the port number of Inport and Outport blocks.
- Allows for easy identification of port numbers that are within a subsystem.
- Prevents misconnections to hierarchized subsystems by displaying the block names and making the names of signal lines to the Inport and Outport blocks the same as the block names.

Verification

Model Advisor check: "Check display for port blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Set Block Parameter Values”
- “Signal Basics”
- “Simulink Bus Capabilities”

Version History

Introduced in R2020a

na_0011: Scope of Goto and From blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

Goto block parameter **tag visibility** shall be set to **Local**.

Custom Parameter

Not Applicable

Sub ID b

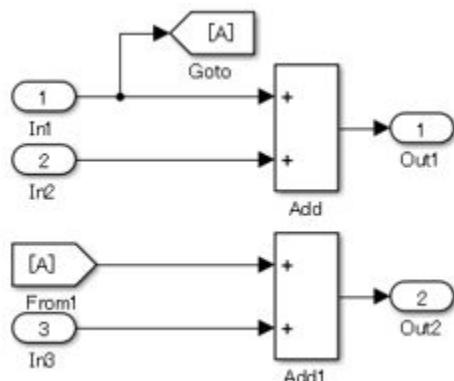
The output signal of Import block shall not exclusively connect to Goto block.

Custom Parameter

Not Applicable

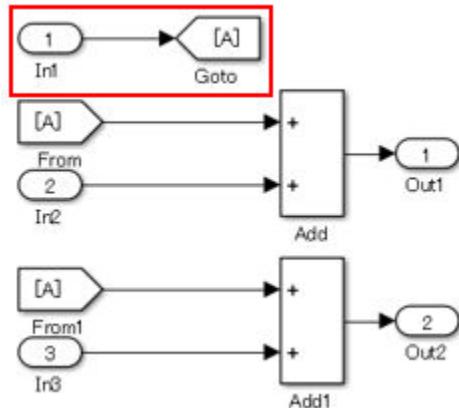
Example — Correct

The output signal of Import block connects to something other than Goto block.



Example — Incorrect

The output signal of Input block connects exclusively to Goto block.

**Sub ID c**

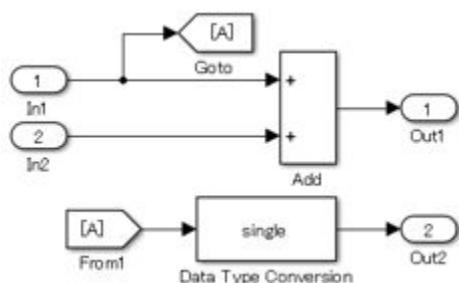
The output signal of a From block shall not connect to an Outport block.

Custom Parameter

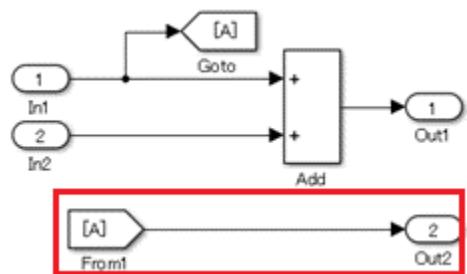
Not Applicable

Example — Correct

The output signal of a From block does not connect to an Outport block.

**Example — Incorrect**

The output signal of a From block connects to an Outport block.



Rationale

Sub ID a:

- When the hierarchies of the Goto block and the corresponding From block are different, the relationships between the blocks can be difficult to understand.
- Simulation errors can occur when hierarchies of the Goto block and the corresponding From block are different and a virtual subsystem changes to an Atomic subsystem.

Sub ID b:

- When connecting the output signal of an Inport block to a Goto block, at least one branched signal connects to blocks other than the Goto block.

Verification

Model Advisor check: "Check scope of From and Goto blocks" (Simulink Check)

Last Changed

R2024b

See Also

- "Programmatically Specify Block Parameters and Properties"
- "Explore Types of Subsystems"
- "Simulink Subsystem Semantics"

Version History

Introduced in R2020a

jc_0161: Definition of Data Store Memory blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

The smallest scope level shall be used to define the Data Store Memory block.

Custom Parameter

Not Applicable

Sub ID b

Only data required for execution and code generation shall be defined in the Data Store Memory block.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Readability improves when usage is limited.

Sub ID b:

- Unused data can affect maintenance and operability.

Verification

Model Advisor check: "Check for usage of Data Store Memory blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Data Store Basics”

Version History

Introduced in R2020a

jc_0141: Usage of the Switch blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

The second Switch block input condition shall be a logical type. Switchblock parameter **Criteria for passing first input** shall be set to $u2 \sim= 0$.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- It is easier to understand specifications when the configuration is applied by using Simulink blocks rather than by writing operation expressions in blocks.

Verification

Model Advisor check: "Check usage of Switch blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0650: Block input/output data type with switching function

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

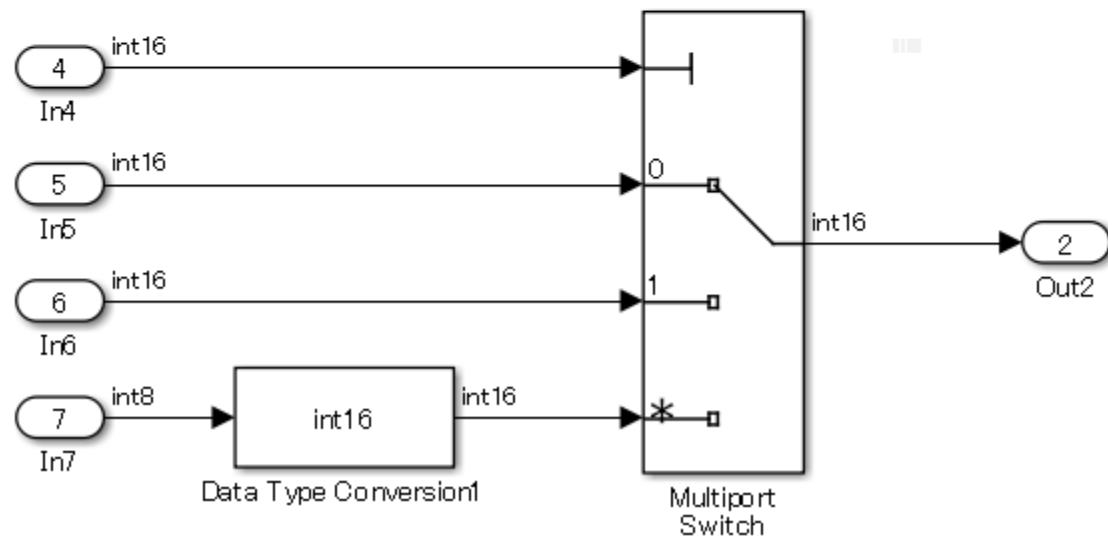
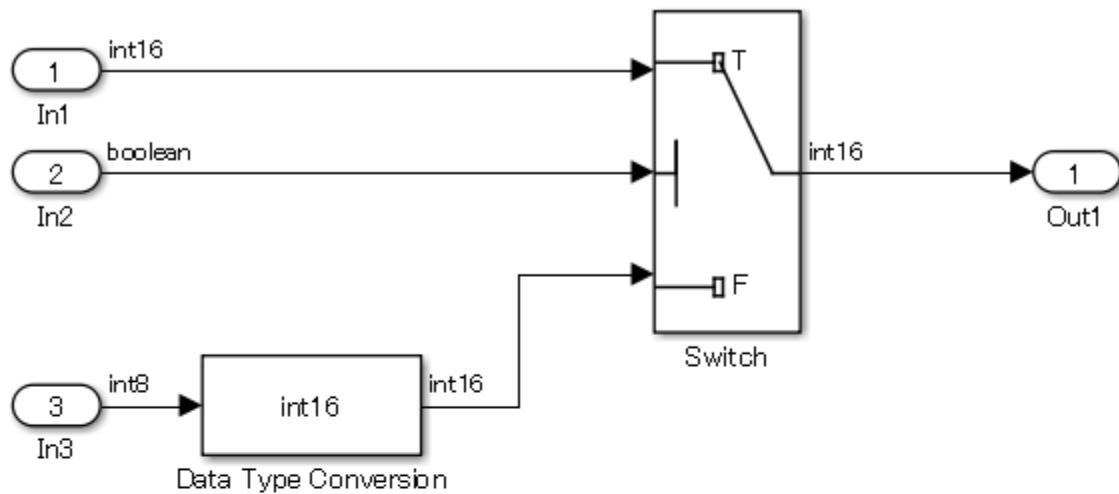
For blocks with switching functions (Switch, Multiport Switch, and Index Vector), the same data type shall be used for data ports and output ports.

Custom Parameter

Not Applicable

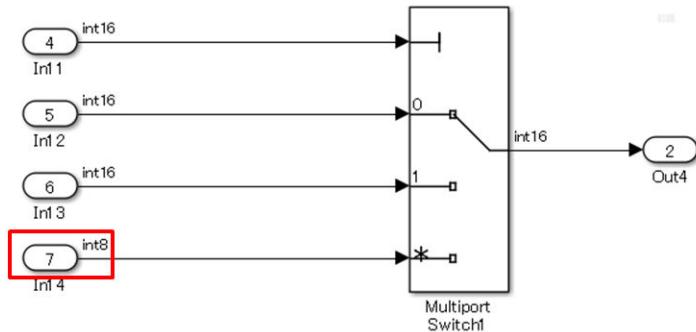
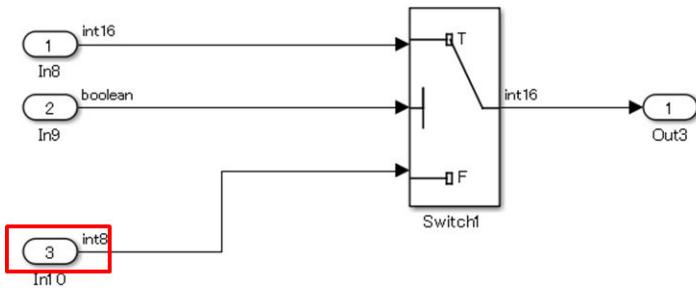
Example — Correct

The data type for the data port and output port is the same.



Example — Incorrect

The data port and output port have different data types.



Rationale

Sub ID a:

- Prevents implicit data conversion.

Verification

Model Advisor check: "Check input and output datatype for Switch blocks" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0630: Usage of Multiport Switch blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c
- JMAAB — a, b, c

MATLAB Versions

All

Rule

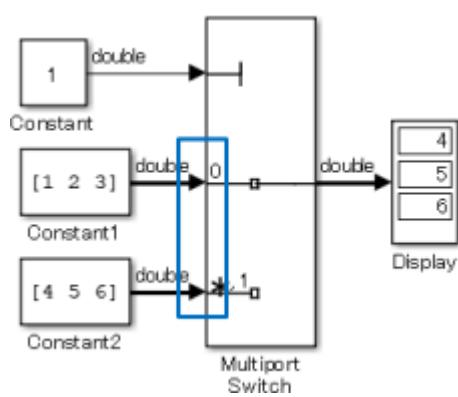
Sub ID a

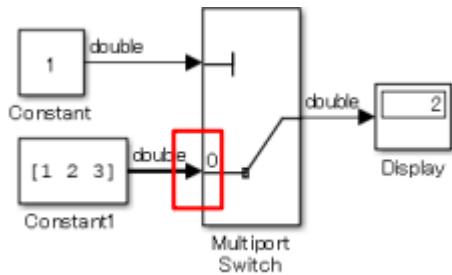
Multiport Switch block parameter **Number of data ports** shall be two or more.

Custom Parameter

Not Applicable

Example — Correct



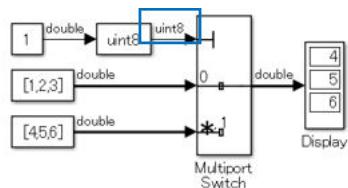
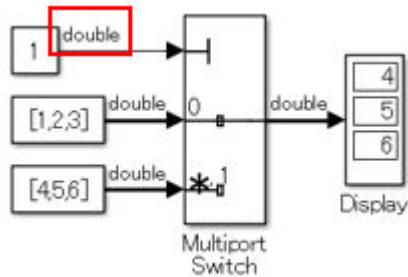
Example — Incorrect**Sub ID b**

The data type of the input signal to the control port of Multiport Switch shall be:

- uint8, uint16, or uint32
- enum (avoid using negative values for literals)

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Sub ID c**

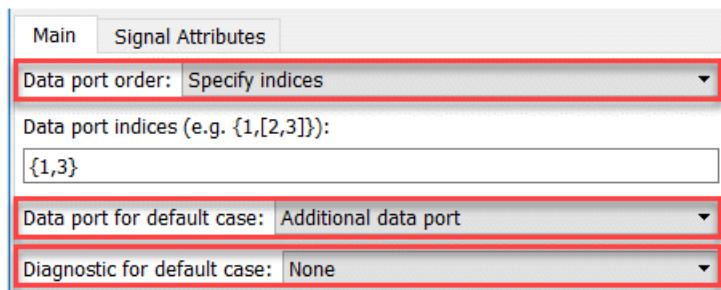
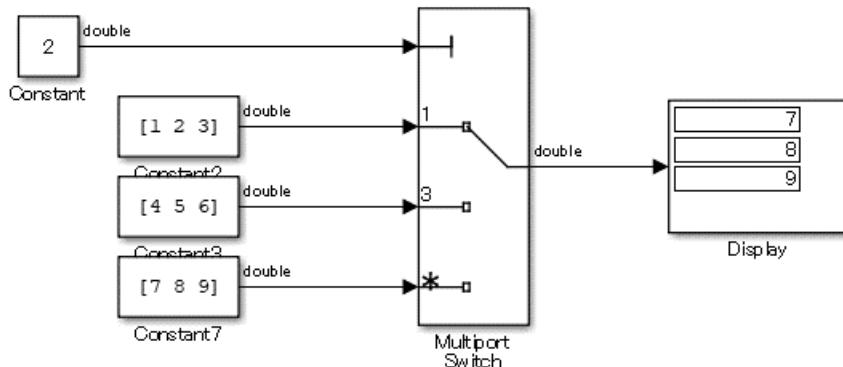
When Multiport Switch block parameter **Data port order** is set to **Specify indices**, these block parameters shall be set as follows:

- Set **Data port for default case** to **Additional data port**.
- Set **Diagnostic for default case** to **None**.

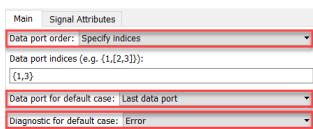
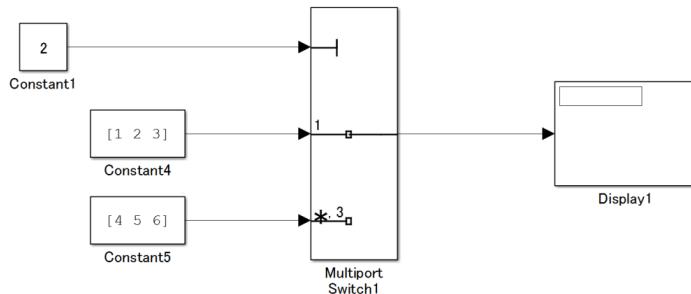
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- Unintended output can occur when there is only one data port because the block changes to extract scalars from vectors.

Sub ID b:

- The control port is an input range that expects an integer value of zero or greater. When a signed or non-integer signal is connected to the control port, it can appear as a misconnection.

- There is a possibility of data ports being unintentionally selected when negative or non-integer values are input.

Sub ID c:

- When block parameter **Data port order** is set to **Specify indices**, a value that inputs to the Multiport Switch block, other than the index specified for the control port, is treated the same as the last value of the specified index. As a result, an unintended data port can be selected.

Verification

Model Advisor check: "Check settings for data ports in Multiport Switch blocks" (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2020a

na_0020: Number of inputs to variant subsystems

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — b
- JMAAB — a, b

MATLAB Versions

All

Rule

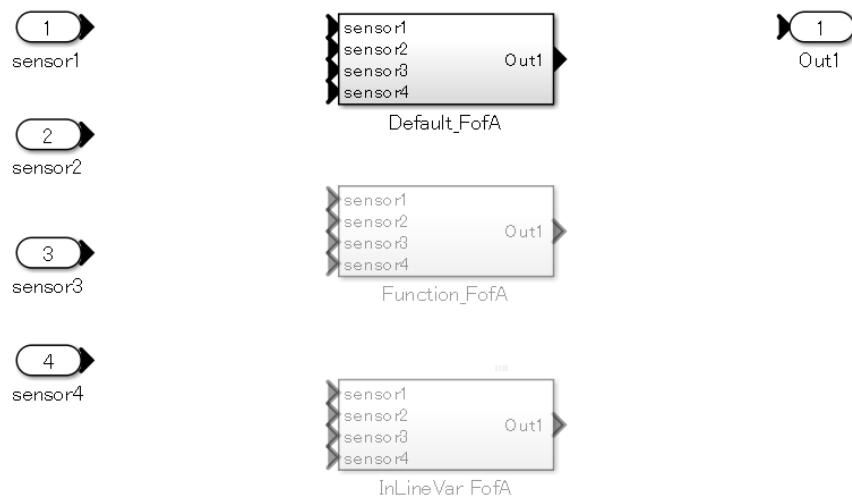
Sub ID a

The number of inputs/outputs for a Variant Subsystem block and its child structured subsystem or model reference shall be the same.

Custom Parameter

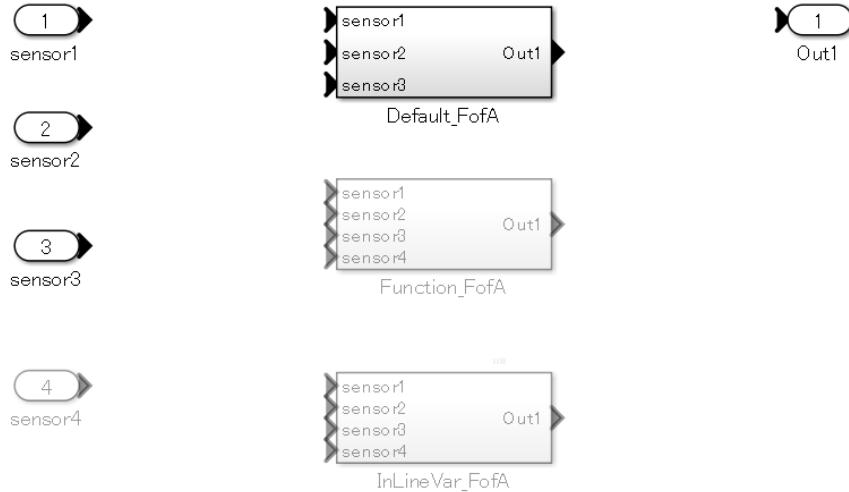
Not Applicable

Example — Correct



Example — Incorrect

The number of inputs to the child subsystem is different.



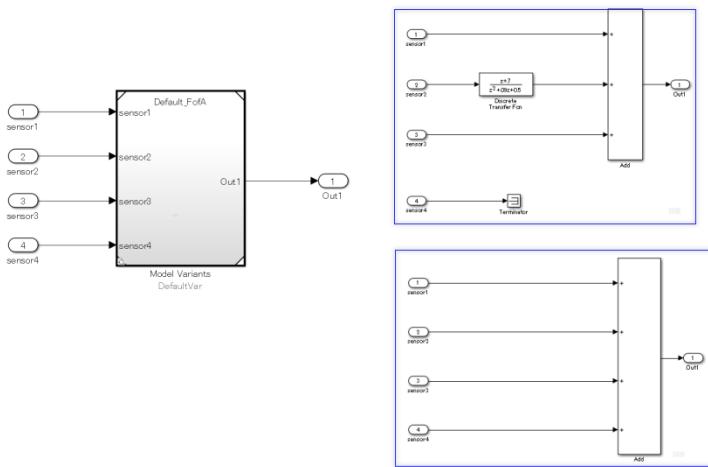
Sub ID b

The number of inputs/outputs for a Variant Model block shall be the same as its referenced model.

Custom Parameter

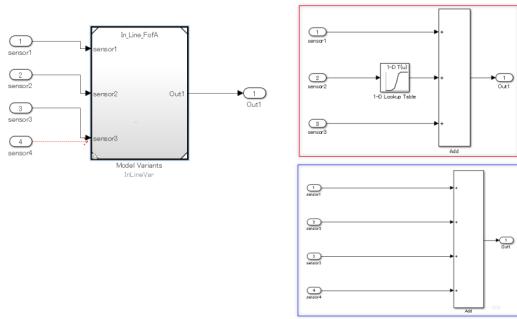
Not Applicable

Example — Correct



Example — Incorrect

The number of inputs to the referenced model is different than the inputs to the Variant Model block.



Rationale

Sub IDs a, b:

- Unconnected signals can be unintentionally overlooked when the number of inputs/outputs is different.

Verification

Model Advisor check: "Check for missing ports in Variant Subsystems" (Simulink Check)

Last Changed

R2024b

See Also

- “Explore Types of Subsystems”
- “Model Reference Basics”
- “Create a Simple Variant Model”

Version History

Introduced in R2020a

na_0036: Default variant

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

Variant subsystems shall be configured so that one subsystem is always selected. This is achieved by using one of these methods:

- Use the default variant for the variant.
- Define conditions that exhaustively cover all possible values of the conditional variables. For example, define conditions for true and false values of a Boolean.

Custom Parameter

Not Applicable

Example — Correct

A default variant is used.

Variant choices (list of child subsystems or model blocks)			
	Name (read-only)	Variant control	Condition (read-only)
		(default)	(N/A)
	Default_FofA	functionVar	FUNC==1
	In_Line_FofA	inLineVar	FUNC==2

FUNC is a logical type.

Variant choices (list of child subsystems or model blocks)			
	Name (read-only)	Variant control	Condition (read-only)
	Function_FofA	functionVar	FUNC==1
	In_Line_FofA	inLineVar	FUNC==0

Example — Incorrect

An active subsystem will not exist when FUNC is not 1 or 2.

Variant choices (list of child subsystems or model blocks)			
	Name (read-only)	Variant control	Condition (read-only)
	Function_FofA	functionVar	FUNC==1
	In_Line_FofA	inLineVar	FUNC==2

Sub ID b

Model variant conditions shall be set so all values that can be applied to conditional variable signals are configured so one subsystem is always selected. For example, a condition is prepared for the variable signal value being true, as well as false.

Custom Parameter

Not Applicable

Example — Correct

The condition is set so that all values for the conditional variable are covered.

Variant choices			
	Model name	Variant control	Condition (read-only)
	Model_SUM	defaultVar	(FUNC~=1)&&(FUNC~=2)
	Model_SUB	functionVar	FUNC==1
	Model_MUL	inLineVar	FUNC==2

Example — Incorrect

An active subsystem will not exist when FUNC is not 1 or 2.

Variant choices			
	Model name	Variant control	Condition (read-only)
	Model_SUB	functionVar	FUNC==1
	Model_MUL	inLineVar	FUNC==2

Rationale

Sub IDs a, b:

- Prevents the omission of conditions.
- There may not be an active subsystem when conditions are omitted.

Verification

Model Advisor check: "Check use of default variants" (Simulink Check)

Last Changed

R2020a

See Also

- "Implement Variations in Separate Hierarchy Using Variant Subsystems"
- "Propagate Variant Conditions to Define Variant Regions in Virtual and Nonvirtual Subsystems"

Version History

Introduced in R2020a

na_0037: Use of single variable for variant condition

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Variant conditions shall be used to prohibit compound conditions that are formed from multiple variables.

Exception

Conditional expressions that are formed from multiple variables can be used when using default variants.

Custom Parameter

Not Applicable

Example — Correct

The variant condition is set by a single condition that is formed from multiple variables.

The usage of enumerated type variables is recommended in a condition equation. This example uses numerical values to improve readability.

Name	Submodel Configuration	Variant Control	Condition
na0037a_OK			
Variant Subsystem			
Default_FofA		DefaultVar	(INLINE==0)&&(FUNC==0)
Function_FofA		FunctionVar	FUNC==1
InLineVar_FofA		InLineVar	INLINE==1

Example — Incorrect

The variant condition is set by a compound condition that is formed from multiple variables.

Name	Submodel Configuration	Variant Control	Condition
na0037a_NG			
Variant Subsystem			
AutoTrans		autoTrans	(INLINE==0)&&(transType==5)
Default_4speed		defaultTrans	((INLINE==0)&&(transType==3))==0)&&(FUNC==0)&&(transType~==2)
ManualTrans		manualTrans	(FUNC==1) ((transType==2))

Rationale

Sub ID a:

- Complicates the conditions, which makes it difficult to determine which subsystem will become active. This can result in conditions being omitted.
- When conditions are omitted, there is a risk that there may not be an active subsystem.

Verification

Model Advisor check: “Check use of single variable variant conditionals” (Simulink Check)

Last Changed

R2020a

See Also

- “What Are Variants and When to Use Them”
- “Introduction to Variant Controls”

Version History

Introduced in R2020a

jc_0900: Usable characters for data type definition

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

Underscores shall not be used at the end of bus object names, enumeration class names, or enumeration member names.

Custom Parameter

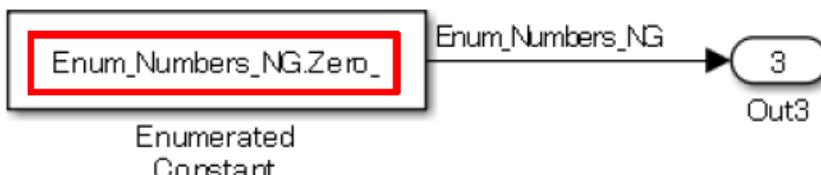
Not Applicable

Example — Incorrect

An underscore is used at the end of the bus object name.



An underscore is used at the end of the enumeration member name.



Sub ID b

Consecutive underscores shall not be used in bus object names, enumeration class names, or enumeration member names.

Custom Parameter

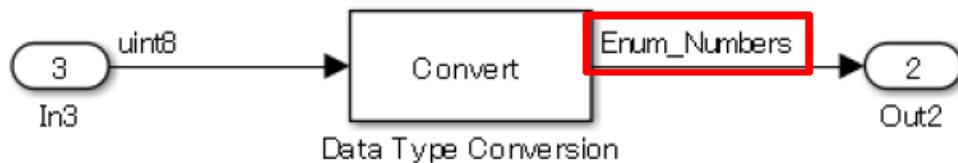
Not Applicable

Example — Incorrect

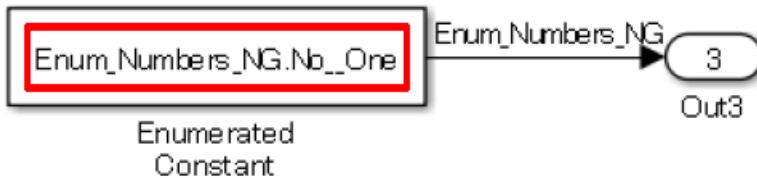
Consecutive underscores are used in the bus object name.



Consecutive underscores are used in the enumeration class name.



Consecutive underscores are used in the enumeration member name.



Sub ID c

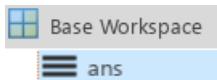
Bus object names, enumeration class names and enumeration member names shall not be the same as reserved MATLAB words.

Custom Parameter

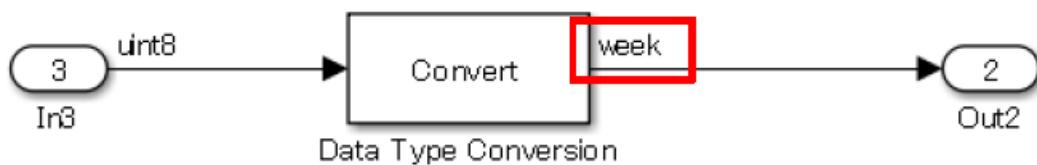
Not Applicable

Example — Incorrect

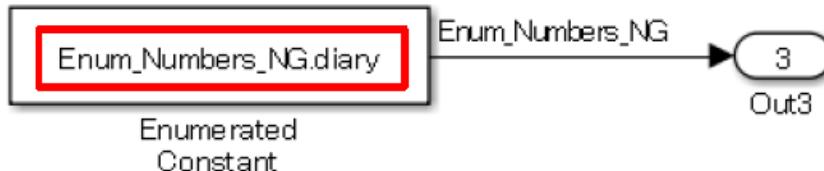
A bus object has the same name as a reserved MATLAB word.



A reserved MATLAB word that exactly matches the enumeration class name is used.



A reserved MATLAB word that exactly matches the enumeration member name is used.



Rationale

Sub ID a:

- Deviation reduces readability. Since underscores are commonly used as word separators, it may look like a typographical error.

Sub ID b:

- Deviation reduces readability.

Sub ID c:

- Deviation reduces readability. Deviating from common practices may lead to unexpected problems.

Verification

Model Advisor check: “Check bus and enumeration data type names” (Simulink Check)

Last Changed

R2024b

See Also

“Specify Bus Properties with Bus Objects”

Version History

Introduced in R2024b

jc_0901: Length restriction for Bus and Enumeration data type names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

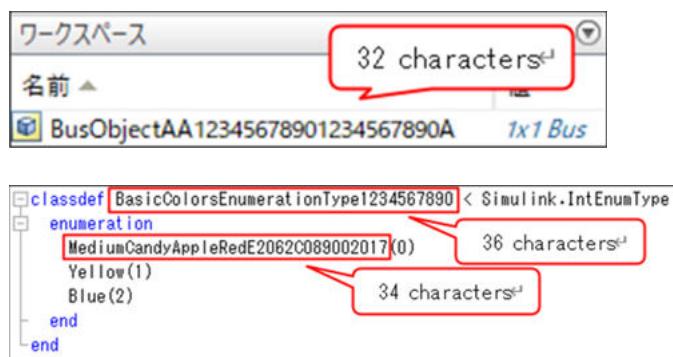
The number of characters in a bus object name, enumeration class name, or enumeration member name shall be less than or equal to the configured limit. The default is 63.

Custom Parameter

- Maximum number of characters for bus object names
- Maximum number of characters for enumeration class names
- Maximum number of characters for enumeration member names

Example — Incorrect

When the maximum number of characters is set to 31. The bus object, enumeration class, and enumeration member names are longer than 31 characters.



Rationale

Sub ID a:

- Deviation reduces readability.
- Deviation makes it difficult to maintain the integrity of the model and code.

Verification

Model Advisor check: "Check length of bus and enumeration data type names" (Simulink Check)

Last Changed

R2024b

See Also

"Specify Bus Properties with Bus Objects"

Version History

Introduced in R2024b

jc_0902: Arrowhead size of transition lines

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

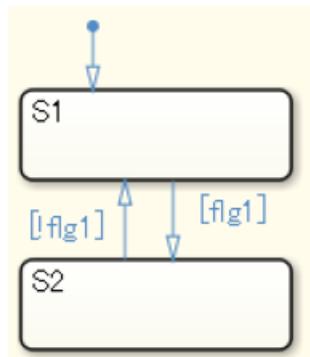
Sub ID a

The arrowhead {size} of transition lines shall be set to the value defined in the project.

Custom Parameter

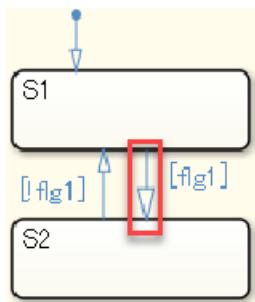
Arrowhead {size} of transition lines.

Example — Correct



Example — Incorrect

The arrowhead {size} of transition lines are inconsistent.



Rationale

Sub ID a:

- Maintaining a consistent arrowhead `{size}` of transition lines improves readability.

Verification

Model Advisor check: "Check arrowhead size of transition lines" (Simulink Check)

Last Changed

R2024b

See Also

"Transition Between Operating Modes" (Stateflow)

Version History

Introduced in R2024b

jc_0903: Prohibition of overlapping or crossing of blocks and signal lines

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b1/b2
- JMAAB — a, b1/b2

MATLAB Versions

All

Rule

Sub ID a

Blocks, block names, block annotations, annotations, and signal labels shall not overlap.

Exception

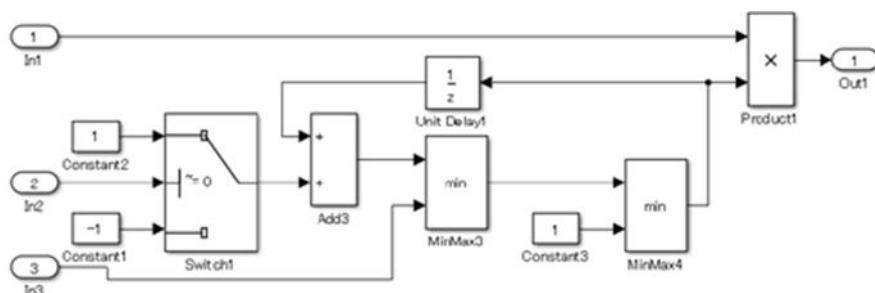
When modeling within an annotation frame by treating the frame as a subsystem boundary, only the lines of the annotation frame, not the entire frame, are subject to this rule.

Custom Parameter

Not Applicable

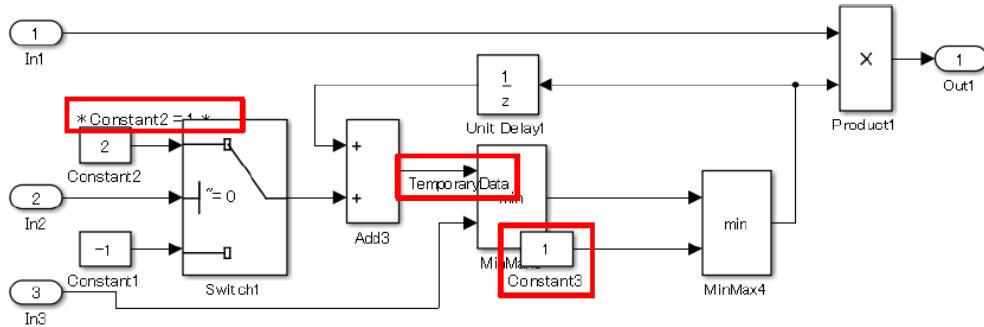
Example — Correct

Blocks, block names, and signal lines do not overlap.



Example — Incorrect

Blocks, block names, and signal labels overlap.

**Sub ID b1**

- Signal lines shall not overlap with blocks, block names, block annotations, annotations, signal lines, or signal labels.
- Signal lines shall not intersect with other signal lines.

Custom Parameter

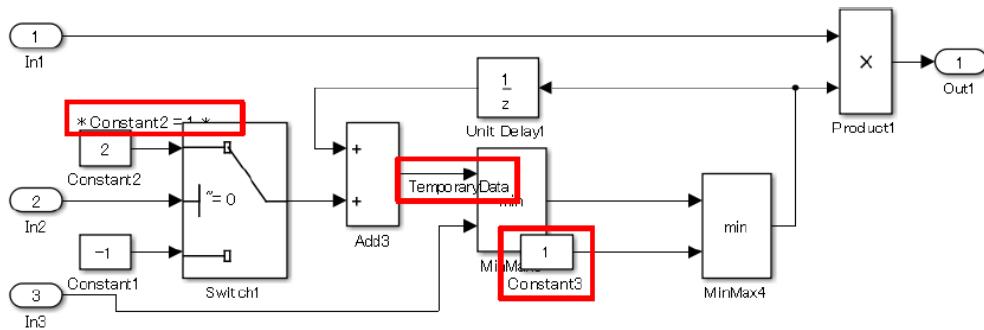
Not Applicable

Exception

Intersection of annotation frames and signal lines when modeling within an annotation frame by considering the frame as a subsystem boundary.

Example — Incorrect

Signal lines overlap with blocks and block names, and also and intersect with other signal lines.

**Sub ID b2**

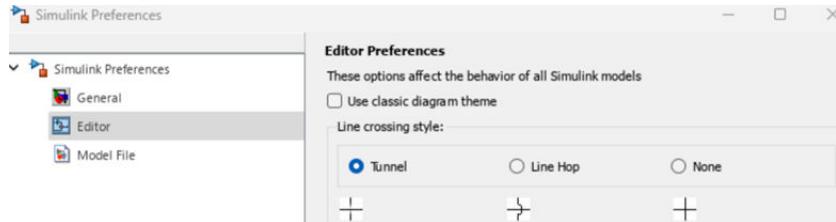
- The Line crossing style shall be set to Line Hop option.
- Signal lines shall not overlap with blocks, block names, block annotations, annotations, signal lines, or signal labels.

Custom Parameter

Not Applicable

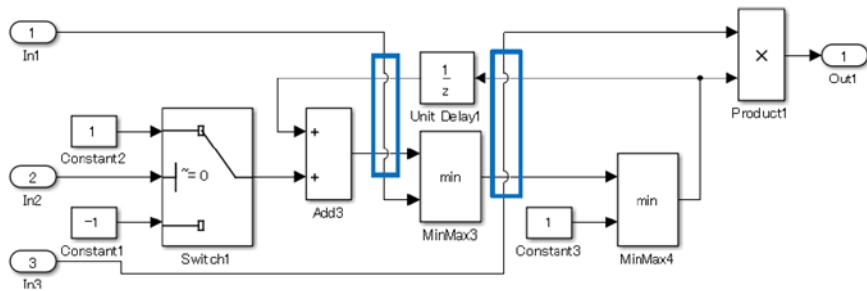
Exception

Intersection of annotation frames and signal lines when modeling within an annotation frame by considering the frame as a subsystem boundary.



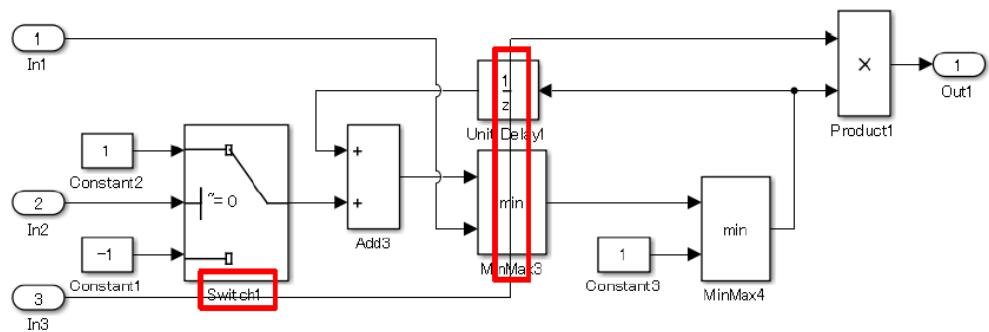
Example — Correct

Signal lines are not overlapped with blocks or block names.



Example — Incorrect

Signal lines are overlapped with blocks and block names.



Rationale

Sub IDs a, b1, b2:

- Overlapping elements in a block diagram can reduce readability.

Verification

Model Advisor check: "Check for prohibited overlapping or intersecting blocks and signal lines" (Simulink Check)

Last Changed

R2024b

See Also

“Signal Basics”

Version History

Introduced in R2024b

jc_0904: Prohibition of overlap/intersection of states and transition lines

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

States, graphical functions, Simulink functions, MATLAB functions, truth tables, boxes, junctions, annotations, or text labels shall not overlap.

Custom Parameter

Not Applicable

Exceptions

Exceptions apply in any of the following cases:

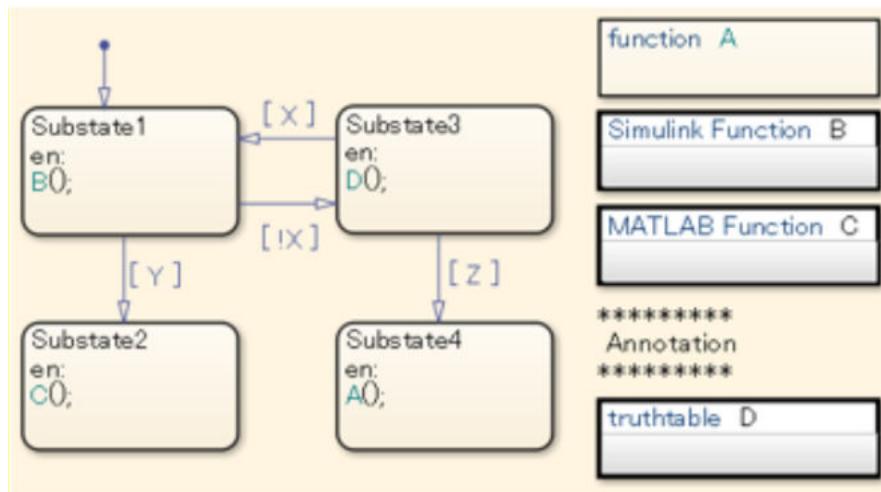
- Overlapping of parent and child in hierarchical modeling, provided that the child is contained within the parent's frame.
 - Modeling with hierarchical states (overlap with superstates)
 - Modeling with graphical functions overlap with graphical function frames
- States and their labels must fit within the state frame. Likewise for graphical functions, Simulink functions, MATLAB functions, truth tables, and boxes.

Supplement

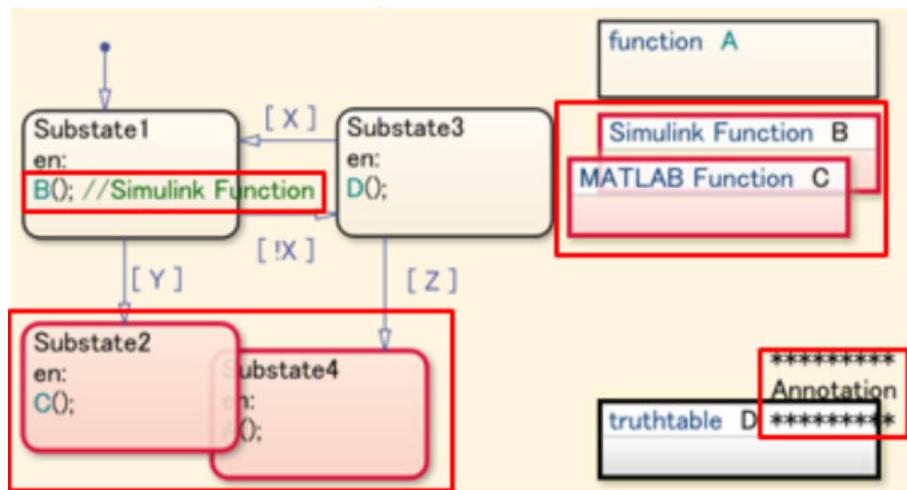
Some combinations, such as the overlap of states, may result in errors.

Example — Correct

State labels do not overlap.

**Example — Incorrect**

State labels, etc. are overlapping.

**Sub ID b**

Transition lines shall not overlap or intersect with states, graphical functions, Simulink functions, MATLAB functions, truth tables, boxes, junctions, annotations, text labels, or other transition lines.

Custom Parameter

Not Applicable

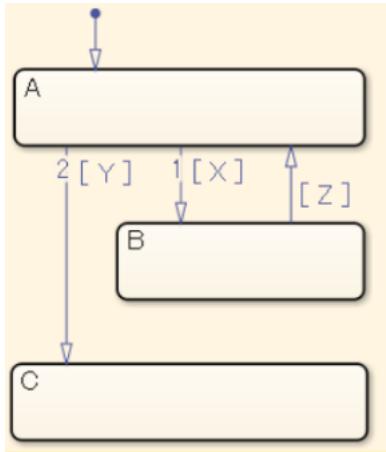
Exceptions

Exceptions apply in any of the following cases:

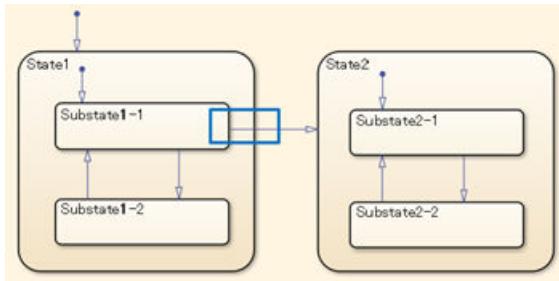
- Overlapping of parent and child in hierarchical modeling, provided that the child is contained within the parent's frame.
 - Modeling with hierarchical states (overlap with superstates)

- Modeling with graphical functions overlap with graphical function frames

Example — Correct

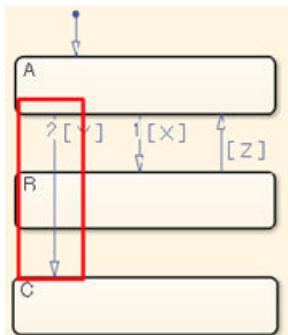


Since this is an intersection for crossing state hierarchies, it does not violate the rules.



Example — Incorrect

This transition line intersects with the state.



Rationale

Sub IDs a, b:

- Deviation reduces readability.

Verification

Model Advisor check: “Check for prohibited overlapping of states and transition lines in Stateflow charts” (Simulink Check)

Last Changed

R2024b

See Also

- “Signal Basics”
- “Transition Between Operating Modes” (Stateflow)

Version History

Introduced in R2024b

jc_0905: Usable characters for data names in Functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Data names defined within MATLAB functions shall not use underscores at the end.

Custom Parameter

Not Applicable

Sub ID b

Consecutive underscores shall not be used in data names defined within MATLAB functions.

Custom Parameter

Not Applicable

Sub ID c

Data names defined within a MATLAB functions shall not be the same as reserved MATLABwords.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Deviation reduces readability. Since underscores are commonly used as word separators, it may look like a typographical error.

Sub ID b:

- Deviation reduces readability.

Sub ID c:

- Deviation reduces readability.
- Deviating from common practices may lead to unexpected problems.

Verification

Model Advisor check: “Check data names in MATLAB functions” (Simulink Check)

Last Changed

R2024b

See Also

“Model Reference Data Storage”

Version History

Introduced in R2024b

jc_0906: Length restriction of data names in MATLAB Functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

The number of characters for data names defined within a MATLAB function shall be less than or equal to configured limit. The default is 63.

Custom Parameter

The maximum number of characters for data names in MATLAB function is 63.

Rationale

Sub ID a:

- Deviation reduces readability.
- Deviation makes it difficult to maintain the integrity of the model and code.

Verification

Model Advisor check: "Check the length of data names in MATLAB functions" (Simulink Check)

Last Changed

R2024b

See Also

[“Reserved Keywords” \(Embedded Coder\)](#)

Version History

Introduced in R2024b

jc_0907: Size of a Junction

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

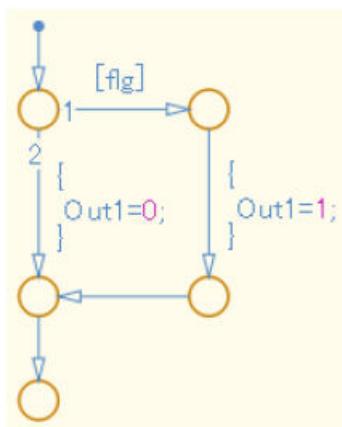
The size of junctions shall be set to the value defined in the project.

Custom Parameter

Size of junctions

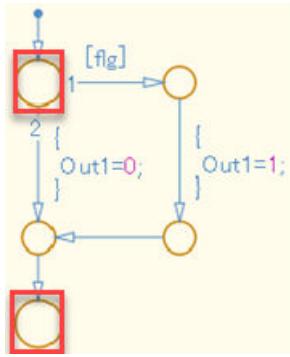
Example — Correct

The sizes of the junctions are the same.



Example — Incorrect

The sizes of the junctions are not same.



Rationale

Sub ID a:

- Using junctions of the same size improves readability.

Verification

Model Advisor check: "Check size of junctions" (Simulink Check)

Last Changed

R2024b

See Also

"Represent Multiple Paths by Using Connective Junctions" (Stateflow)

Version History

Introduced in R2024b

mp_0007: How to describe executable statements

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

Multiple execution statements shall not be written in a single line.

Example — Correct

```
if foo > 0  
    bar = 10;  
end
```

Example — Incorrect

```
if foo > 0;bar = 10;  
end
```

Sub ID b

Executable statements shall end with a semicolon.

Example — Correct

```
foo = 1 + 2;
```

Example — Incorrect

```
foo = 1 + 2
```

Rationale

Sub IDs a:

- Readability is improved by standardizing the way to write statements according to common practices.

Sub ID b:

- Readability is improved by standardizing the way to write statements according to common practices.
- If you execute a statement without a semicolon, the results are displayed in the **Command Window**, and you might miss important warnings.

Verification

Model Advisor check: "Check description of execution statements" (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2024b

mp_0008: Format of parentheses

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Single-byte spaces shall not be inserted between function/variable names and left parentheses.

Example — Correct

```
fcnc(arg)  
foo(bar)
```

Example — Incorrect

```
fcnc (arg)  
foo (bar)
```

Rationale

Sub ID a:

- Readability is improved by standardizing the way to write statements according to common practices.

Verification

Model Advisor check: "Check for spaces between function or variable names and left parenthesis symbol" (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2024b

mp_0010: Precedence of Operators in Arithmetic Expressions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

When an expression contains operators with different precedence levels, parentheses shall be used to clarify the order of operations.

Example — Correct

```
foo = (bar * baz) + qux;  
foo = bar + (baz * qux);  
(bar && baz) || (qux && quux);
```

When the operators have the same precedence, parentheses are unnecessary.

```
foo = bar + baz - qux;
```

Example — Incorrect

```
foo = bar * baz + qux;  
foo = bar + baz * qux;  
bar && baz || qux && quux;
```

Rationale

Sub ID a:

- Readability could lead to unintended functionality.

Verification

Model Advisor check: “Check for operator precedence” (Simulink Check)

Last Changed

R2024b

See Also

“Operator Precedence”

Version History

Introduced in R2024b

mp_0011: Method of inserting a single-width space

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

At least one space shall be inserted before and after operators and after commas.

Single-byte spaces shall not be inserted between unary operators and operands.

Example — Correct

```
foo = bar + baz;  
strcmp(qux, 'AEIOU')  
  
x = -baz;
```

Example — Incorrect

```
foo=bar+baz;  
strcmp(qux, 'AEIOU')  
  
x = - baz;
```

Rationale

Sub ID a:

- The proper use of spaces around operators and commas can improve readability.

Verification

Model Advisor check: "Check spaces in expressions" (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2024b

mp_0022: Format of Conditional Expressions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c1/c2
- JMAAB — a, b, c1/c2

MATLAB Versions

All

Rule

Sub ID a

Conditional expressions shall be written as expressions that evaluate to a single logical value (true or false).

Example — Correct

```
% foo  
    . . . a non-logical variable  
if (foo ~= 0)
```

Example — Incorrect

```
% foo  
    . . . a non-logical variable  
if foo
```

Sub ID b

The number of binary operators per line in a conditional expression shall be defined.

Example — Correct

```
% If up to 3 is allowed  
if (foo1 && foo2) ...  
    || (foo3 && foo4 && foo5)
```

Example — Incorrect

```
% If up to 3 is allowed  
if (foo1 && foo2) || (foo3 && foo4 && foo5)
```

Custom Parameter

The number of binary operators per line in a conditional expression.

Sub ID c1

When dividing a conditional expression into multiple lines, line breaks shall be inserted immediately after a logical operator, and the subsequent lines shall be indented.

Example — Correct

```
if fool || ...
    foo2 || ...
    foo3
end
```

Example — Incorrect

```
if fool ...
    || foo2 ...
    || foo3
end
```

Sub ID c2

When dividing a conditional expression into multiple lines, line breaks shall be inserted immediately before a logical operator, and the subsequent lines shall have aligned indentation.

Example — Correct

```
if fool ...
    || foo2 ...
    || foo3
end
```

Example — Incorrect

```
if fool || ...
    foo2 || ...
    foo3
end
```

Rationale

Sub ID a:

- The conditions for true and false are clarified.

Sub ID b:

- Limiting the number of operators makes it easy to understand the relationships between each condition.

Sub ID b:

- Limiting the number of operators makes it easy to understand the relationships between each condition.

Sub IDs c1, c2:

- The relationships between each condition become easier to understand.

Verification

Model Advisor check: "Check description of conditional expressions" (Simulink Check)

Last Changed

R2024b

See Also

"Operator Precedence"

Version History

Introduced in R2024b

mp_0023: Format of relational operations using constants

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1, a2
- JMAAB — a1, a2

MATLAB Versions

All

Rule

Sub ID a1

When you are using relational operators with variables and constants, the constants shall be placed on the right-hand side.

Example — Correct

```
foo = 10;
if bar >= foo
    ret1 = 1;
end
```

Example — Incorrect

```
foo = 10;
if foo <= bar
    ret1 = 1;
end
```

Sub ID a2

When you are using relational operators with variables and constants, the constants shall be placed on the left-hand side.

Example — Correct

```
foo = 10;
if foo <= bar
    ret1 = 1;
end
```

Example — Incorrect

```
foo = 10;  
if bar >= foo  
    ret1 = 1;  
end
```

Rationale

Sub IDs a1, a2:

- Readability is improved by standardizing the way to write statements.

Verification

Model Advisor check: “Check relational operators usage” (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2024b

mp_0032: Function header information

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Necessary information shall be described in the function header.

Supplementary Information

The descriptive items can include an overview, processing details, input and output arguments, management information and other relevant information. The combination of descriptive items shall be adjusted for main functions and sub-functions as needed.

Example — Correct

```
function ret = sample(arg1)
% Overview
% sample program
% Processing details
% Adds arg1 and foo and returns the value
% Arguments
% arg1 : First term
% Return Value
% ret : Result of addition
% Change History
% -----
% 2018/3/14 Created
% -----
% 2018/3/30 Added comments

foo = 3;
ret = arg1 + foo;
end
```

Custom Parameter

Descriptive items for the function header.

Rationale

Sub ID a:

- Readability and maintainability is improved by clearly specifying the information to be shared.

Verification

Model Advisor check: "Check function headers" (Simulink Check)

Last Changed

R2024b

See Also

- na_0025: MATLAB Function header

Version History

Introduced in R2024b

mp_0034: Number of lines in a function

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

The number of lines in a function shall be kept below the limit.

Example — Correct

- Lines of the range that can be collapsed at the `function` line in the editor are counted.
- The help text just below the `function` line should be collapsed and counted.
- The number of lines in the `libinfo` function is 88 in the following example.

```

行数 libinfo.m + 
1 1 %function LibData=libinfo(Sys,varargin)
2 2 %LIBINFO Library Information
3 19
4 20 % Loren Dean
5 21 % Copyright 1990-2012 MathWorks, Inc.
6 22
7 23
8 24 % First find update all links and find all libraries
9 25 - CellBlocks = find_system(Sys , ...
:
85 101 'Library' ,Library , ...
86 102 'ReferenceBlock' ,ReferenceBlock , ...
87 103 'LinkStatus' ,LinkStatus ...
88 104 );
105 |
```

libinfo 行 105 列 1

Custom Parameter

- Number of lines

Rationale

Sub ID a:

- When a function has too many lines, it can be difficult to understand and maintain the function. Separating it by each feature deepens understanding of the functions and can limit the range affected by changes.

Verification

Model Advisor check: "Check number of lines of functions" (Simulink Check)

Last Changed

R2024b

See Also

- "Integrate MATLAB Functions in a Stateflow Charts" (Stateflow)
- "Implement MATLAB Functions in Simulink with MATLAB Function Blocks"

Version History

Introduced in R2024b

mp_0040: Using the return value of a function

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

The return values of functions shall be assigned to variables before use.

Example — Correct

```
ret1 = func1(arg1);
ret2 = func2(arg2);
ret = ret1 + ret2;
```

Example — Incorrect

```
ret = func1(arg1) + func2(arg2);
```

Rationale

Sub ID a:

- Although it is not syntactically incorrect to use the return values of functions directly, the results may vary depending on the order of the function evaluation.

Readability improves where the return values of functions are assigned to variables before use, rather than where the return values are used directly in operations.

Verification

Model Advisor check: "Check for utilization of the return value of functions" (Simulink Check)

Last Changed

R2024b

See Also

- jc_0511: Return values from a graphical function

Version History

Introduced in R2024b

mp_0046: Usage of expressions in array indices

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Array indices shall not be used to calculate array numbers. However, arithmetic operations with end and the colon operator are exceptions.

Example — Correct

```
foo = rand(1, 10);
k = 1;
m = 2 * k - 1;
foo(m) = 1;
foo(end - 1) = 2;
```

Example — Incorrect

```
foo = rand(1, 10);
k = 1;
foo(2 * k - 1) = 1;
```

Rationale

Sub ID a:

- Calculating values before using them as indices to access array elements reduces readability.

Verification

Model Advisor check: "Check array indices" (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2024b

mp_0047: Conditions that a nonempty statement must satisfy

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

A non-empty statement shall meet one of the following criteria:

- Produces at least one side effect, regardless of how it is executed.
- Has the capability to alter the control flow.

Example — Incorrect

```
foo;  
bar == 1;
```

Rationale

Sub ID a:

- Non-empty statements should be written to change data values or alter control flow, impacting the execution results in some way. Statements that do not meet these criteria are unnecessary and may not reflect the designer's intentions.

Verification

Model Advisor check: "Check for usage of nonempty statements" (Simulink Check)

Last Changed

R2024b

Version History

Introduced in R2024b

Stateflow

- “Stateflow Blocks / Data / Events” on page 4-2
- “Stateflow Diagram” on page 4-29
- “Conditional Transition / Action” on page 4-82
- “Label Description” on page 4-149
- “Miscellaneous” on page 4-176

Stateflow Blocks / Data / Events

db_0123: Stateflow port names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

The name of a Stateflow input/output shall be the same as the corresponding signal.

Exception

Reusable Stateflow blocks can have different port names.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Improves readability.
- Code generation may not be possible.

Verification

Model Advisor check: "Check for names of Stateflow ports and associated signals" (Simulink Check)

Last Changed

R2020a

See Also

- “Set Data Properties” (Stateflow)
- “Reusable Components in Charts” (Stateflow)

Version History

Introduced in R2020a

db_0125: Stateflow local data

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d
- JMAAB — a, b, c, d

MATLAB Versions

All

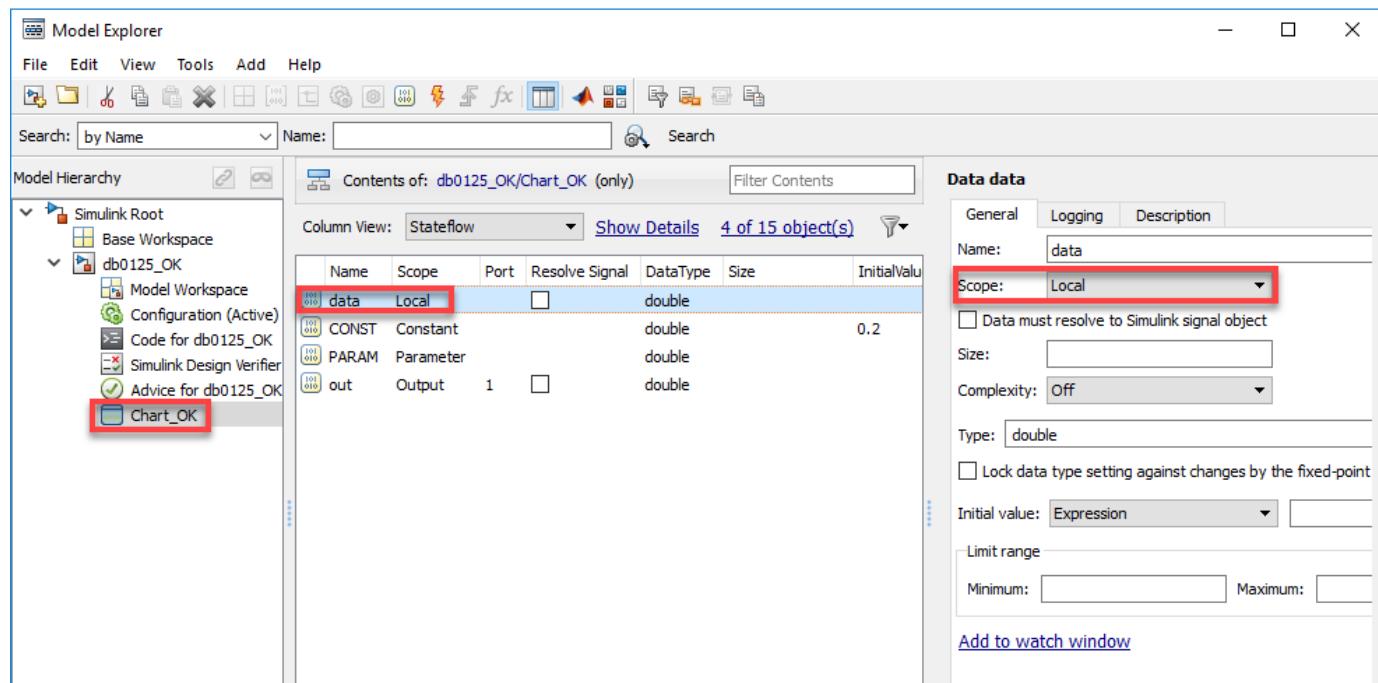
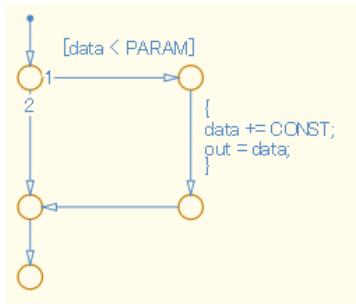
Rule

Sub ID a

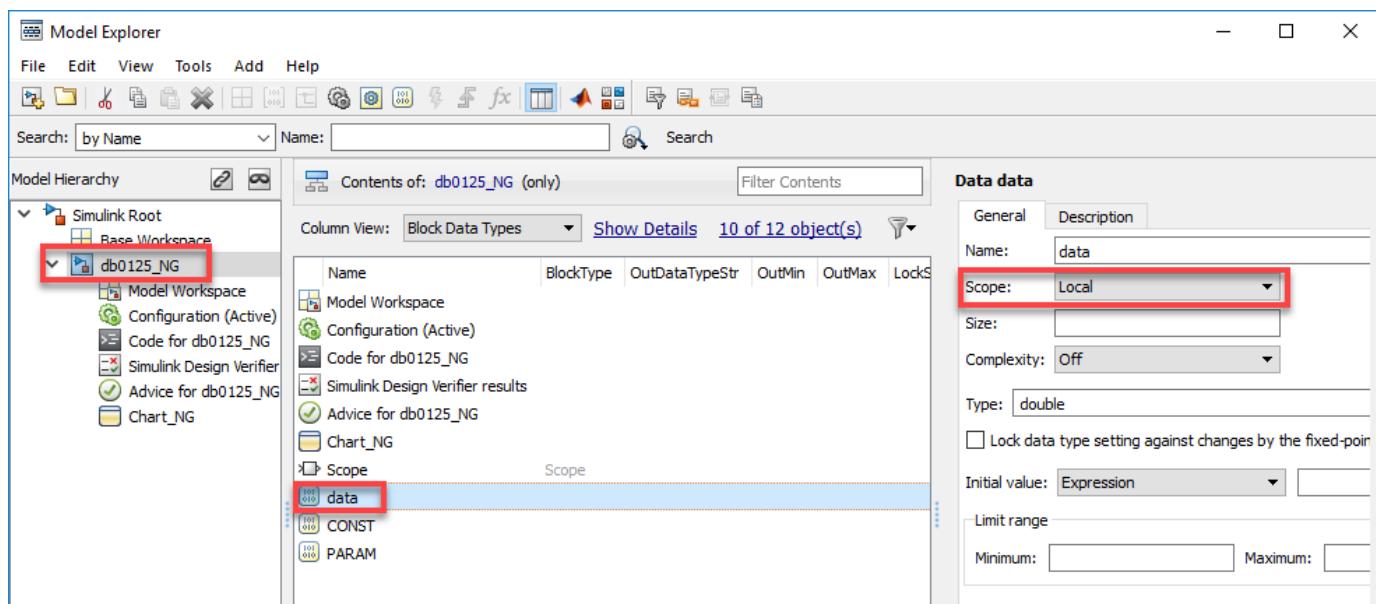
Data objects shall not be defined with Scope set to Local at the machine level.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Scope has set Local local data at the machine level.

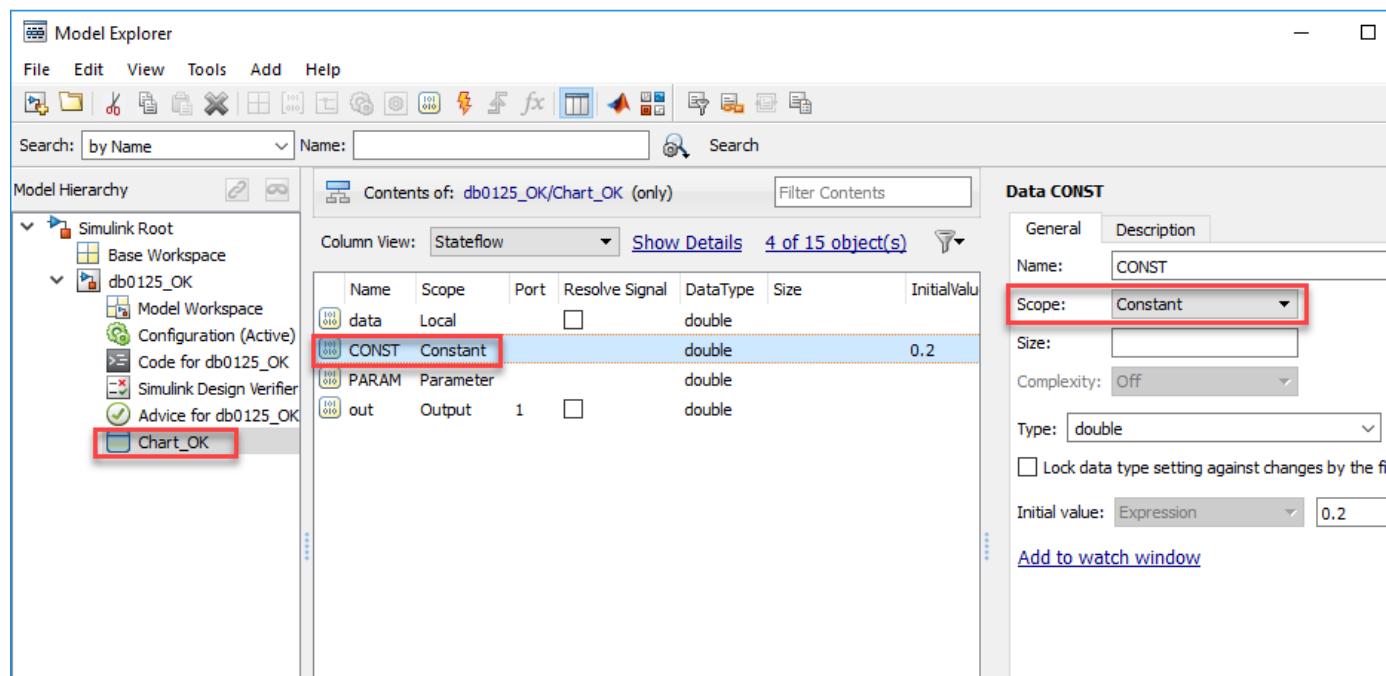
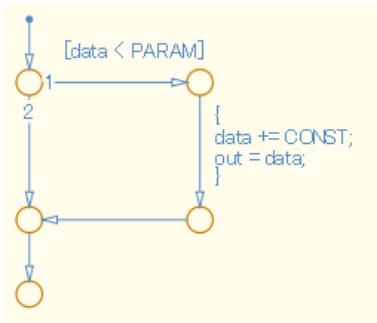


Sub ID b

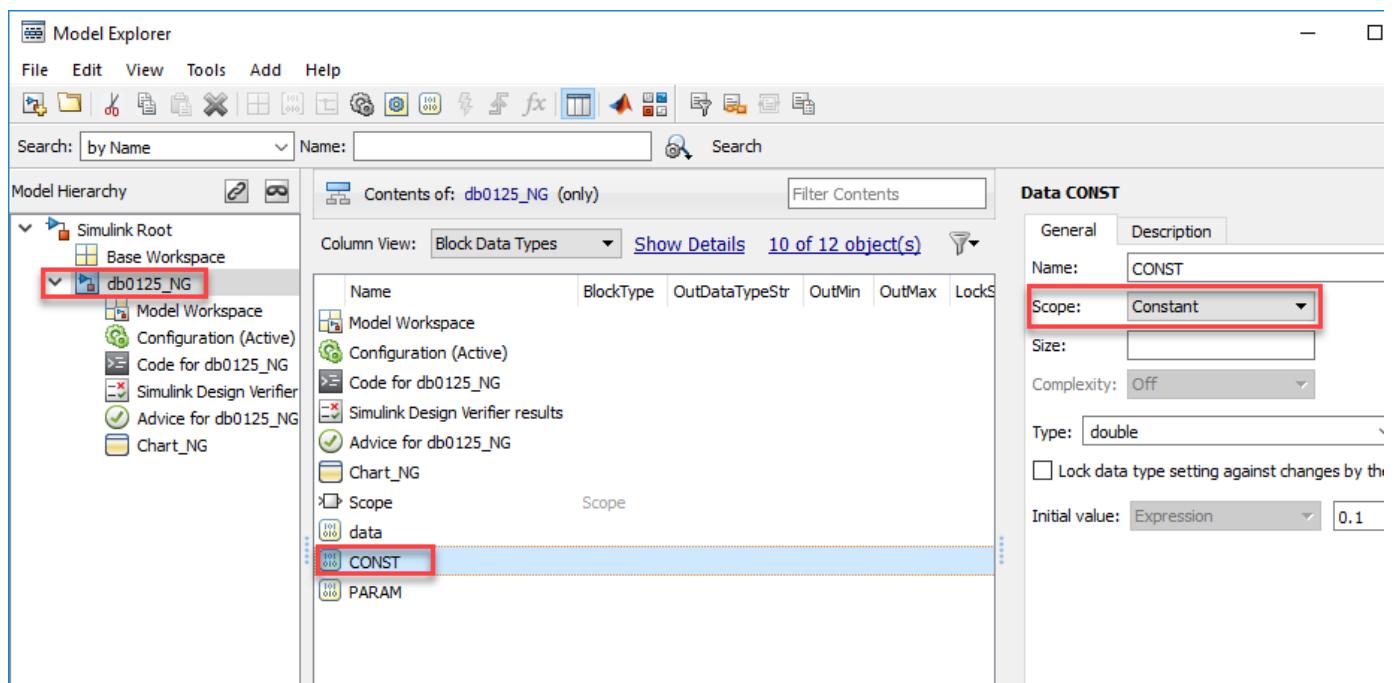
Data objects shall not be defined with Scope set to Constant at the machine level.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Scope has set Constant local data at the machine level.

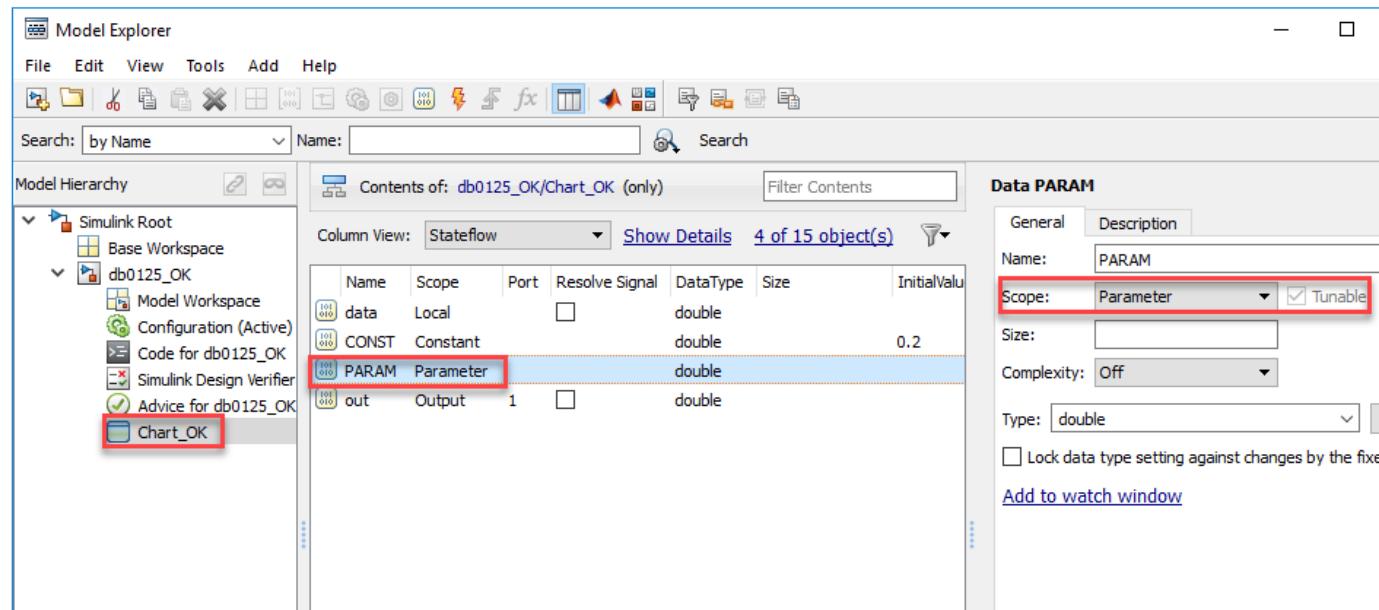
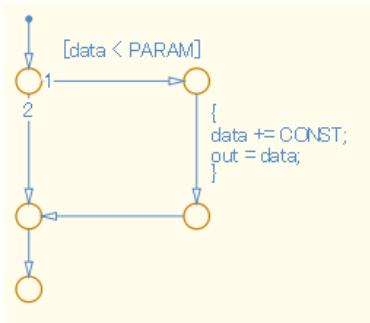


Sub ID c

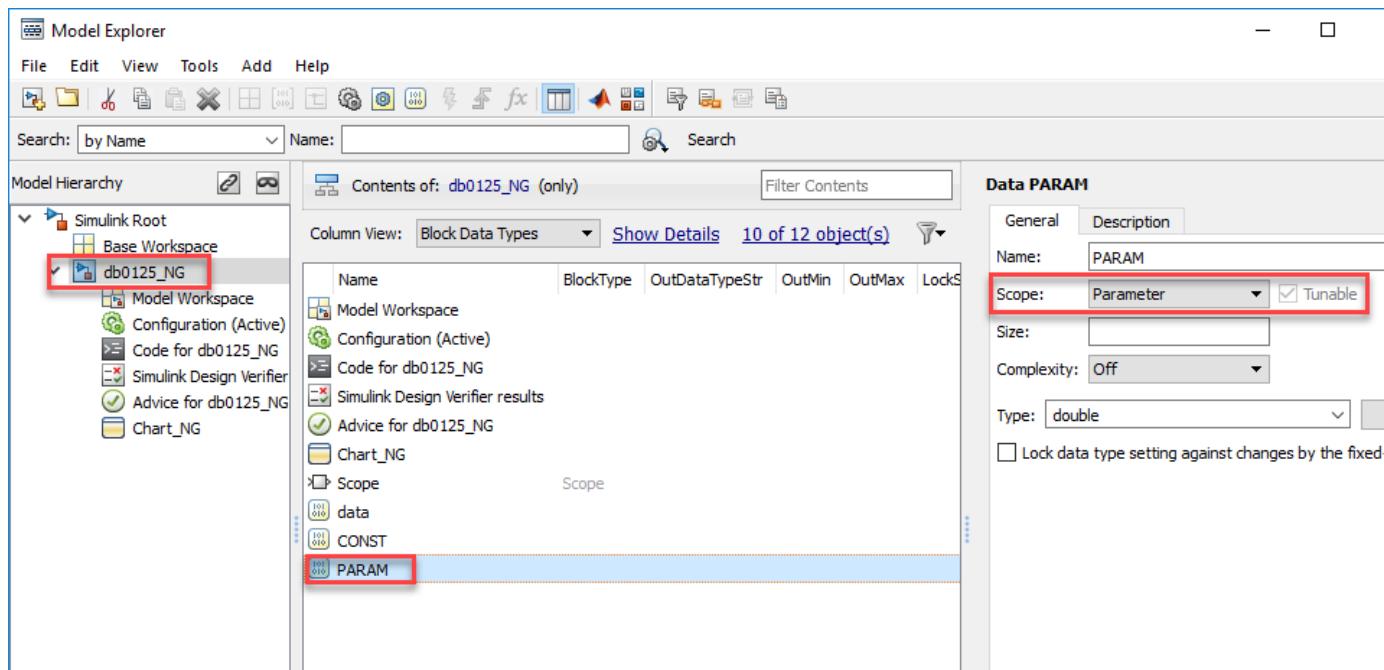
Data objects shall not be defined with Scope set to Parameter at the machine level.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Scope has set Parameter local data at the machine level.



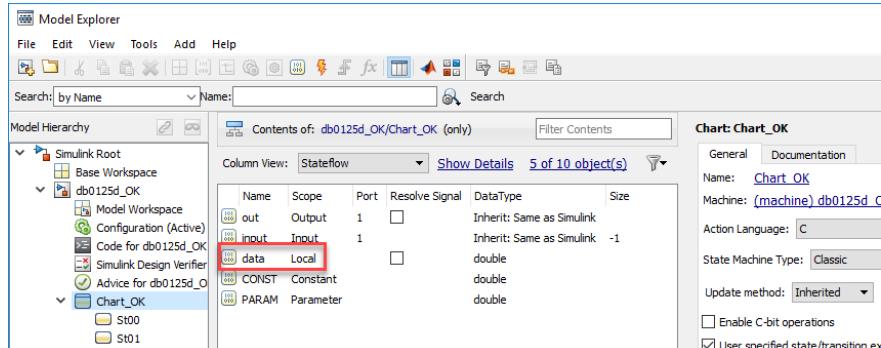
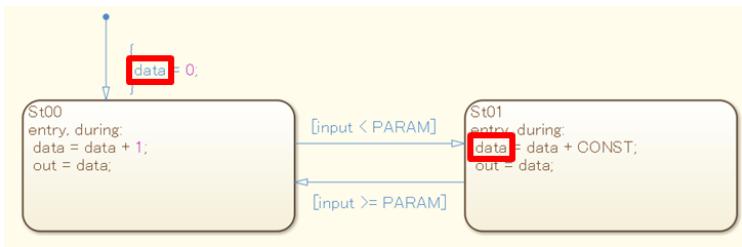
Sub ID d

A Stateflow block with parent-child relationships shall not include stateflow data with the same name.

Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect

A Stateflow block with parent-child relationships has local data with the same name.

The screenshots show the MATLAB Model Explorer interface. The top screenshot shows the 'db0125d_NG/Chart_NG' workspace, and the bottom screenshot shows the 'db0125d_NG/Chart_NG/St01' workspace. Both screens highlight a 'data' variable in red, which is defined as 'Local' in both cases. This is a violation of Stateflow rules, as local data should not have the same name as machine-level data.

Name	Scope	Port	Resolve Signal	DataType	Size
[101] out	Output	1	<input type="checkbox"/>	Inherit: Same as Simulink	
[101] input	Input	1	<input type="checkbox"/>	Inherit: Same as Simulink	-1
[101] data	Local		<input type="checkbox"/>	double	
[101] CONST	Constant			double	
[101] PARAM	Parameter			double	

Rationale

Sub ID a:

- When local data is defined at the machine level, it is shared with all blocks in the model. The data will not behave like a local variable and can be influenced by any operation.
- Adherence to the rules prevent the definition from disappearing when copying a Stateflow block to another model.

Sub IDs b, c:

- Adherence to the rules prevent the definition from disappearing when copying a Stateflow block to another model.

Sub ID d:

- When a Stateflow block with parent-child relationships includes stateflow data with the same name, readability decreases due to lack of clarity with regard to the influence of the stateflow data.

Verification

Model Advisor check: "Check definition of Stateflow data" (Simulink Check)

Last Changed

R2024b

See Also

- "Stateflow Data Properties" (Stateflow)
- "Use the Model Explorer with Stateflow Objects" (Stateflow)
- "Use State Hierarchy to Design Multilevel State Complexity" (Stateflow)

Version History

Introduced in R2020a

db_0126: Defining Stateflow events

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

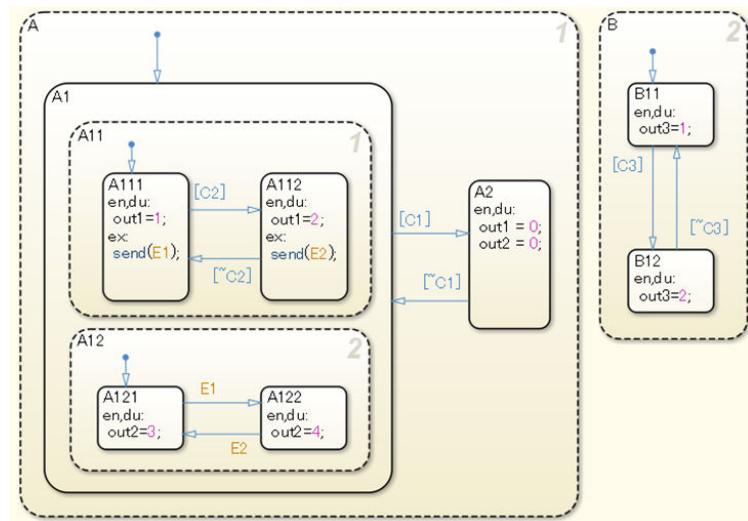
MATLAB Versions

All

Rule

Sub ID a

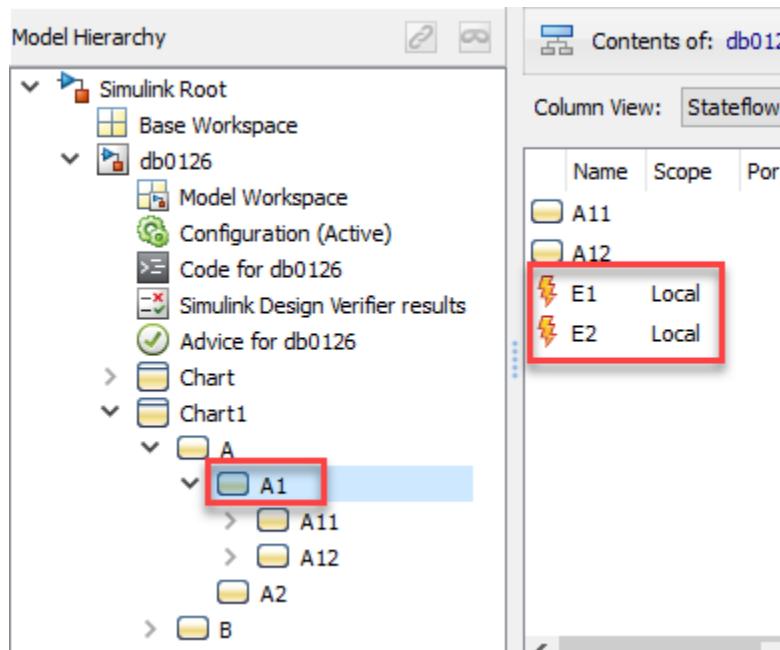
Stateflow events shall be defined by the smallest scope level in the Stateflow block being used.



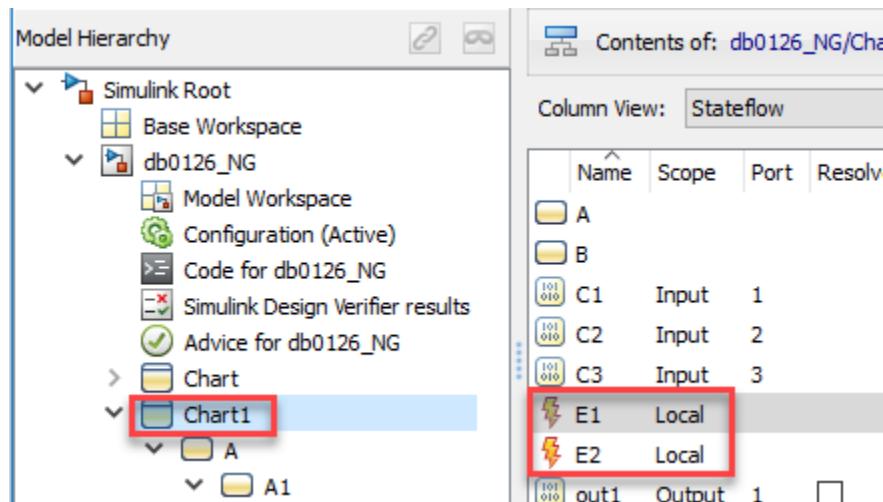
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- Limiting use locations increases reliability.

Verification

Model Advisor check: "Check definition of Stateflow events" (Simulink Check)

Last Changed

R2020a

See Also

- “Synchronize Model Components by Broadcasting Events” (Stateflow)
- “Use the Model Explorer with Stateflow Objects” (Stateflow)

Version History

Introduced in R2020a

jc_0701: Usable number for first index

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1/a2
- JMAAB — a1/a2

MATLAB Versions

All

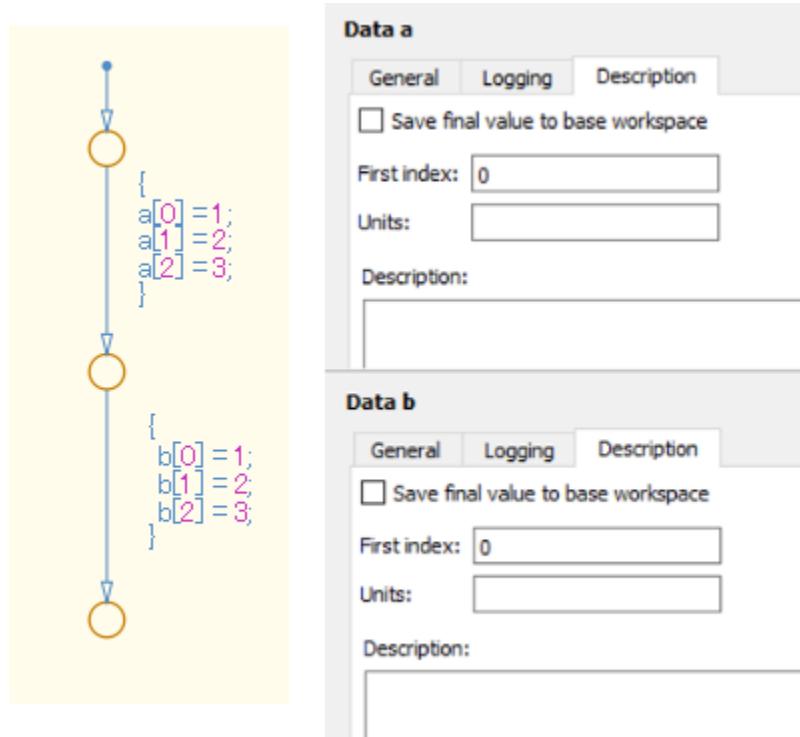
Rule

Sub ID a1

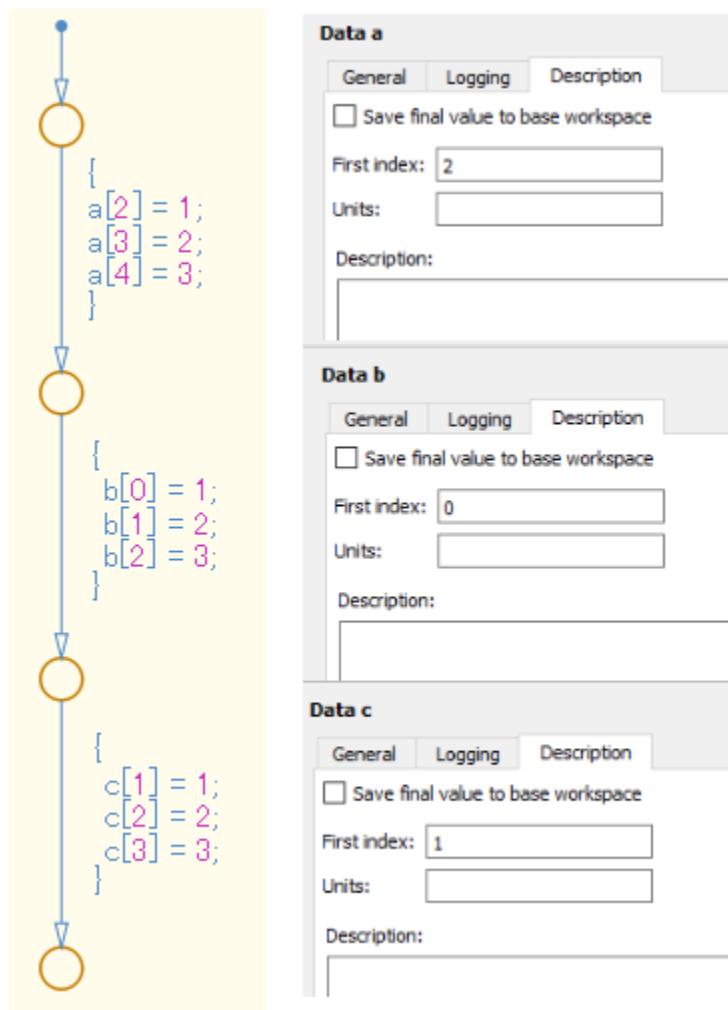
When Stateflow Chart property **Action language** is set to C, Stateflow data property **First Index** shall be set to 0.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

First Index is set to a combination of 0, 1, and 2.

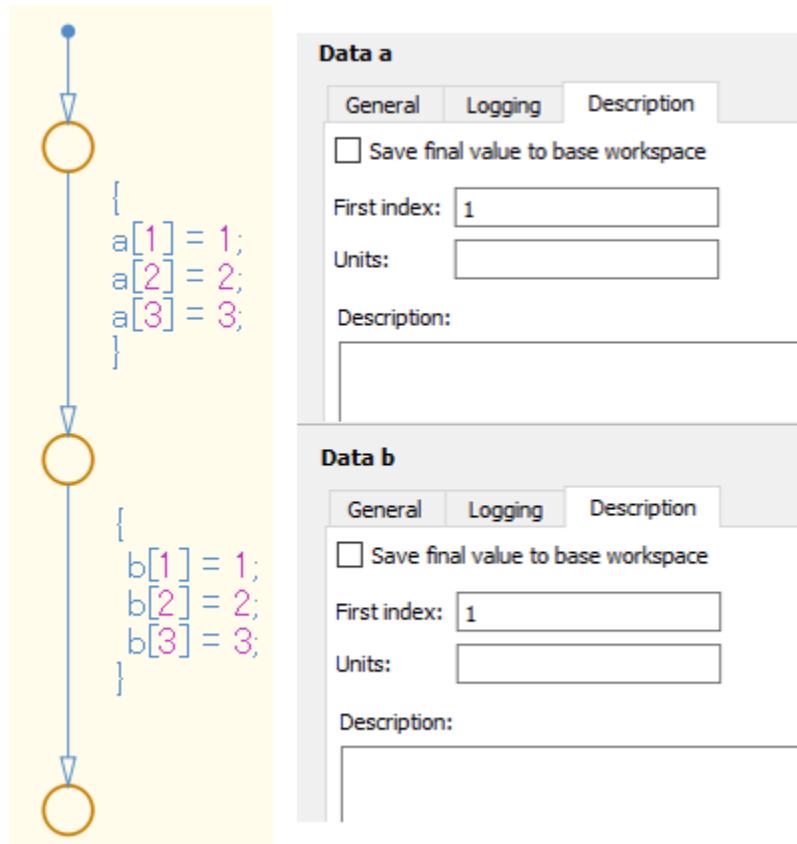


Sub ID a2

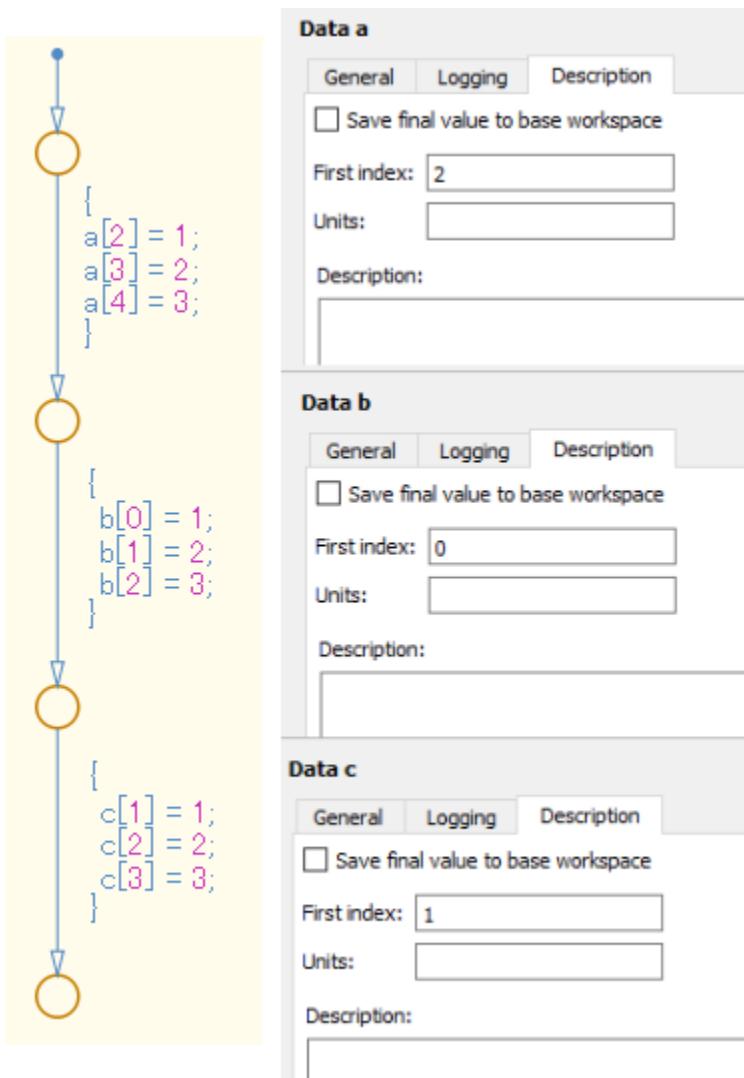
When Stateflow Chart property **Action language** is set to C, Stateflow data property **First Index** shall be set to 1.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

First Index is set to a combination of 0, 1, and 2.



Rationale

Sub ID a1:

- Logic becomes easier to understand when **First Index** is uniform.

Sub ID a2:

- Logic becomes easier to understand when **First Index** is uniform. However, C language is 0-based, which decreases the readability of the code as the index calculation process is 1-based. This is reflected in the generated code.

Verification

Model Advisor check: "Check usable number for first index" (Simulink Check)

Last Changed

R2020a

See Also

- “Differences Between MATLAB and C as Action Language Syntax” (Stateflow)
- “Set Data Properties” (Stateflow)

Version History

Introduced in R2020a

jc_0712: Execution timing for default transition path

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Stateflow Chart property **Execute (enter) chart at initialization** shall not be selected.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Using the same settings for each Stateflow Chart prevents the model from being misinterpreted.
- Use caution when referencing an input signal using the default transition line when property **Execute (enter) chart at initialization** is selected.

Verification

Model Advisor check: "Check execution timing for default transition path" (Simulink Check)

Last Changed

R2020a

See Also

- “Transition Between Operating Modes” (Stateflow)
- “Execution of a Stateflow Chart” (Stateflow)

Version History

Introduced in R2020a

jc_0722: Local data definition in parallel states

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

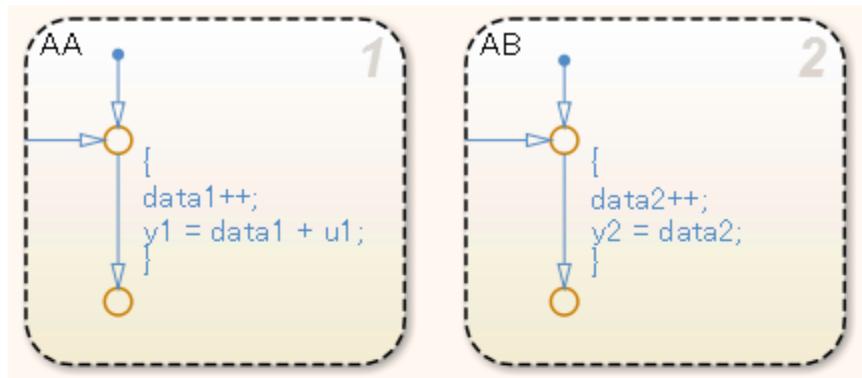
Local variables that are completed in one state shall be defined in that state.

Custom Parameter

Not Applicable

Example — Correct

Local variables are defined in the state being used.



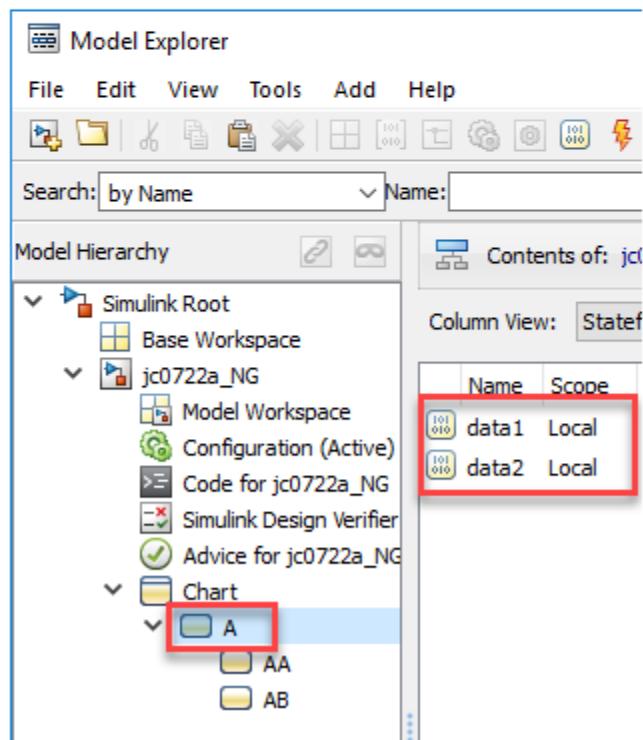
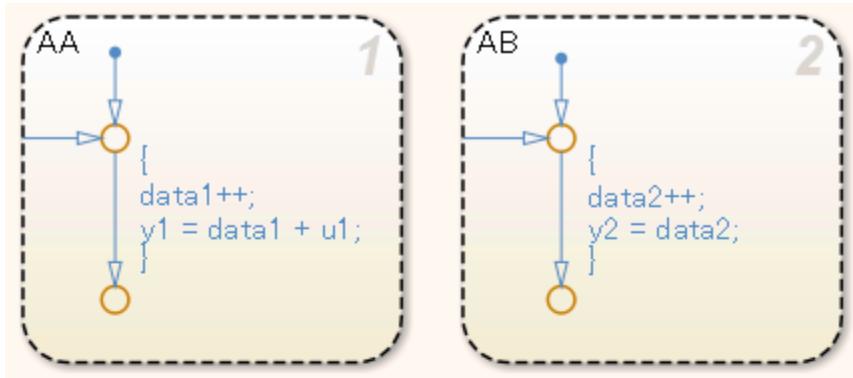
Model Explorer windows showing workspace contents for jc0722a_OK:

Name	Scope
data1	Local

Name	Scope
data2	Local

Example — Incorrect

Local variables are not defined in the state being used.



Rationale

Sub ID a:

- Readability and maintainability can be improved by explicitly limiting the valid range of the variables, thereby avoiding unintended references and changes.

Verification

Model Advisor check: "Check scope of data in parallel states" (Simulink Check)

Last Changed

R2020a

See Also

- “Represent Operating Modes by Using States” (Stateflow)
- “Parallel and Exclusive States” (Stateflow)

Version History

Introduced in R2020a

Stateflow Diagram

jc_0797: Unconnected transitions / states / connective junctions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

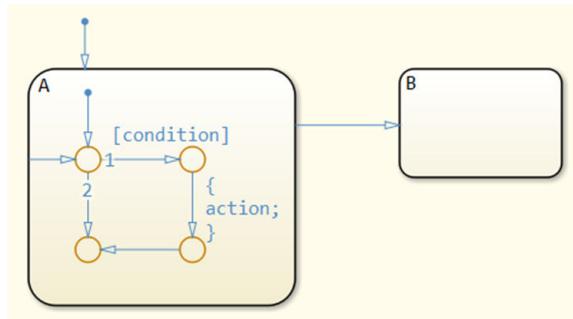
Sub ID a

Stateflow Chart shall not have unconnected transitions.

Custom Parameter

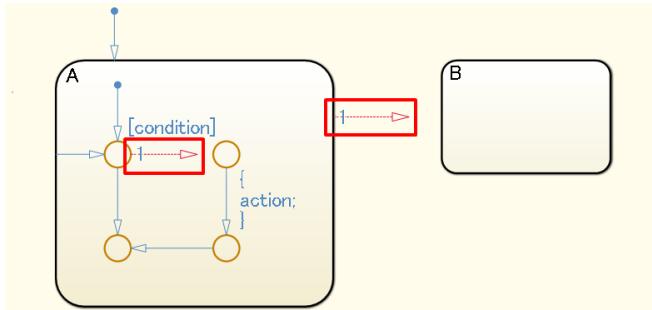
Not Applicable

Example — Correct



Example — Incorrect

There are unconnected transitions.



Sub ID b

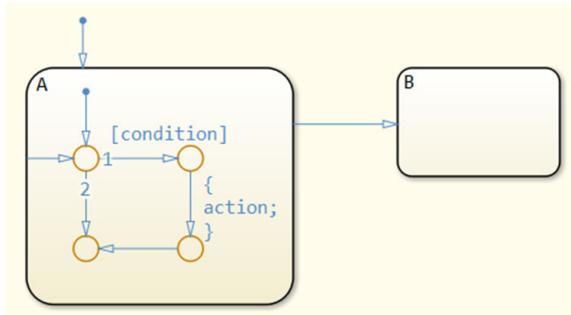
Stateflow Chart shall not have unconnected exclusive (OR) states and connective junctions without a transition source.

Custom Parameter

Not Applicable

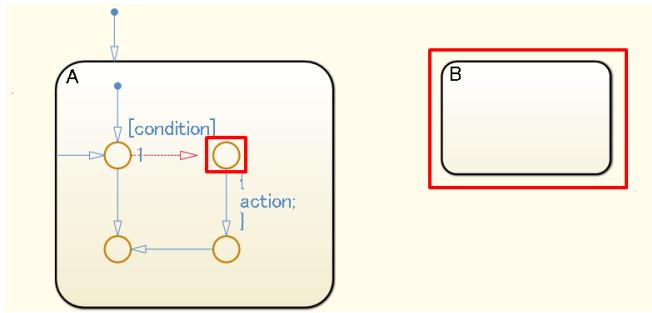
Example — Correct

Does not have unconnected exclusive (OR) states or connective junctions without a transition source.



Example — Incorrect

There are unconnected exclusive (OR) states and connective junctions without a transition source.



Rationale

Sub IDs a, b:

- Unconnected transitions can result in adverse effects, such as misinterpretation of simulation results or failure to generate code.

Verification

Model Advisor check: "Check for unconnected objects in Stateflow Charts" (Simulink Check)

Last Changed

R2020a

See Also

- "Transition Between Operating Modes" (Stateflow)
- "Combine Transitions and Junctions to Create Branching Paths" (Stateflow)
- "Define Exclusive and Parallel Modes by Using State Decomposition" (Stateflow)
- "Transition Between Exclusive States" (Stateflow)
- "Represent Multiple Paths by Using Connective Junctions" (Stateflow)

Version History

Introduced in R2020a

db_0137: States in state machines

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

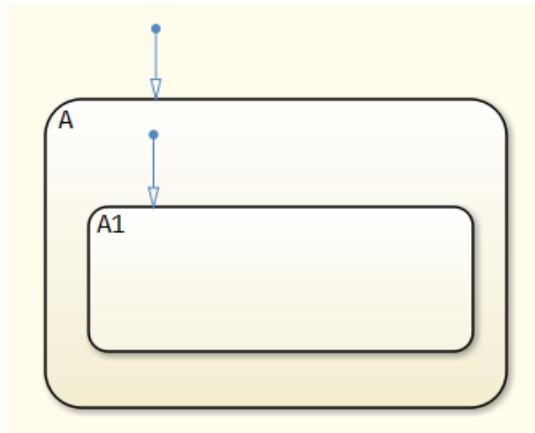
When the **Decomposition** for the Chart block or State is set to OR (Exclusive), there shall be at least two states in the hierarchy.

Custom Parameter

Not Applicable

Example — Incorrect

The hierarchy contains only one state when the **Decomposition** option is set to OR (Exclusive)



Rationale

Sub ID a:

- Redundant descriptions impair readability.
- Generated code includes unnecessary state variables.

Verification

Model Advisor check: "Check for state in state machines" (Simulink Check)

Last Changed

R2020a

See Also

- "Represent Operating Modes by Using States" (Stateflow)
- "Define Exclusive and Parallel Modes by Using State Decomposition" (Stateflow)

Version History

Introduced in R2020a

jc_0721: Usage of parallel states

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

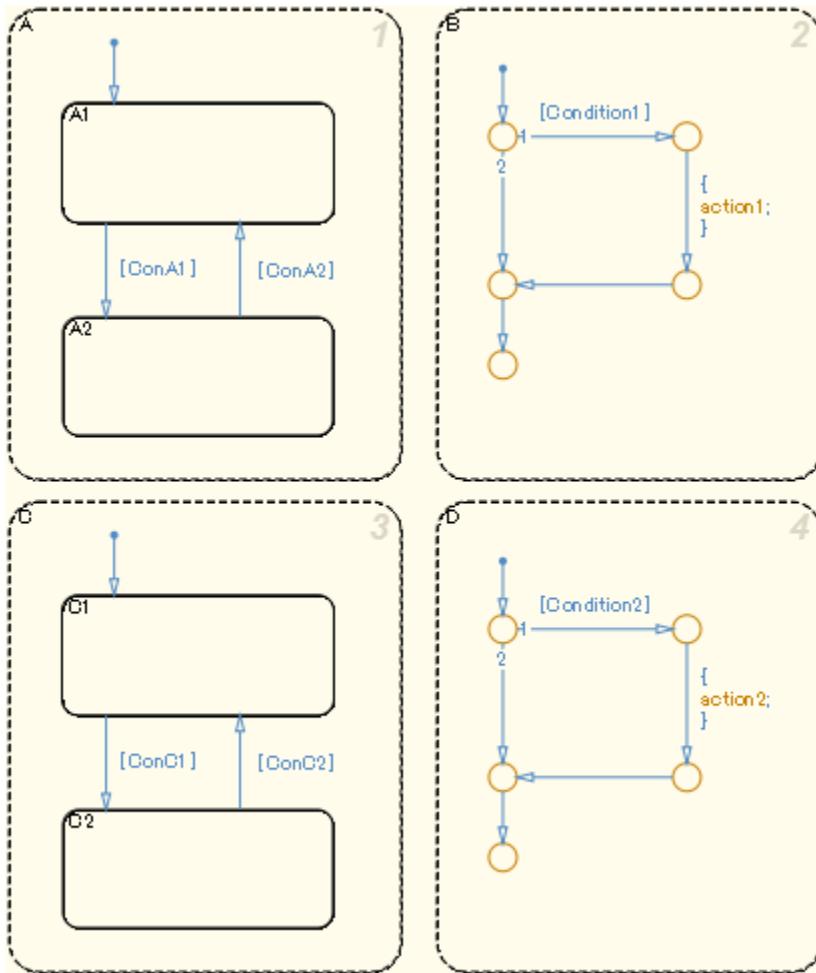
Rule

Sub ID a

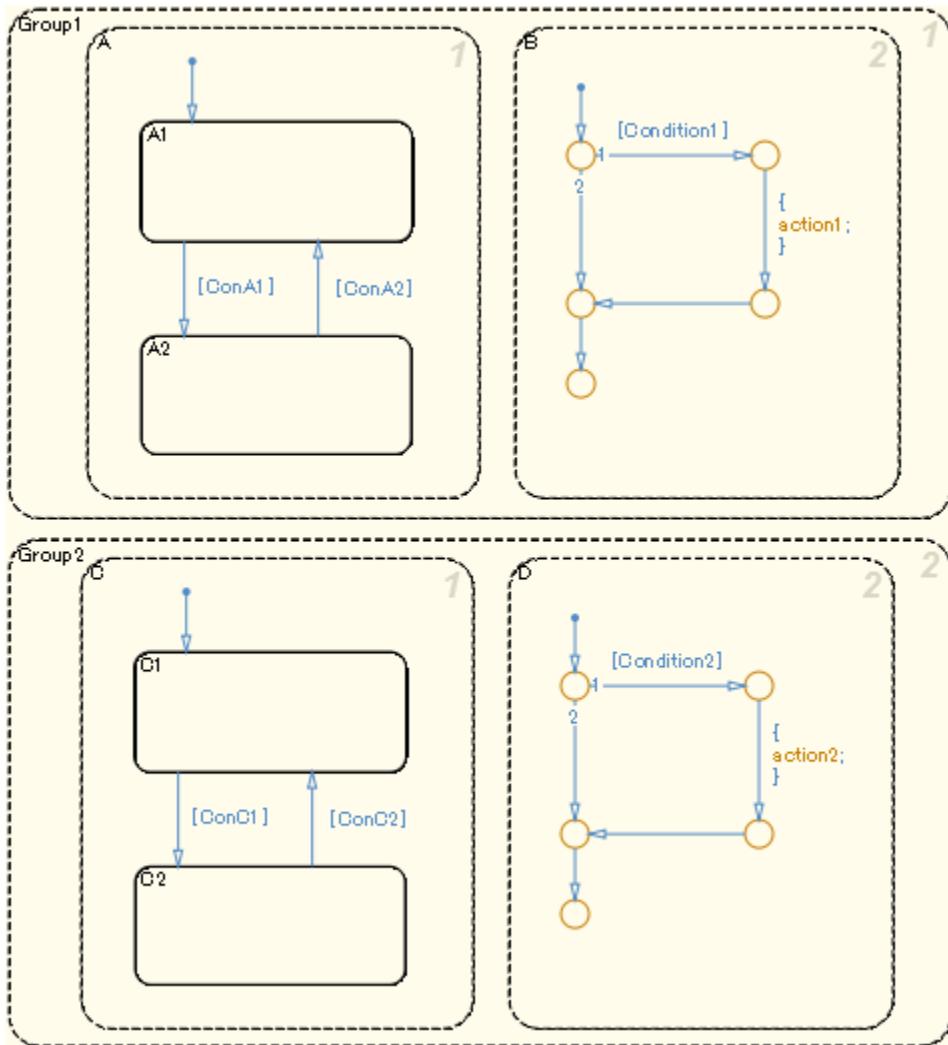
Substates of parallel states shall not be parallel states.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Substates of parallel states are parallel states.



Rationale

Sub ID a:

- Behavior is not affected by nesting parallel states in a parent superstate.
- Hierarchization of the parallel state decreases readability.

Verification

Model Advisor check: "Check usage of parallel states" (Simulink Check)

Last Changed

R2020a

See Also

- “Represent Operating Modes by Using States” (Stateflow)
- “Parallel and Exclusive States” (Stateflow)
- “Use State Hierarchy to Design Multilevel State Complexity” (Stateflow)

Version History

Introduced in R2020a

db_0129: Stateflow transition appearance

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

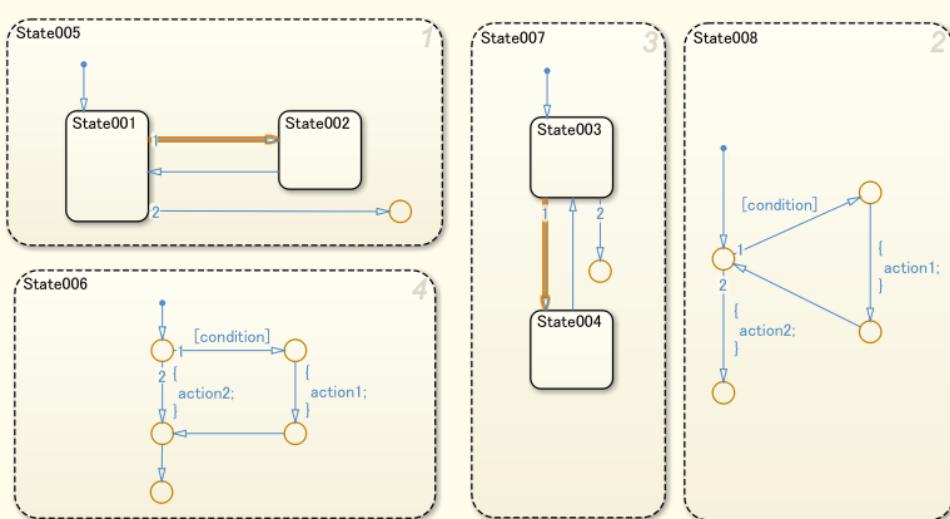
Transition lines shall be drawn vertically or horizontally.

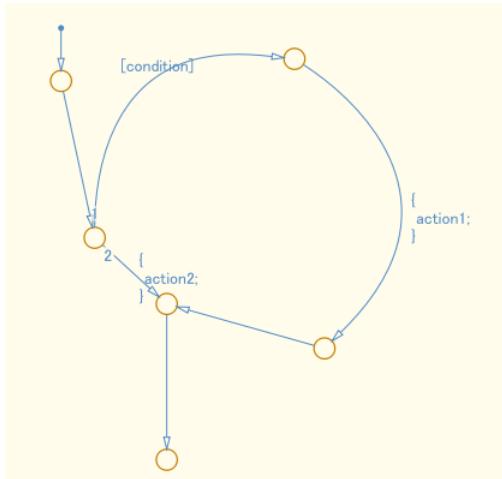
Diagonal lines can be used for flow charts.

Custom Parameter

Not Applicable

Example — Correct

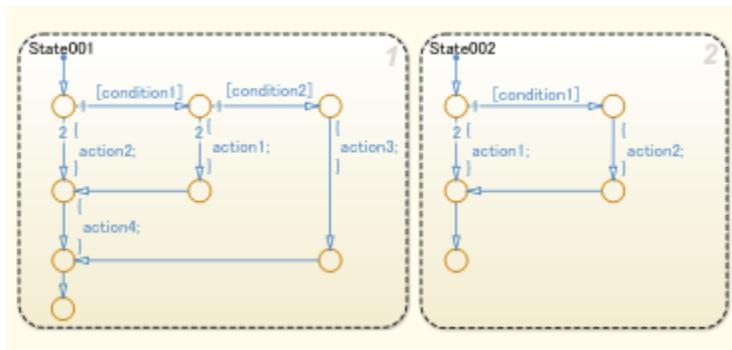
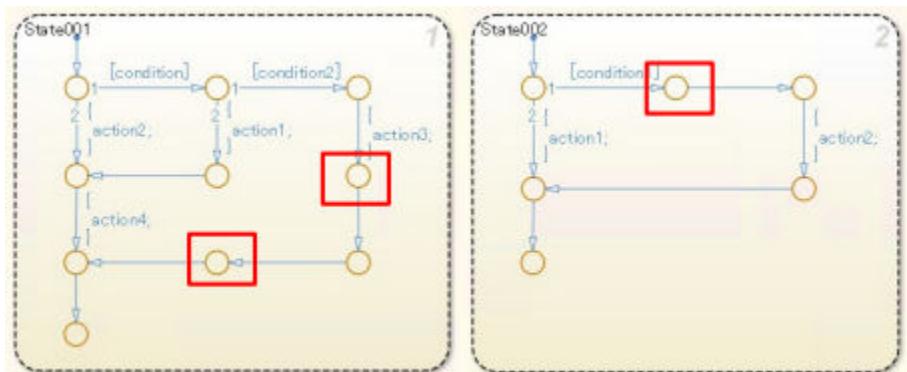


Example — Incorrect**Sub ID b**

Unnecessary connective junctions shall not be used.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Rationale

Sub ID a:

- Consistent application of transition lines improves readability.

Sub ID b:

- Transitions can be difficult to understand when unnecessary connective junctions are used.

Verification

Model Advisor check: "Check for Stateflow transition appearance" (Simulink Check)

Last Changed

R2024b

See Also

- "Transition Between Operating Modes" (Stateflow)

Version History

Introduced in R2020a

jc_0531: Default transition

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b, c, d, e, f, g
- JMAAB — a, b, c, d, e, f, g

MATLAB Versions

All

Rule

Sub ID a

When **Decomposition** of a Stateflow Chart is **Exclusive (OR)**, the default transition shall connect at the top of the Chart block.

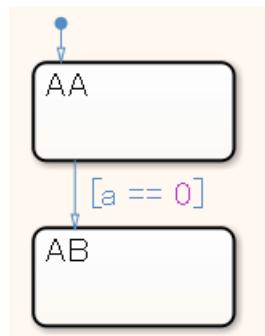
When **Decomposition** of the state is **Exclusive (OR)**, the default transition shall connect immediately beneath the state.

Custom Parameter

Not Applicable

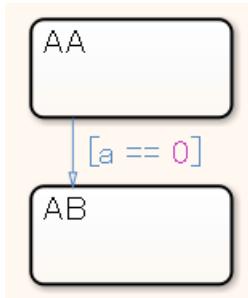
Example — Correct

The default transition line is connected at the top.



Example — Incorrect

The default transition line is not connected.



Sub ID b

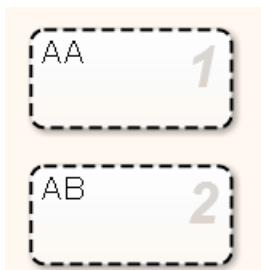
When **Decomposition** is set to “Parallel (AND)”, the default transition line shall not be connected.

Custom Parameter

Not Applicable

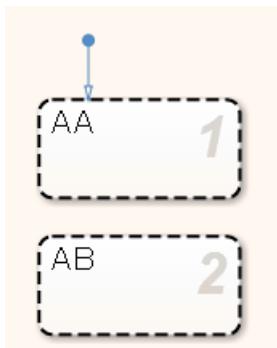
Example — Correct

Decomposition of the parent object for states AA and AB is set to **Parallel (AND)**, which makes states AA and AB parallel states. The default transition line is not connected for these parallel states.



Example — Incorrect

A default transition line is connected for parallel state AA.



Sub ID c

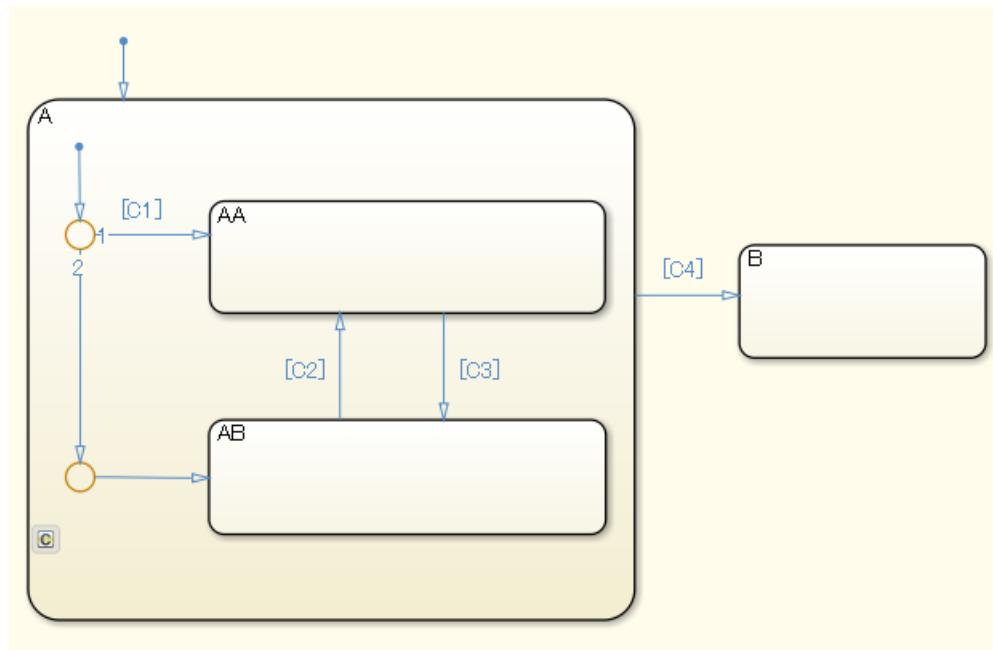
A level shall not have multiple default transitions.

Custom Parameter

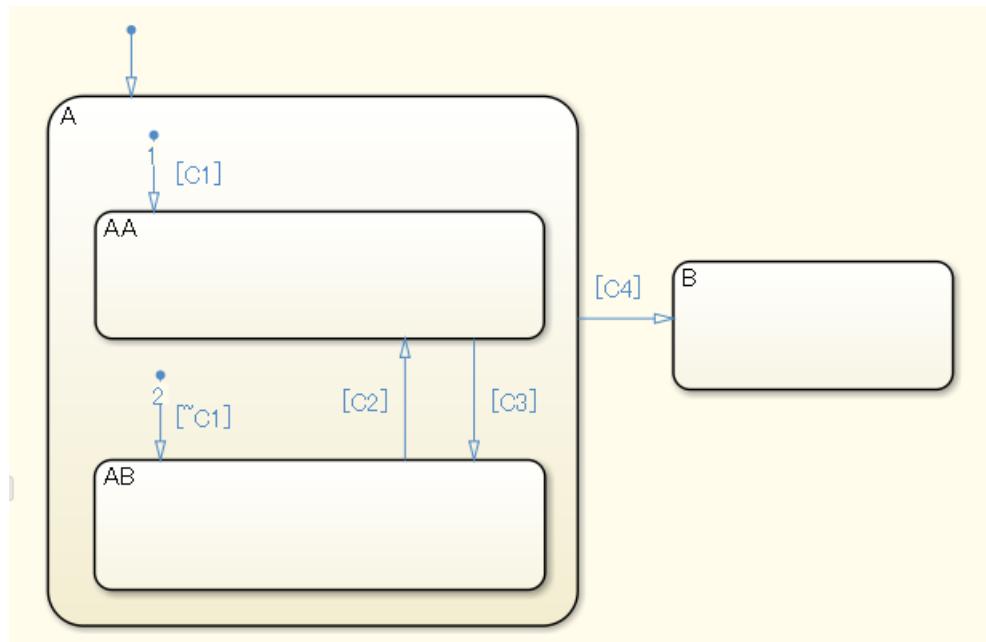
Not Applicable

Example — Correct

The level does not have multiple default transitions

**Example — Incorrect**

Multiple default transitions are included in the same level of state A.



Sub ID d

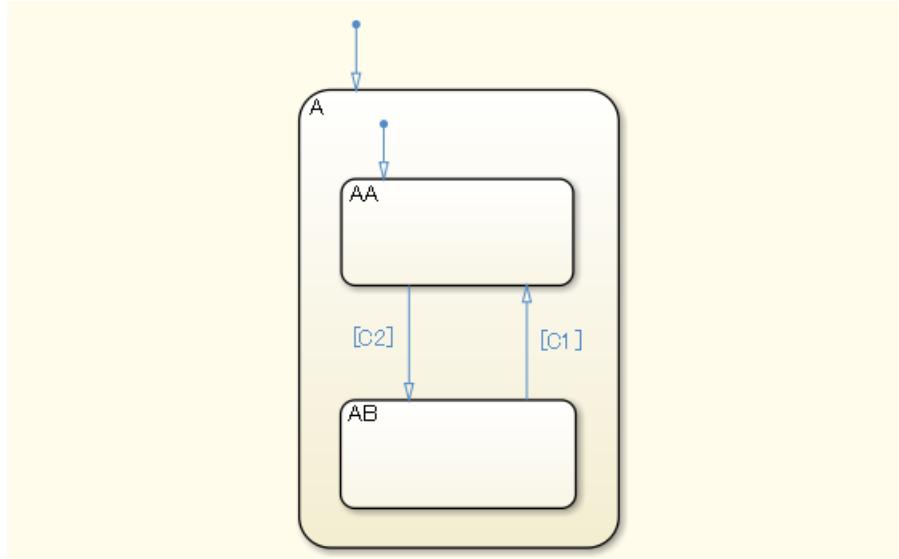
Default transitions shall be connected directly and positioned vertically to the upper part of the state or connective junction.

Custom Parameter

Not Applicable

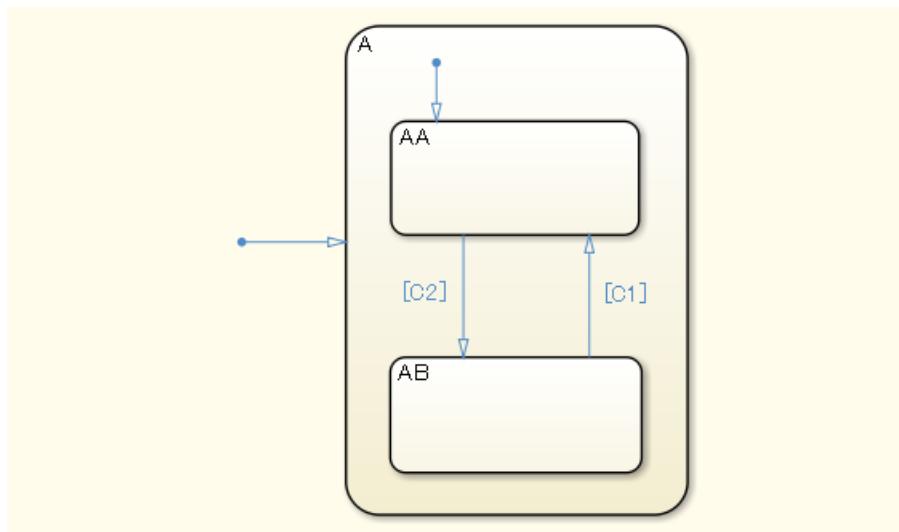
Example — Correct

The default transition is connected vertically to the upper part of the state.



Example — Incorrect

The default transition of state A is not connected vertically to the upper part of the state.



Sub ID e

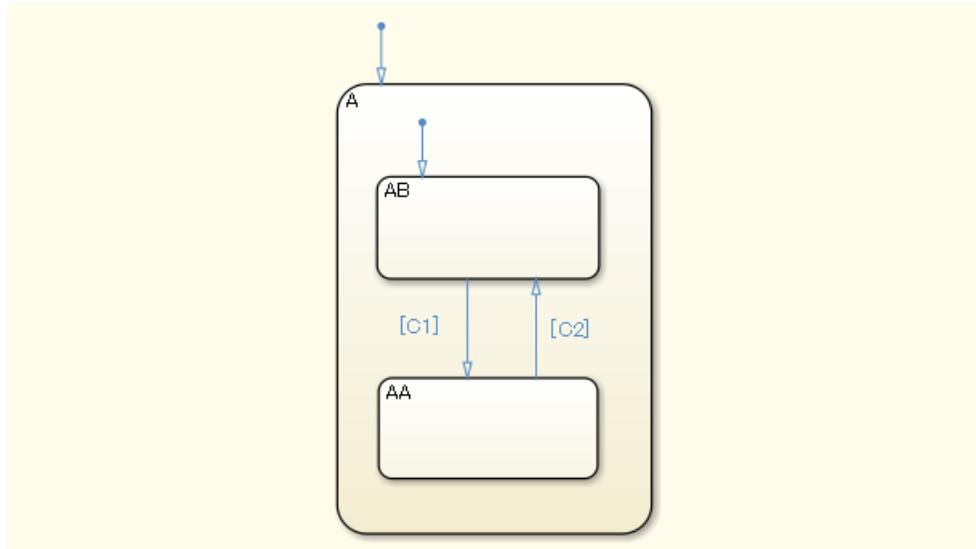
The destination state or destination connective junction for the default transition shall be positioned to the top left in the same level.

Custom Parameter

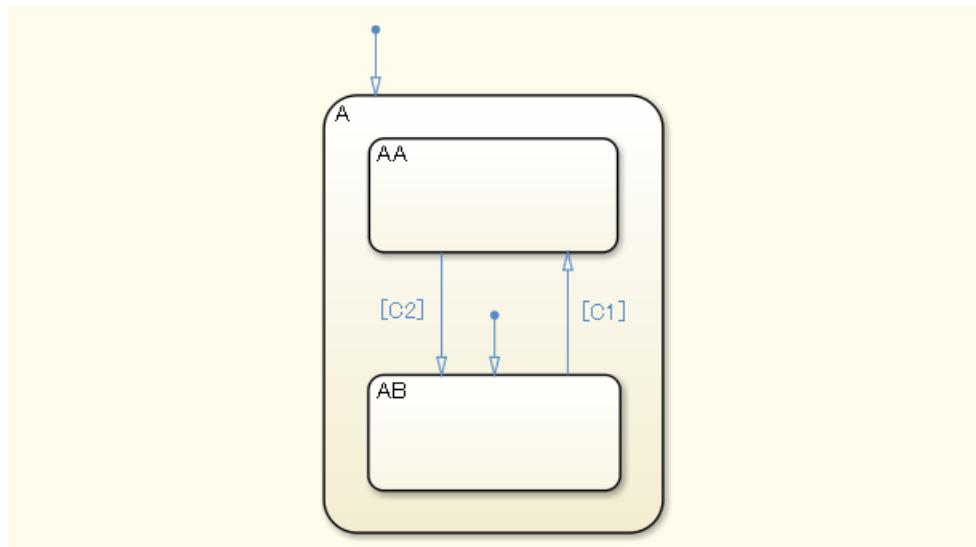
Not Applicable

Example — Correct

The default transition is positioned to the top left in the same level.

**Example — Incorrect**

The default transition of state AB is not positioned to the top left in the same level.



Sub ID f

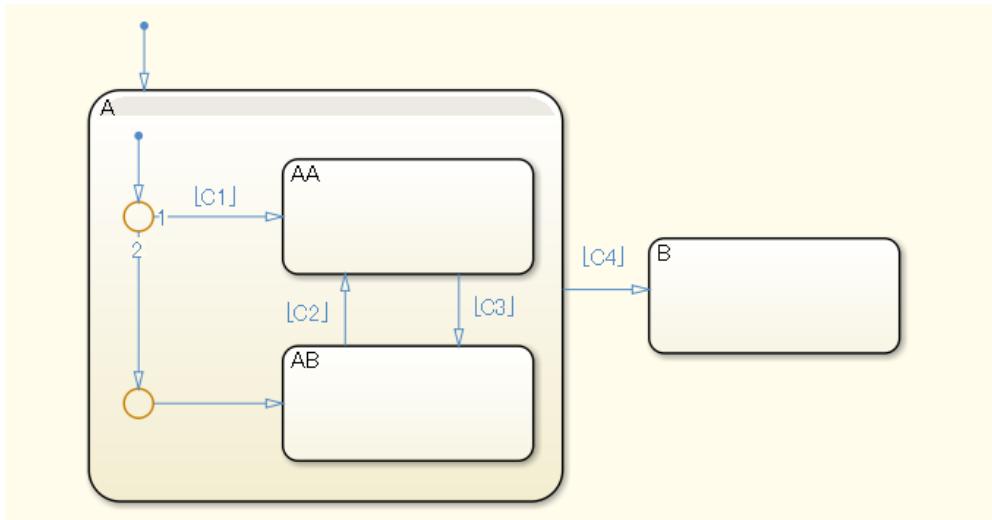
Default transitions shall not extend beyond the boundaries of the state.

Custom Parameter

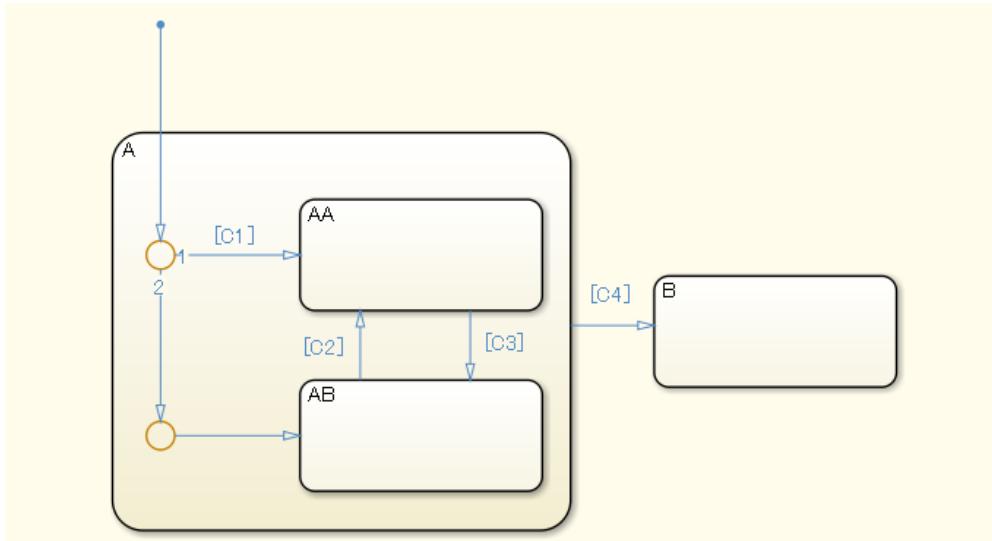
Not Applicable

Example — Correct

The default transition is within the boundaries of the state.

**Example — Incorrect**

The default transition extends beyond the boundaries of the state.



Sub ID g

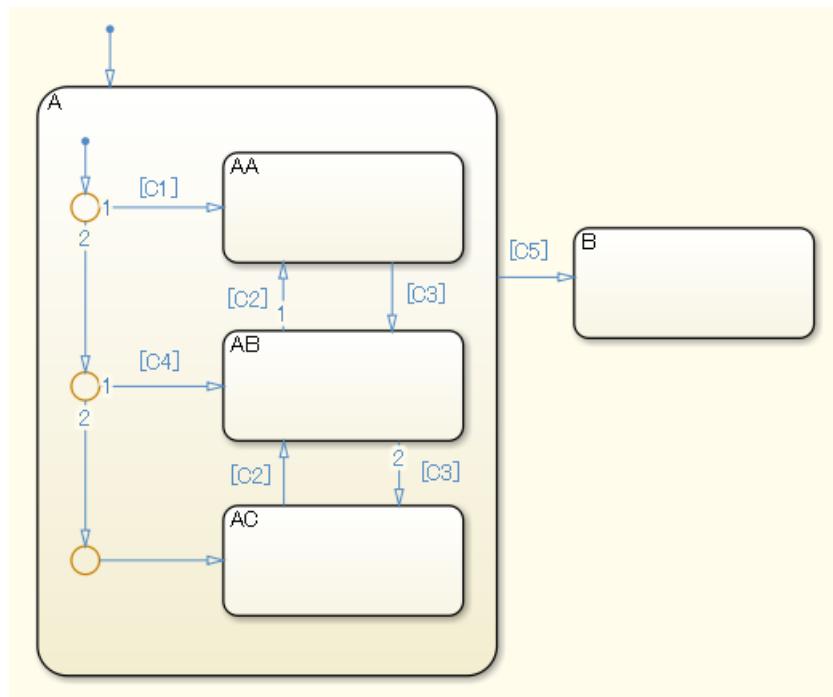
Configuration parameter **No unconditional default transitions** shall be set to **Error** to ensure that in the transition path for the default transition, the path with the lowest priority is an unconditional transition.

Custom Parameter

Not Applicable

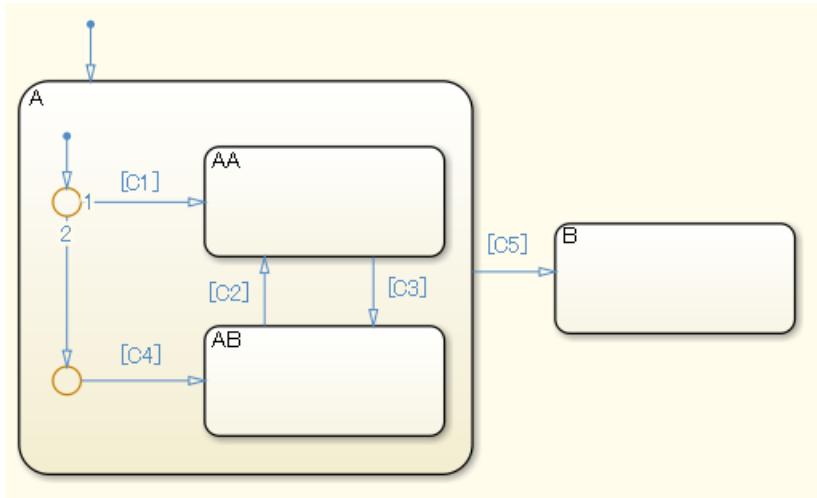
Example — Correct

The path with the lowest priority in the transition path for the default transition is an unconditional transition.



Example — Incorrect

The path with the lowest priority in the transition path for the default transition is not an unconditional transition.



Rationale

Sub ID a:

- Simulation errors can occur when a state chart does not include default transition lines.
- When default transitions are included in a flow chart, it is impossible to determine whether this is intentional or through failure to insert them.

Sub ID b:

- Readability improves when there are no unnecessary default transitions.

Sub ID c:

- The state may not function as intended and produce a warning when multiple default transitions are included in the same level.

Sub ID d:

- Readability decreases when there are curves or variations in the angle or position of default transitions.

Sub ID e:

- Readability decreases when there are variations in the position of the transition destination state or transition destination connective junction for the default transition.

Sub ID f

- Readability decreases when a default transition extends beyond the boundary of a state and intersects with state boundaries and expressions.

Sub ID g:

- When there is not an unconditional transition in the transition path of the default transition, the transition destination disappears if all conditions of the transition path are not met. This can result in unintended behavior.

Verification

Model Advisor check: “Check default transition placement in Stateflow charts” (Simulink Check)

Last Changed

R2020a

See Also

- “Define Exclusive and Parallel Modes by Using State Decomposition” (Stateflow)
- “Transition Between Operating Modes” (Stateflow)
- “How Stateflow Objects Interact During Execution” (Stateflow)

Version History

Introduced in R2020a

jc_0723: Prohibited direct transition from external state to child state

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

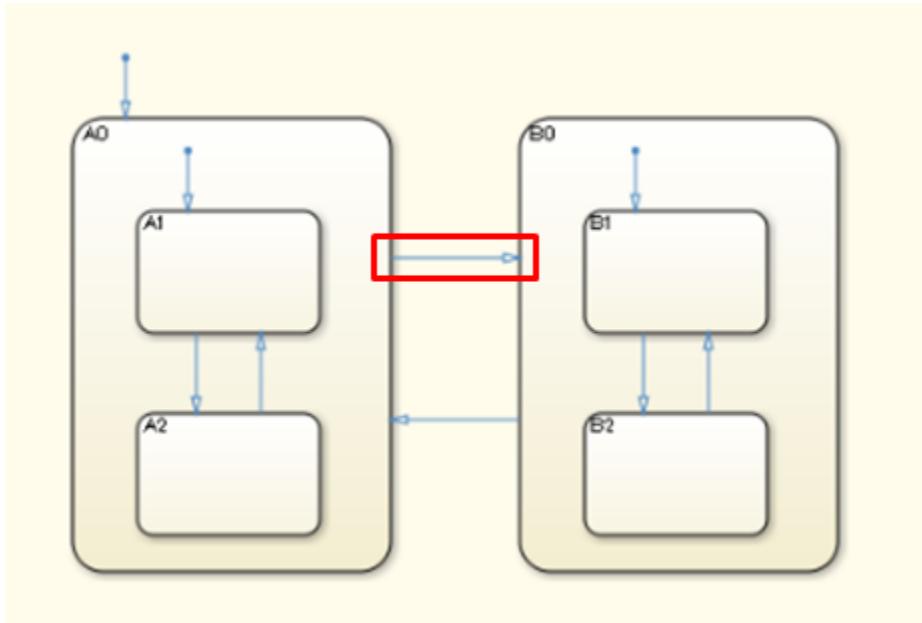
Transitions from one state directly to an external child state shall be prohibited.

Custom Parameter

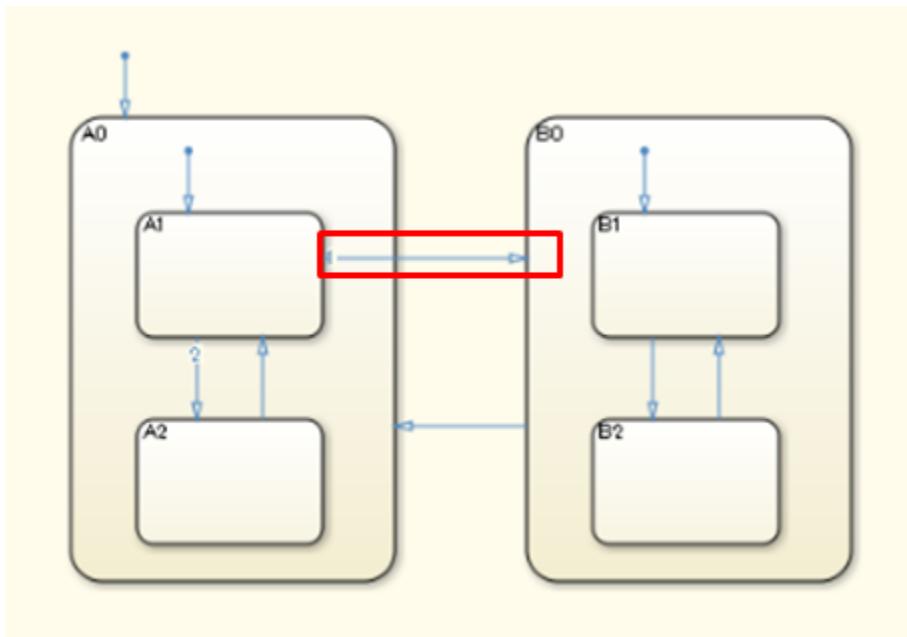
Not Applicable

Example — Correct

Transition from parent state to parent state.

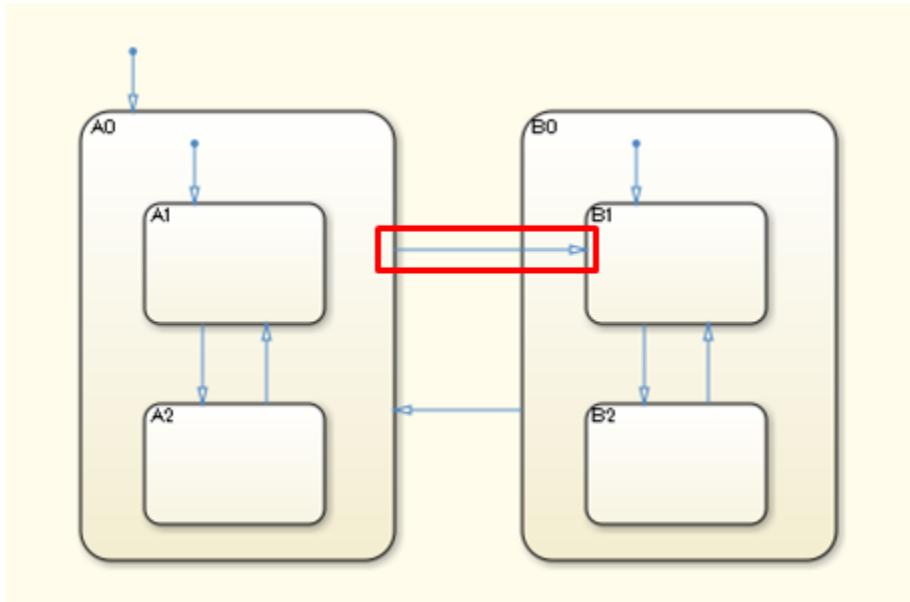


Transition from child state to another parent state.

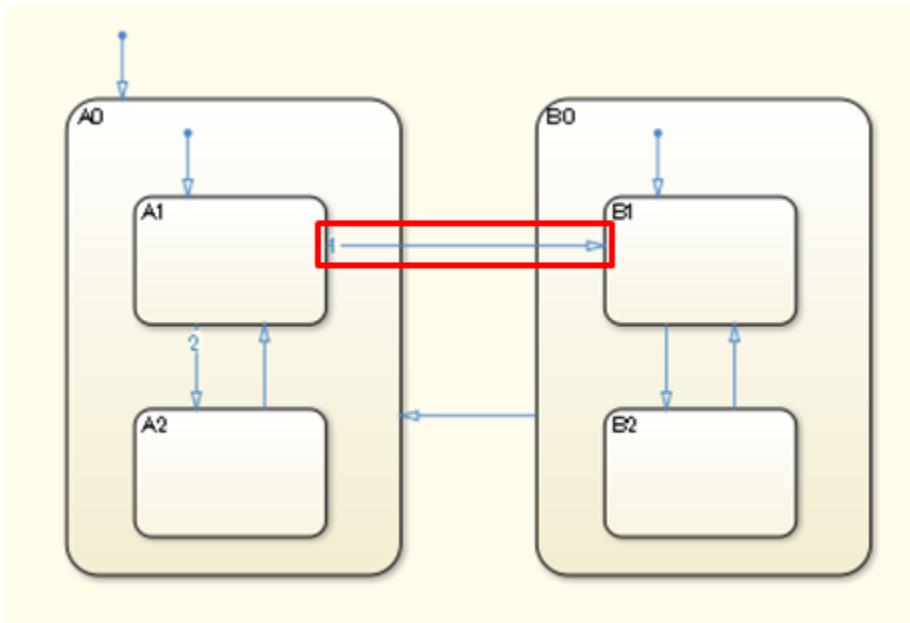


Example — Incorrect

Direct transition from an external state to a child state in a different state.



Direct transition from an external child state to a child state in a different state.



Rationale

Sub ID a:

- Direct transitions between child states can complicate the states and decrease readability.

Verification

Model Advisor check: "Check usage of transitions to external states" (Simulink Check)

Last Changed

R2020a

See Also

- “Transition Between Operating Modes” (Stateflow)

Version History

Introduced in R2020a

jc_0751: Backtracking prevention in state transition

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

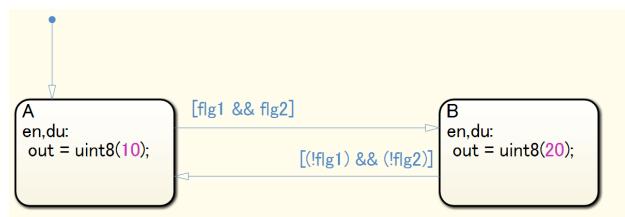
Connective junctions shall not be used to separate complex conditions.

Custom Parameter

Not Applicable

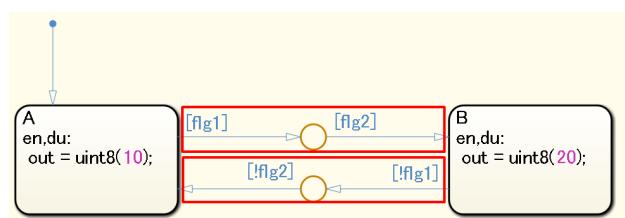
Example — Correct

Connective junctions are not used to separate complex conditions.



Example — Incorrect

Connective junctions are used to separate complex conditions.



Rationale

Sub ID a:

- Deviation from the rule can cause backtracking, which results in unintended behavior.

Verification

Model Advisor check: "Check for unexpected backtracking in state transitions" (Simulink Check)

Last Changed

R2020a

See Also

- "Transition Between Operating Modes" (Stateflow)
- "Combine Transitions and Junctions to Create Branching Paths" (Stateflow)
- "Evaluate Transitions" (Stateflow)

Version History

Introduced in R2020a

jc_0760: Starting point of internal transition

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

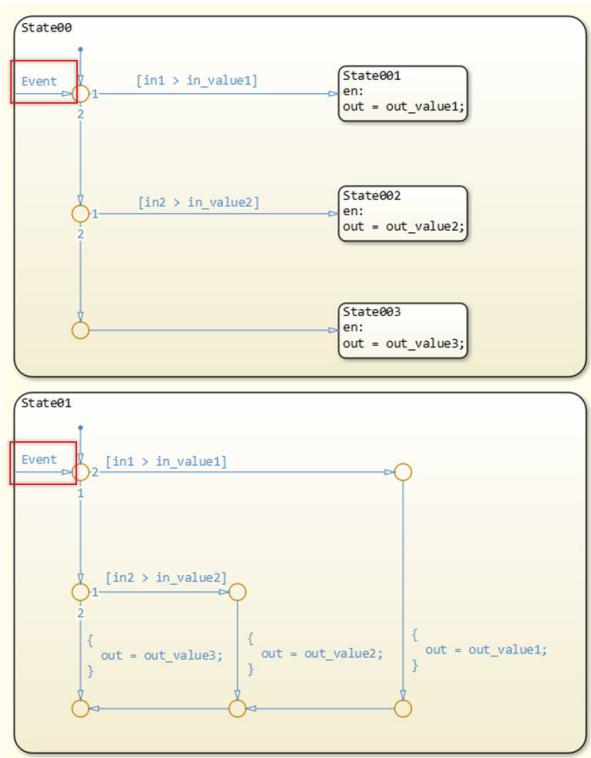
Internal transition lines shall start from the left edge of the state.

Custom Parameter

Not Applicable

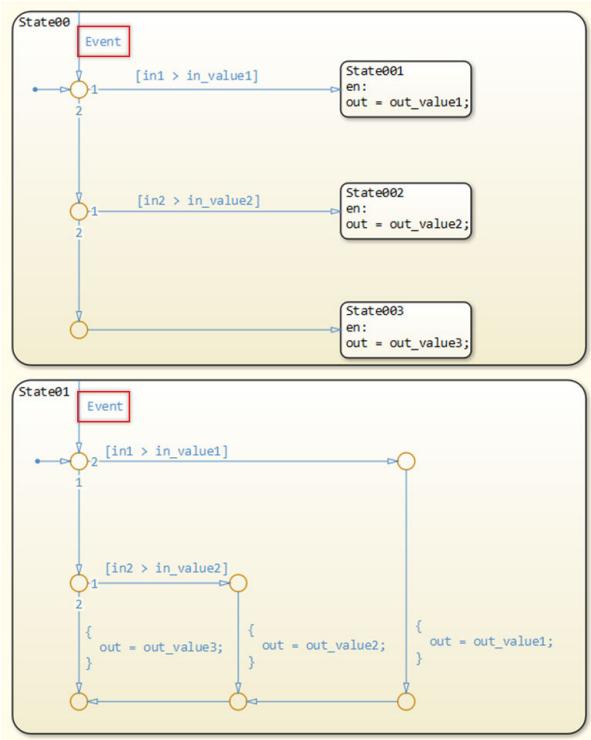
Example — Correct

The inner transition line begins at the left edge of the state.



Example — Incorrect

The inner transition line does not begin at the left edge of the state.



Rationale

Sub ID a:

- Adherence to the rule improves readability.

Verification

Model Advisor check: "Check starting point of internal transition in Stateflow" (Simulink Check)

Last Changed

R2020a

See Also

- "Control Chart Execution by Using Inner Transitions" (Stateflow)

Version History

Introduced in R2020a

jc_0763: Usage of multiple internal transitions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1/a2
 - JMAAB — a1/a2

MATLAB Versions

All

Rule

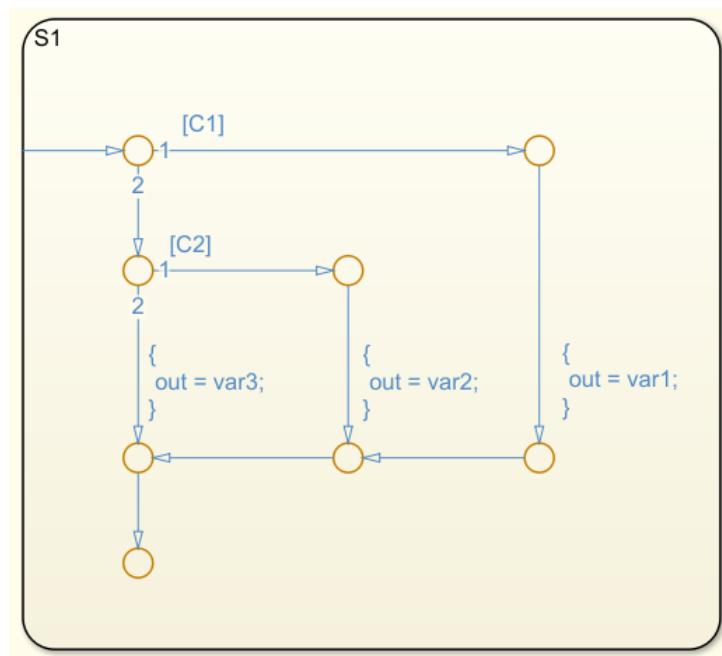
Sub ID a1

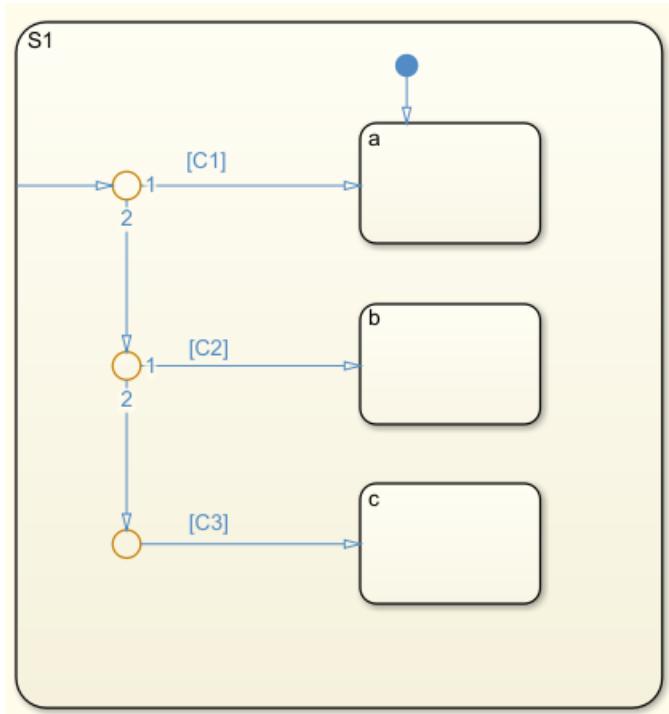
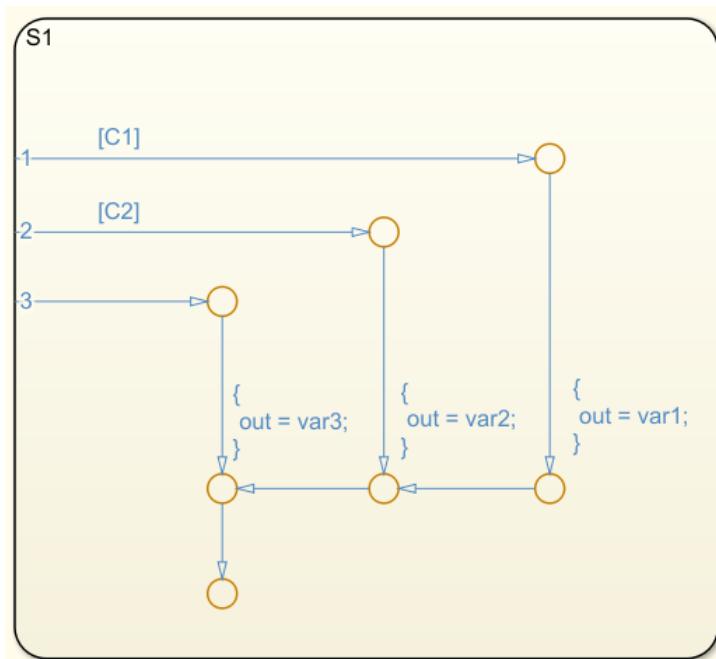
Multiple internal transitions shall not be used in a single state.

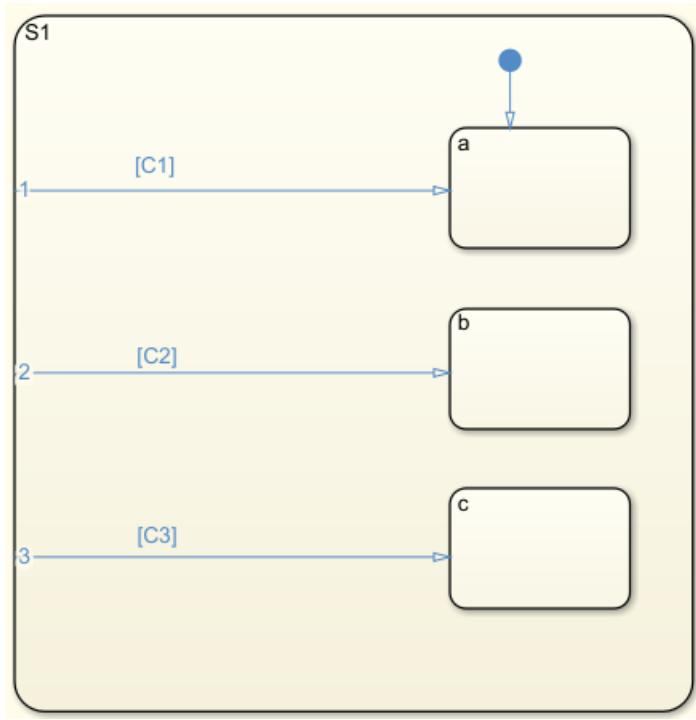
Custom Parameter

Not Applicable

Example – Correct



**Example — Incorrect**



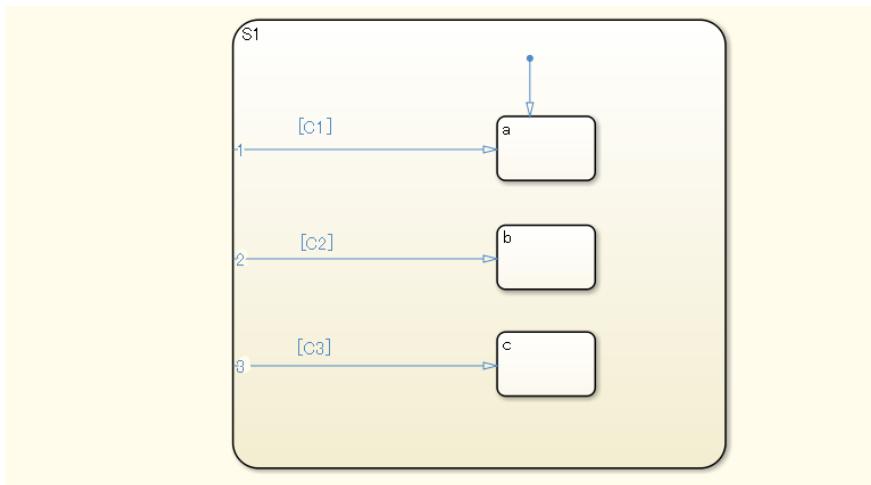
Sub ID a2

When multiple internal transitions are used in a single state, they shall be listed from top to bottom in the order of execution.

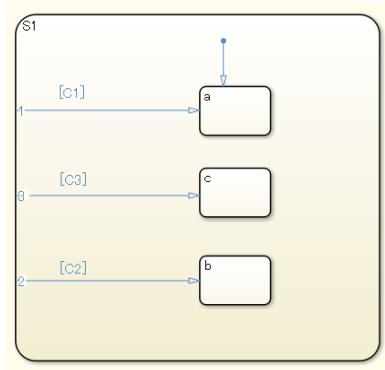
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a1:

- The number of transition conditions is unclear when multiple internal transitions are used. By limiting the use of internal transitions to a single use, transitions are clearer and readability improves.

Sub ID a2:

- Using multiple internal transitions can prevent transition lines from crossing and simplifies state transitions.
- Arranging internal transitions in execution order improves readability.

Verification

Model Advisor check: "Check usage of internal transitions in Stateflow states" (Simulink Check)

Last Changed

R2020a

See Also

- "Overview of Stateflow Objects" (Stateflow)
- "Transition Between Operating Modes" (Stateflow)
- "Types of Chart Execution" (Stateflow)

Version History

Introduced in R2020a

jc_0762: Prohibition of state action and flow chart combination

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

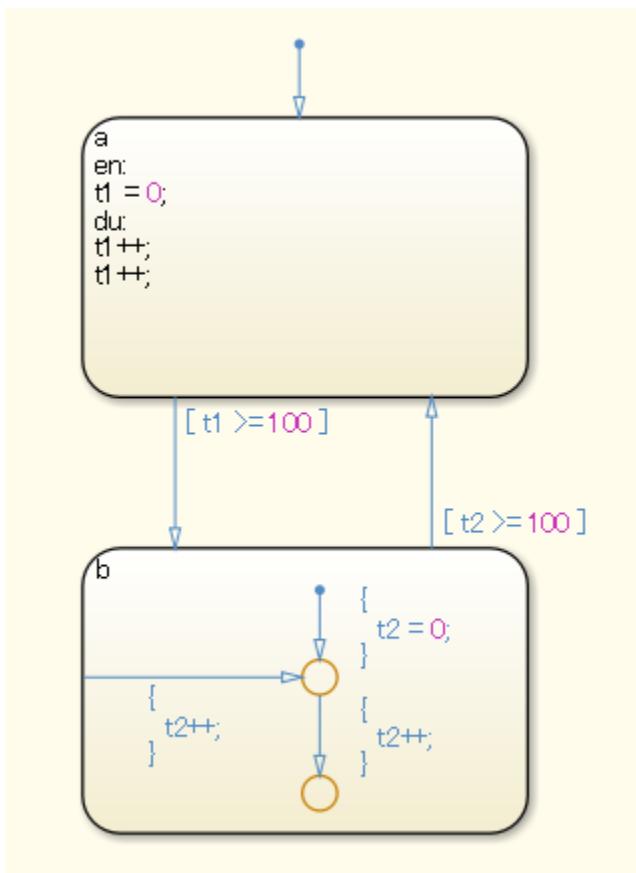
A state shall not include state actions (en, du, or ex) and flow charts.

Custom Parameter

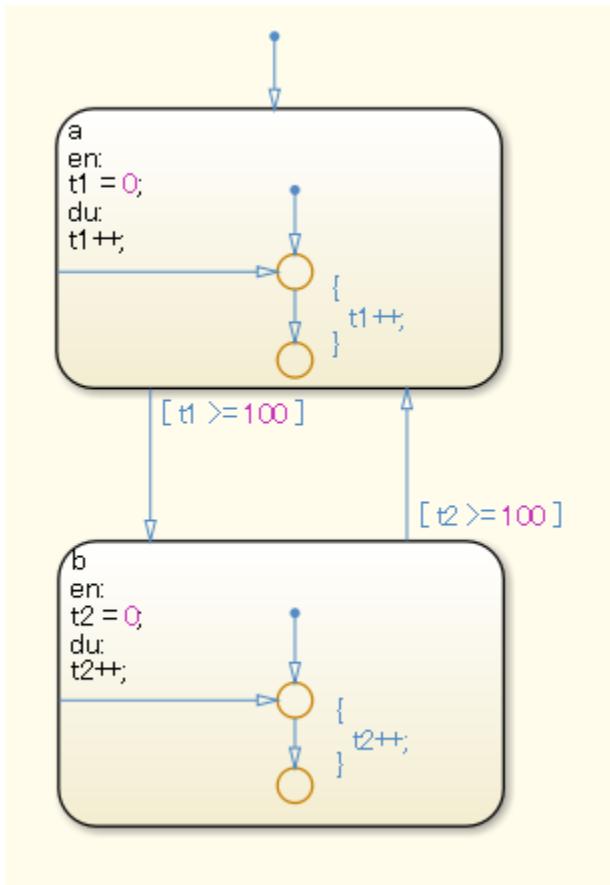
Not Applicable

Example — Correct

Within the state, only a state action is described.

**Example — Incorrect**

The state includes state actions `en` and `du` and also a flow chart.



Rationale

Sub ID a:

- The execution order becomes difficult to understand, which decreases readability.

Verification

Model Advisor check: “Check prohibited combination of state action and flow chart” (Simulink Check)

Last Changed

R2020a

See Also

- “Create Flow Charts in Stateflow” (Stateflow)
- “Represent Operating Modes by Using States” (Stateflow)

Version History

Introduced in R2020a

db_0132: Transitions in flow charts

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

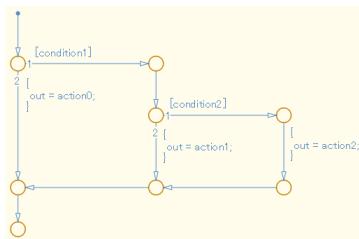
Sub ID a

Transition actions shall not be used in flow charts.

Custom Parameter

Not Applicable

Example — Correct



Sub ID b

In a flow chart, the condition shall be positioned on a horizontal transition line and the condition action shall be positioned on a vertical transition line.

Exception

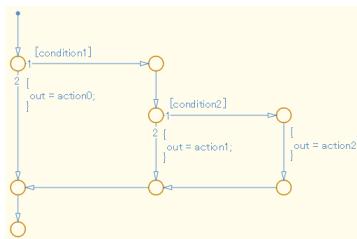
Diagonal transition lines in loop constructs.

Custom Parameter

Not Applicable

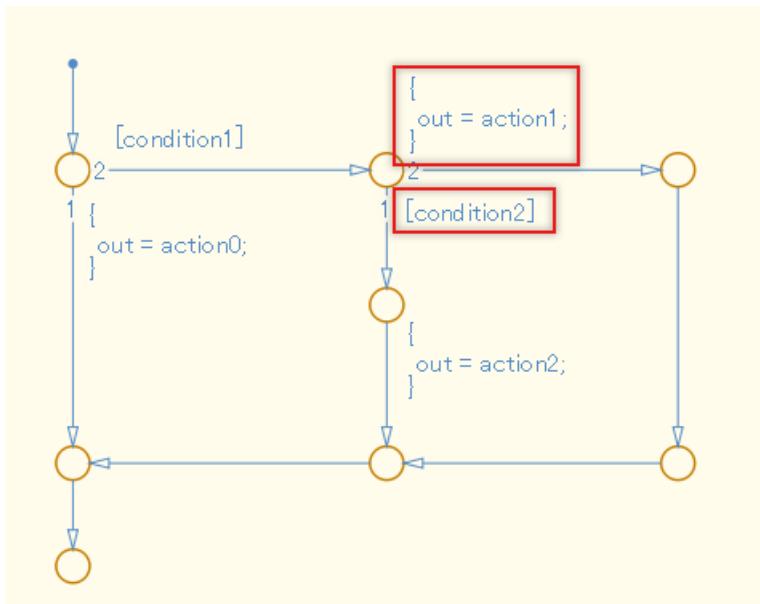
Example – Correct

The condition is positioned on a horizontal transition line and the condition action is on a vertical transition line.



Example – Incorrect

The condition is positioned on a vertical transition line and the condition action is on a horizontal transition line.



Rationale

Sub ID a:

- The transition action in a flow chart is not executed.

Sub ID b:

- Consistent positioning of conditions and condition actions improves readability.

Verification

Model Advisor check: "Check transitions in Stateflow Flow charts" (Simulink Check)

Last Changed

R2020a

See Also

- "Transition Between Operating Modes" (Stateflow)

Version History

Introduced in R2020a

jc_0773: Unconditional transition of a flow chart

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

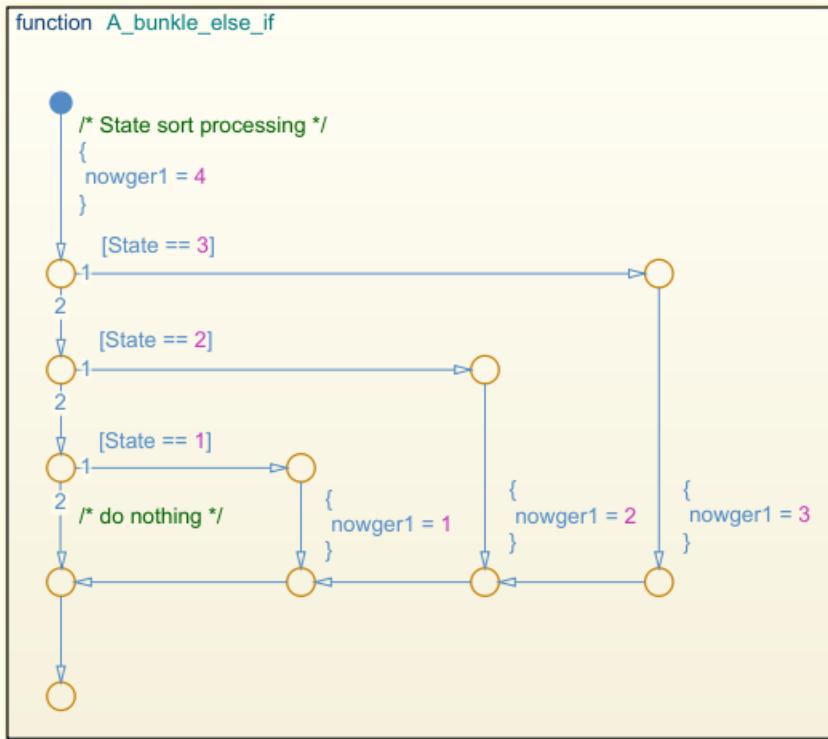
Rule

Sub ID a

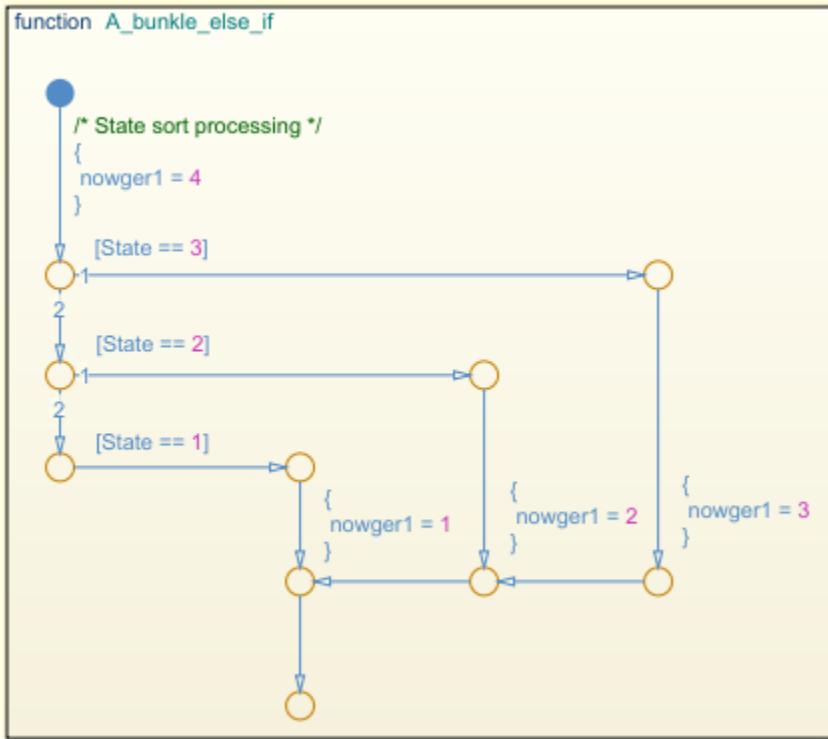
When a transition line with a transition condition originates from a connective junction, the unconditional transition line shall also begin from that junction.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

There is not an unconditional transition line.



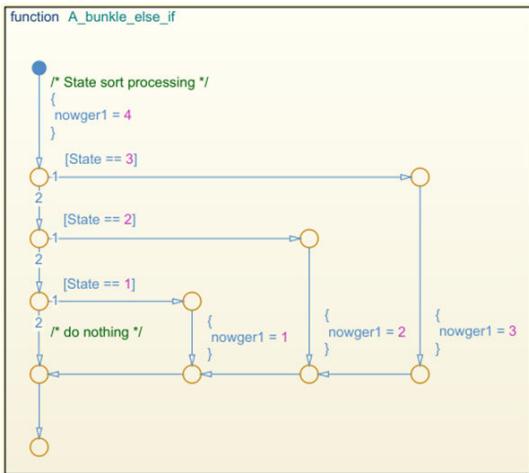
Sub ID b

The **execution order** for unconditional transitions shall be set to the last value.

Custom Parameter

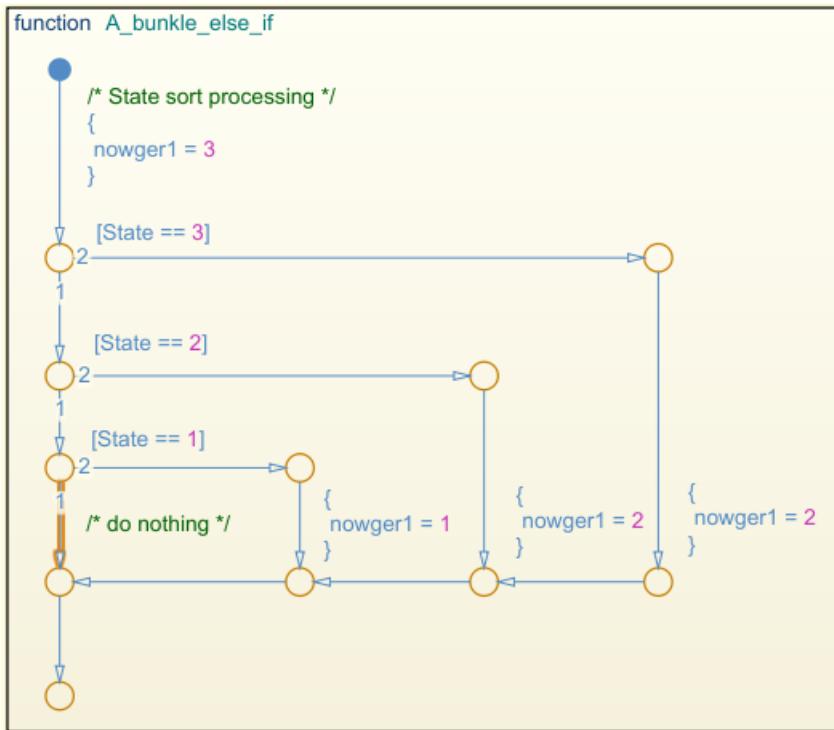
Not Applicable

Example — Correct



Example — Incorrect

The **execution order** for unconditional transitions is not the last value.



Rationale

Sub ID a:

- Prevents unintended behavior that results from backtracking. Setting an unconditional transition explicitly defines the behavior for when the condition is not met.

Sub ID b:

- Setting the unconditional transition to take precedence can prevent unintended behavior.

Verification

Model Advisor check: "Check usage of unconditional transitions in flow charts" (Simulink Check)

Last Changed

R2020a

See Also

- “Transition Between Operating Modes” (Stateflow)
 - “Execution of a Stateflow Chart” (Stateflow)

Version History

Introduced in R2020a

jc_0775: Terminating junctions in flow charts

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1/a2
- JMAAB — a1/a2

MATLAB Versions

All

Rule

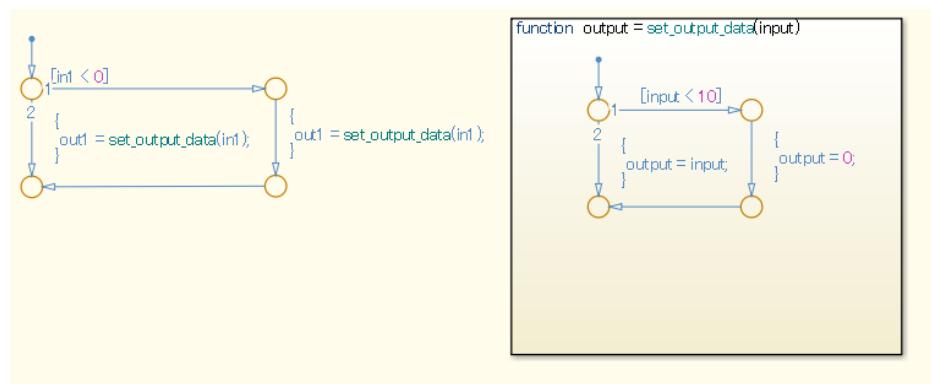
Sub ID a1

Only one terminating junction shall be used.

Custom Parameter

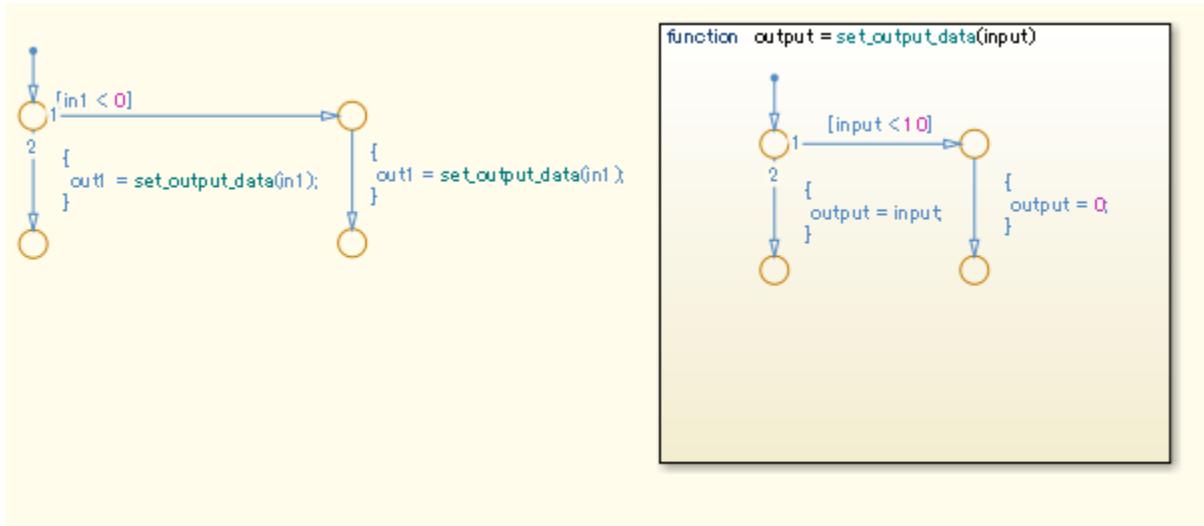
Not Applicable

Example — Correct



Example — Incorrect

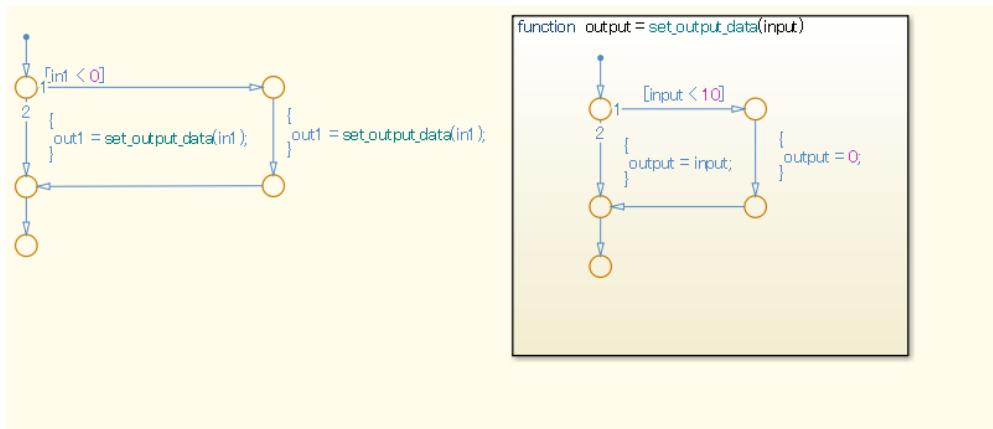
There is more than one terminating junction.

**Sub ID a2**

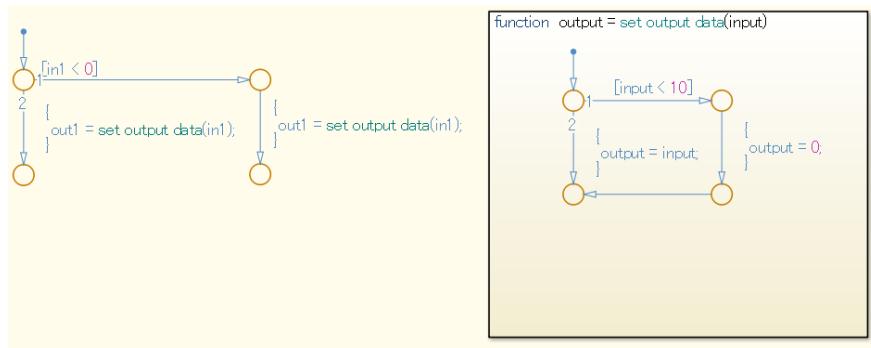
One terminating junction with a single unconditional transition as the input shall be used.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

There is more than one terminating junction and input.



Rationale

Sub IDs a1, a2

- One terminating junction improves understanding of the logic end point.
- Using a consistent style for terminating junction improves readability.

Verification

Model Advisor check: "Check terminal junctions in Stateflow" (Simulink Check)

Last Changed

R2020a

See Also

- “Evaluate Transitions” (Stateflow)
- “Represent Multiple Paths by Using Connective Junctions” (Stateflow)

Version History

Introduced in R2020a

jc_0738: Usage of Stateflow comments

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a, b

MATLAB Versions

All

Rule

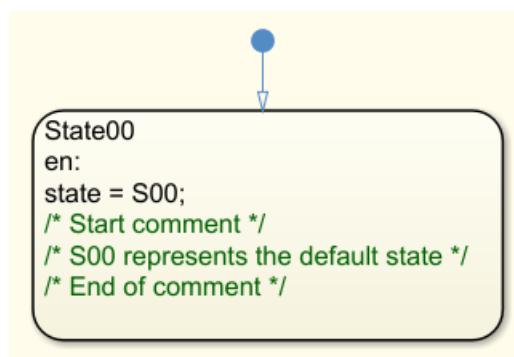
Sub ID a

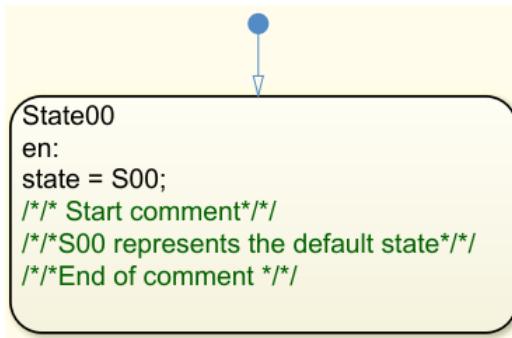
When Chart parameter **Action Language** is set to C, /*...*/ comment nesting shall not be used.

Custom Parameter

Not Applicable

Example — Correct

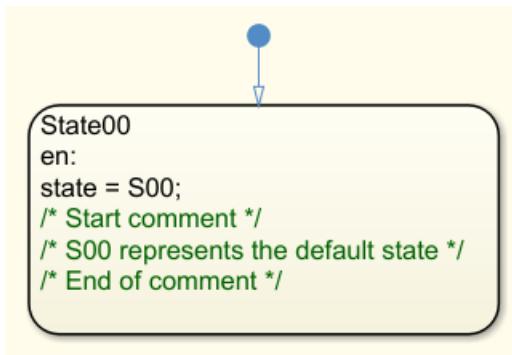
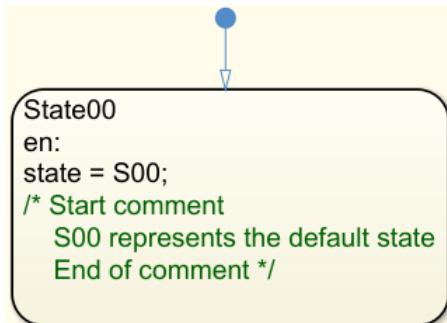


Example — Incorrect**Sub ID b**

When Chart parameter **Action Language** is set to C, new line characters for comments /* */ shall not be used in the middle of a single comment.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Rationale**

Sub ID a:

- The compiler can misinterpret the comments as a program.

Sub ID b:

- A line break in the middle of a comment makes it difficult to determine whether the part being edited is in the comment. There is also a possibility that the comment is nested.
- When Chart parameter **Action Language** is set to MATLAB, comments must use %.

Verification

Model Advisor check: "Check usage of Stateflow comments" (Simulink Check)

Last Changed

R2020a

See Also

- "Differences Between MATLAB and C as Action Language Syntax" (Stateflow)
- "Modify the Action Language for a Chart" (Stateflow)

Version History

Introduced in R2020a

Conditional Transition / Action

jc_0790: Action language of Chart block

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

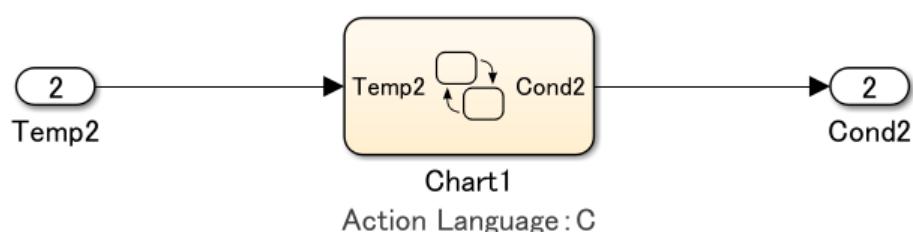
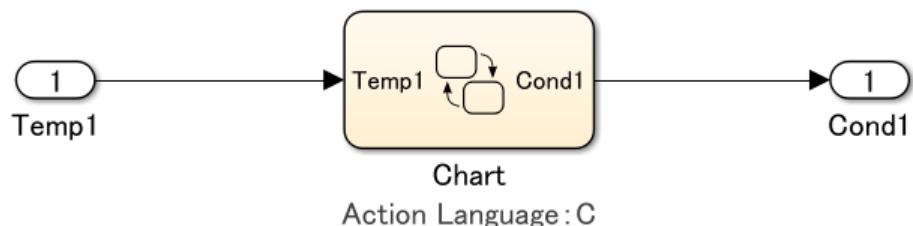
Stateflow Chart property **Action Language** shall be set to C.

Custom Parameter

Not Applicable

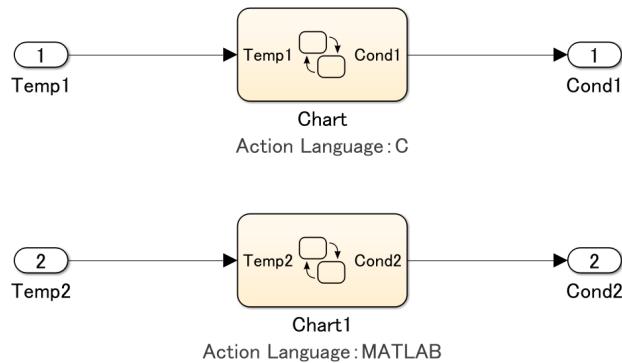
Example — Correct

The **Action Language** is set to C.



Example — Incorrect

The **Action Language** is set to MATLAB.



Rationale

Sub ID a:

- Using a consistent action language improves readability because there is not a difference in syntax.
- Easier to maintain consistency between the model and the generated code when using C as the action language as compared to MATLAB.
- Easier to understand the model for users who are familiar with the C programming language.

Verification

Model Advisor check: “Check Stateflow chart action language” (Simulink Check)

Last Changed

R2020a

See Also

- “Modify the Action Language for a Chart” (Stateflow)
- “Differences Between MATLAB and C as Action Language Syntax” (Stateflow)

Version History

Introduced in R2020a

jc_0702: Use of named Stateflow parameters and constants

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Stateflow block shall not use numeric literal.

Exceptions

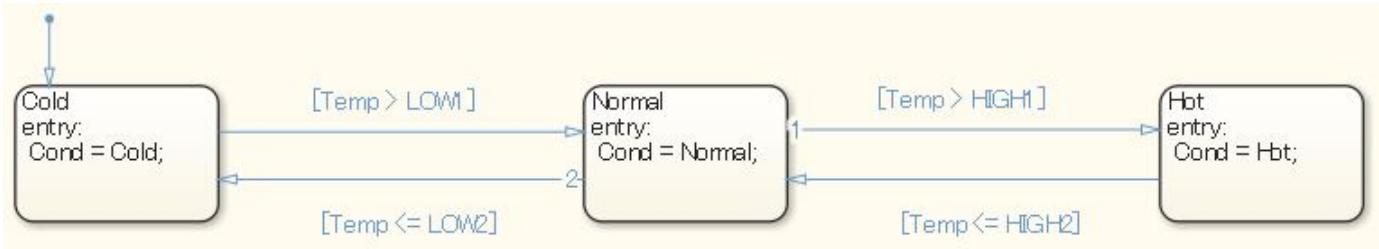
- Initial value is 0
- Increment, decrement 1

Custom Parameter

Not Applicable

Example — Correct

Numeric literals are not used.



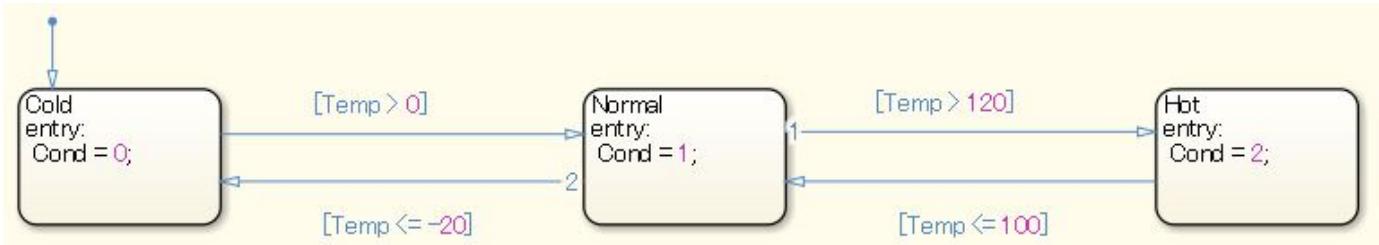
jc0702a_OK_parameter.m

```

1 % 
2 - HIGH1=100;
3 - HIGH2=120;
4 - LOW1=0;
5 - LOW2=10;
  
```

Example — Incorrect

Numeric literals are used.



Rationale

Sub ID a:

- Only the modeler will understand the purpose of the value when numeric literals are used to write constants, which decreases readability.
- Constants that are intended for calibration are generated in the code using numeric literals.

Verification

Model Advisor check: "Check usage of numeric literals in Stateflow" (Simulink Check)

Last Changed

R2020a

See Also

- "Guidelines for Naming Stateflow Objects" (Stateflow)

Version History

Introduced in R2020a

jm_0011: Pointers in Stateflow

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

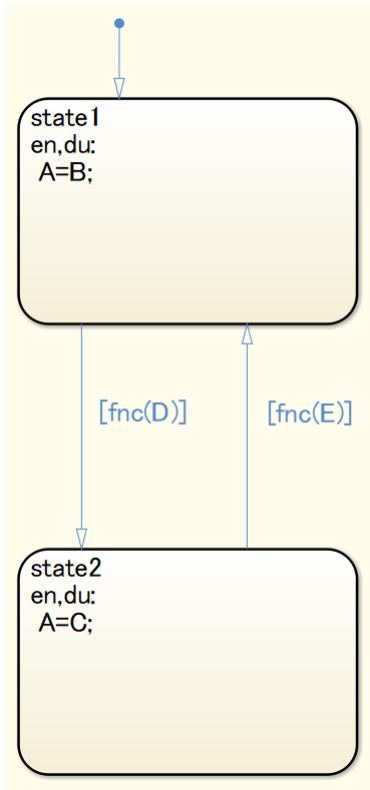
Rule

Sub ID a

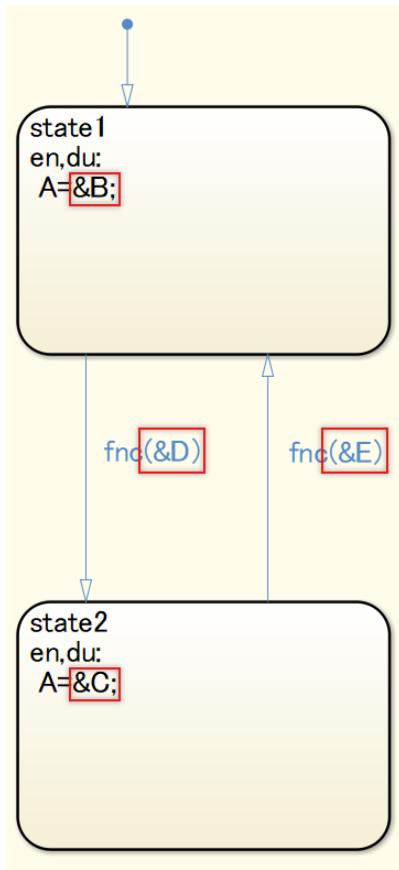
Stateflow Chart shall not use pointer variables.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Pointer variables are used.



Rationale

Sub ID a:

- Readability is impaired when pointer variables are used.
- Code generation may not be possible.

Verification

Model Advisor check: "Check for pointers in Stateflow charts" (Simulink Check)

Last Changed

R2020a

See Also

- "Pointer and Address Operations" (Stateflow)

Version History

Introduced in R2020a

jc_0491: Reuse of Stateflow data

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

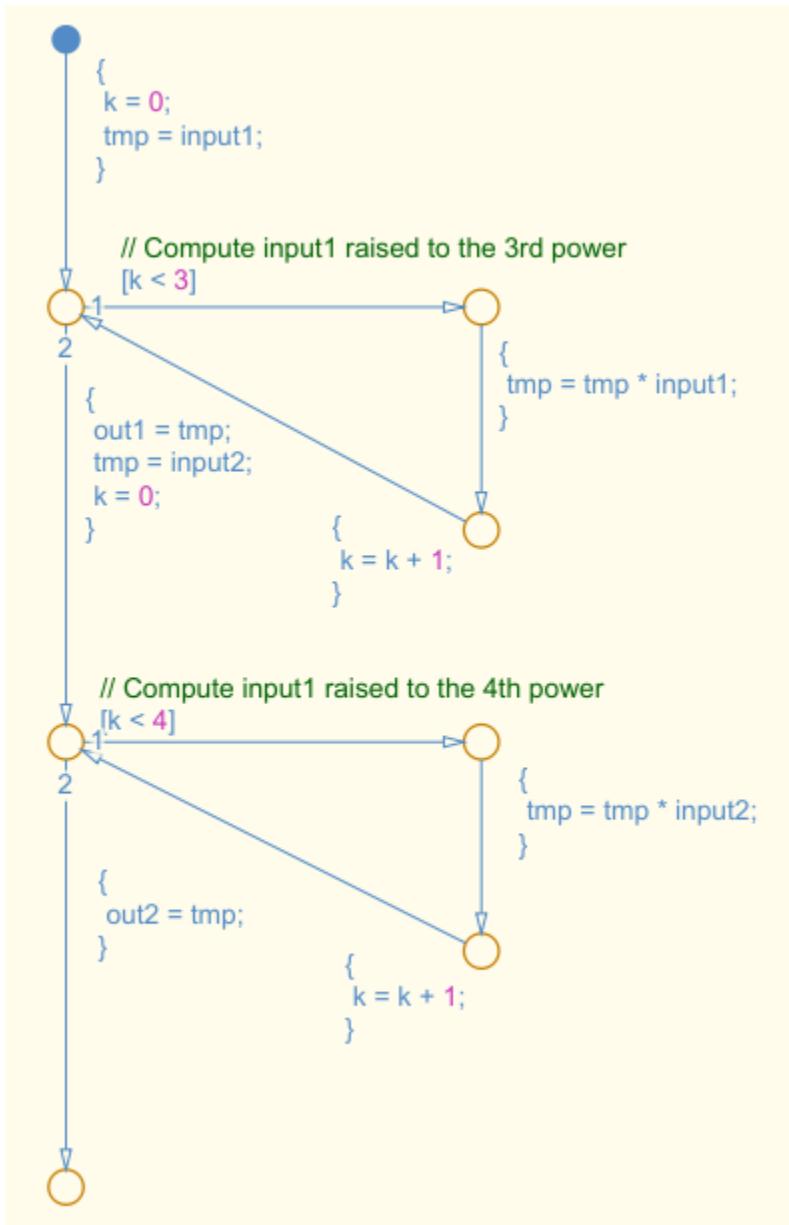
A variable shall not have multiple meanings (usages) in a single Stateflow Chart.

Custom Parameter

Not Applicable

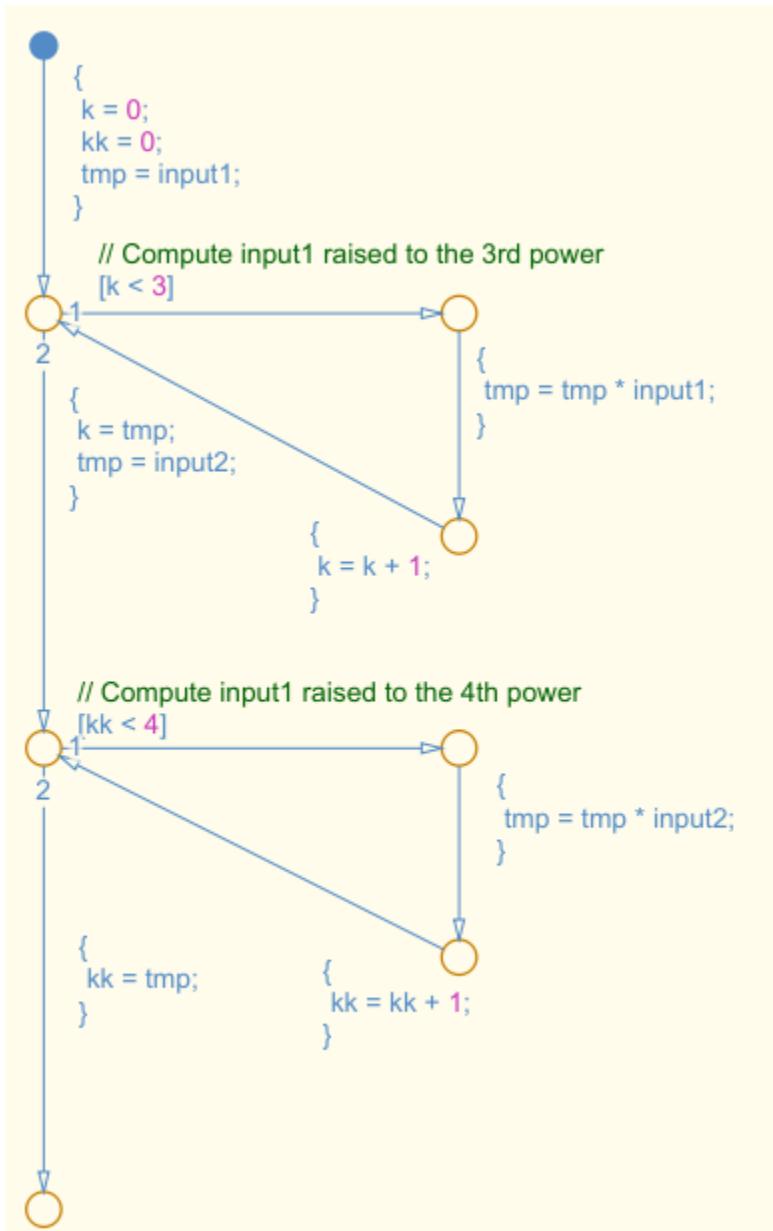
Example — Correct

Variable does not have multiple usages.



Example — Incorrect

Variables `k` and `kk` have multiple usages in the Stateflow Chart.



Rationale

Sub ID a:

- Variables can be misinterpreted when the variable name is different than the meaning of the numerical value that is assigned to the variable.

Verification

Model Advisor check: Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

See Also

- “Construct and Run a Stateflow Chart” (Stateflow)

Version History

Introduced in R2020a

jm_0012: Usage restrictions of events and broadcasting events

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b1/b2

MATLAB Versions

All

Rule

Sub ID a

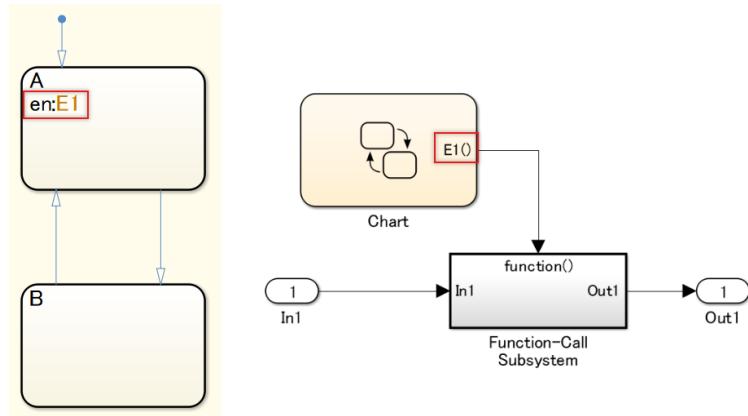
Stateflow events shall be used only in Stateflow Chart output.

Custom Parameter

Not Applicable

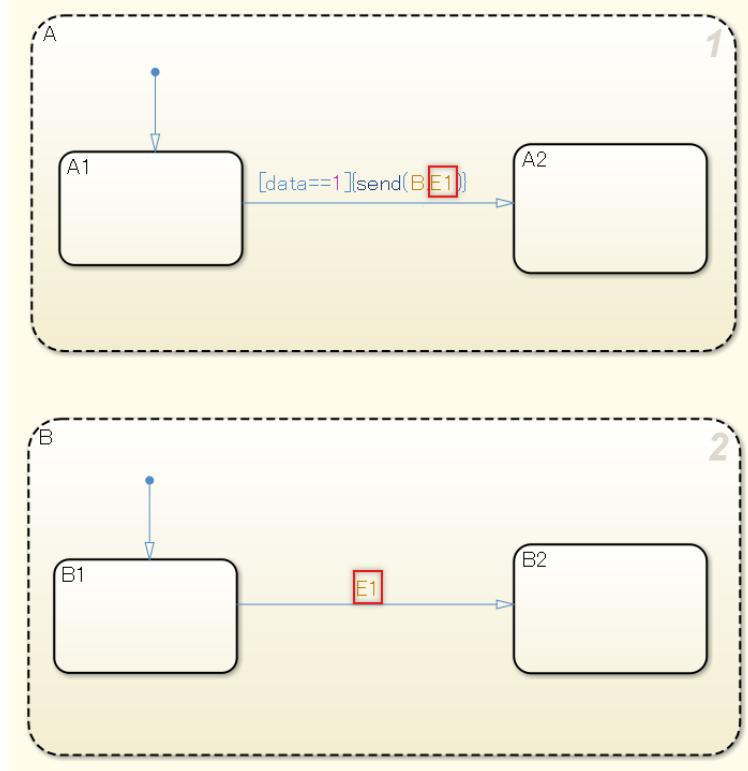
Example — Correct

Event is used only in the Stateflow Chart output.



Example — Incorrect

Event is used other than in the Stateflow Chart output.

**Sub ID b1**

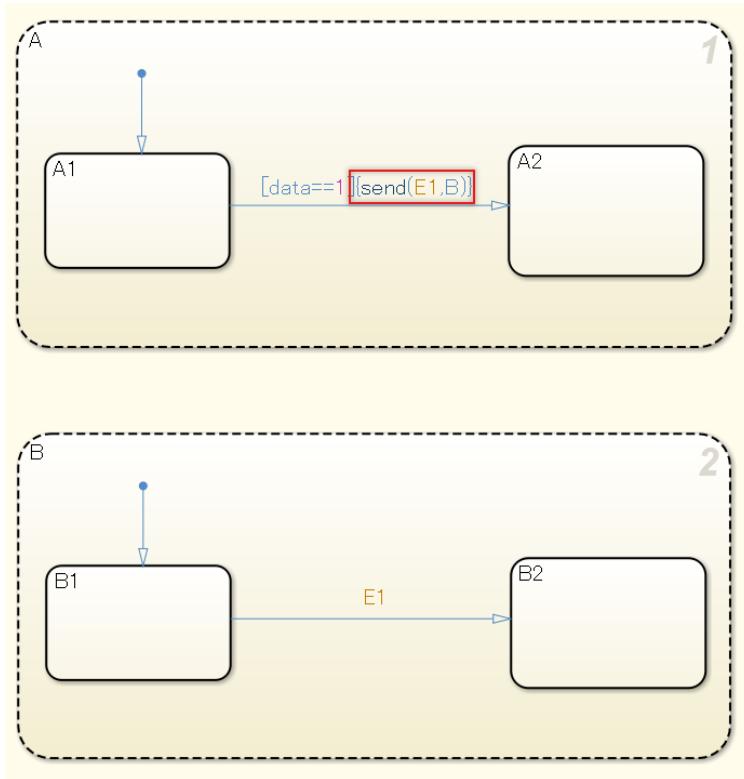
The send syntax `send(event_name, state_name)` shall be used to broadcast Stateflow events.

Custom Parameter

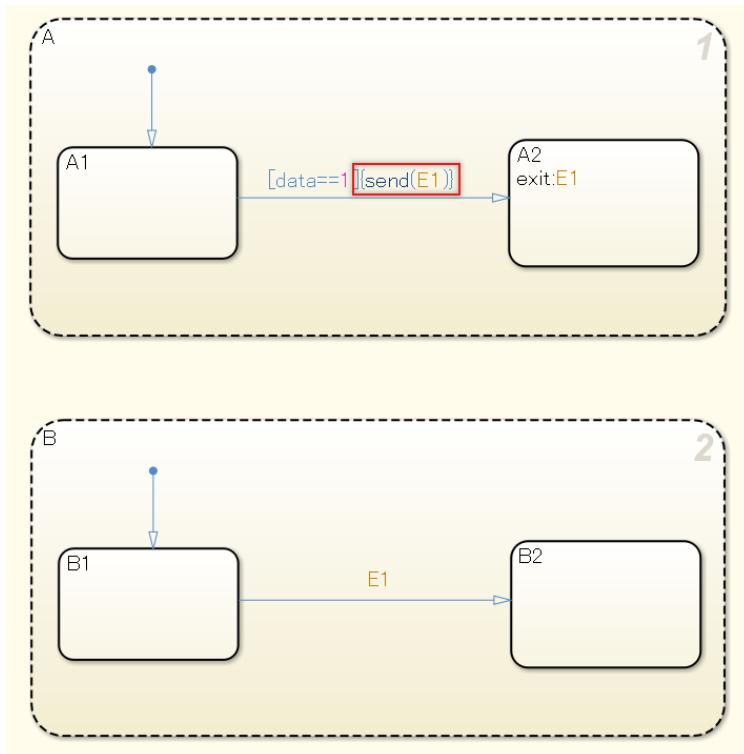
Not Applicable

Example — Correct

Event is broadcast using the `send` syntax.

**Example — Incorrect**

The state that receives the broadcast has not been defined in the `send` syntax.



Sub ID b2

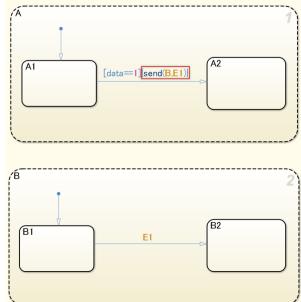
The send syntax `send(state_name.event_name)` with the qualified event name shall be used to broadcast Stateflow events.

Custom Parameter

Not Applicable

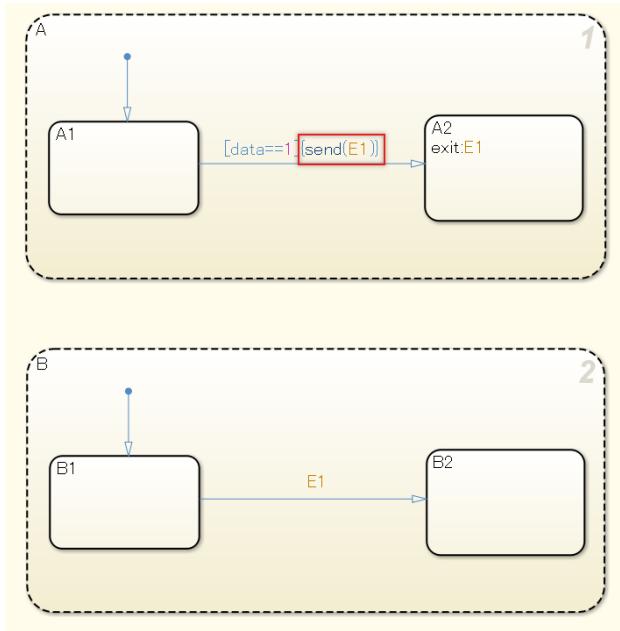
Example — Correct

The qualified event name is used in the event being broadcast.



Example — Incorrect

The state that receives the broadcast has not been described in the send syntax.



Rationale

Sub ID a:

- Recursive processing in a chart is prevented by using Stateflow events in the Stateflow Chart output only.

Sub IDs b1, b2:

- Improves readability because transitions that are triggered by events are clearly identified.

Verification

Model Advisor check: "Check for usage of events in Stateflow charts" (Simulink Check)

Last Changed

R2024b

See Also

- “Events” (Stateflow)
- “Use Events to Execute Charts” (Stateflow)
- “Broadcast Local Events to Synchronize Parallel States” (Stateflow)

Version History

Introduced in R2020a

jc_0733: Order of state action types

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

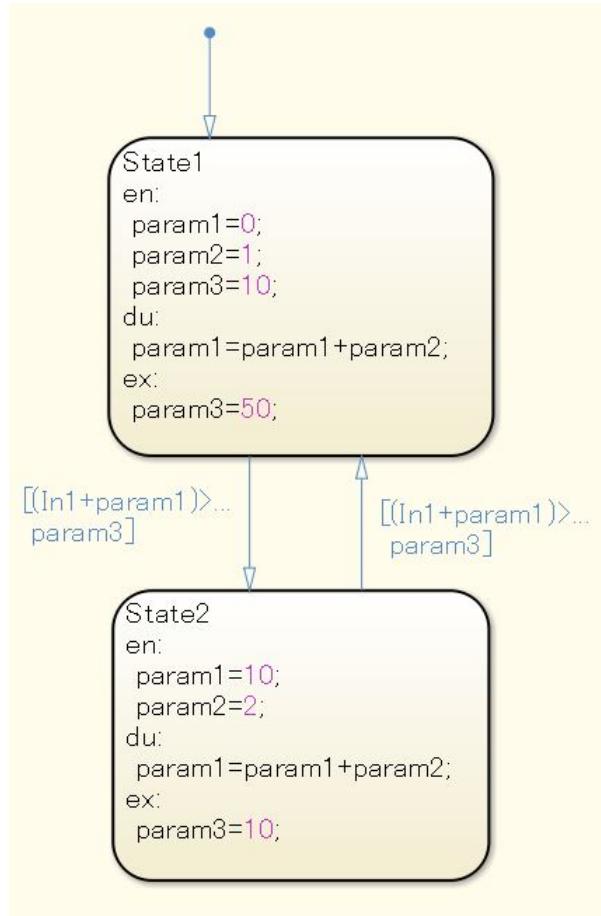
Basic state action types shall be stated in this order:

entry (en)
during (du)
exit (ex)

Custom Parameter

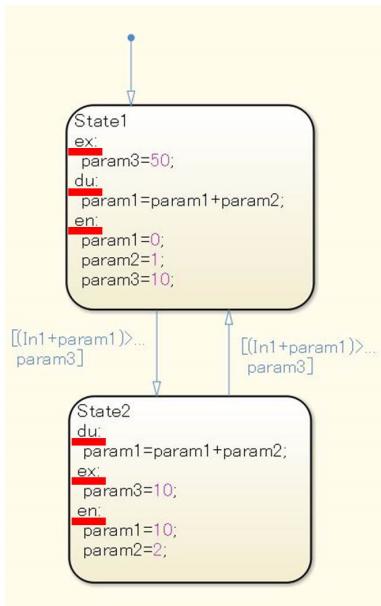
Not Applicable

Example — Correct



Example — Incorrect

Action types are out of order.



Sub ID b

Combined state action types shall be stated in this order:

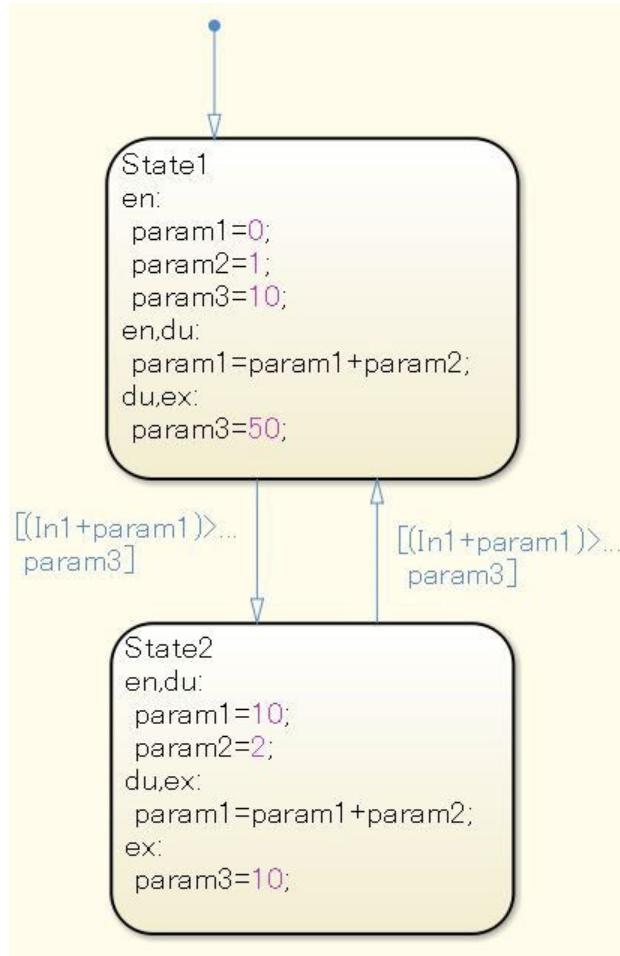
entry (en)

during (du)

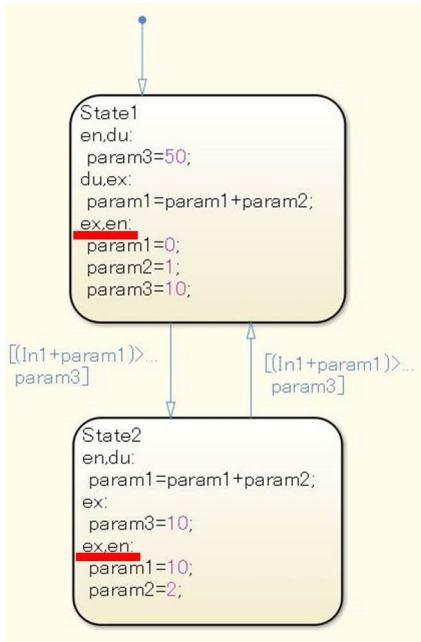
exit (ex)

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Action types are out of order



Rationale

Sub ID a:

- Consistent modeling improves readability and maintainability.

Verification

Model Advisor check: "Check order of state action types" (Simulink Check)

Last Changed

R2020a

See Also

- “Represent Operating Modes by Using States” (Stateflow)
- “During Actions” (Stateflow)
- “Exit a State” (Stateflow)

Version History

Introduced in R2020a

jc_0734: Number of state action types

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

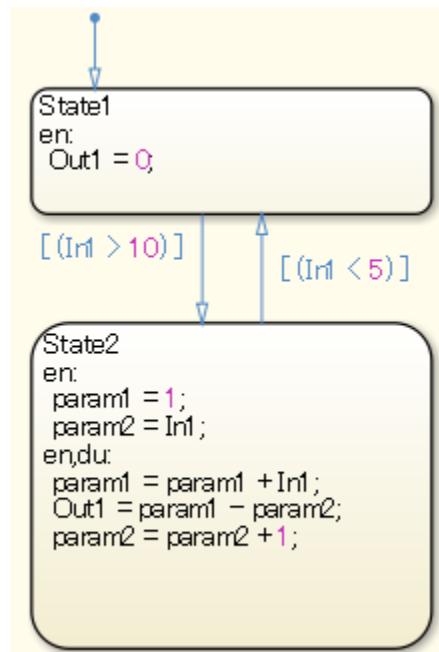
Sub ID a

State action types shall not be used more than one time in a Stateflow state.

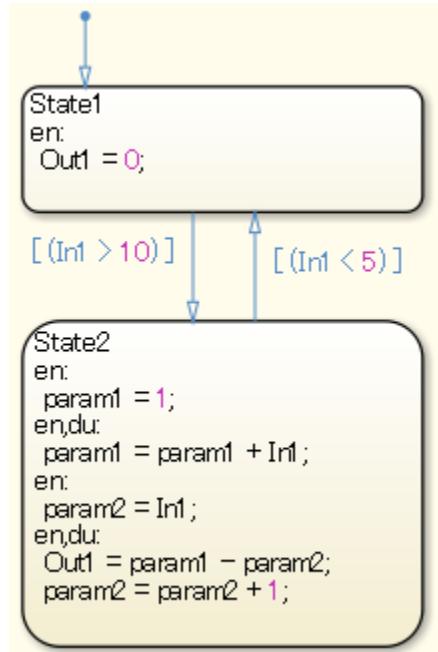
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- The execution order will differ depending on the order in which they are described.
- Execution order can be difficult to understand when the action type is described multiple times.

Verification

Model Advisor check: "Check repetition of action types" (Simulink Check)

Last Changed

R2020a

See Also

- “Represent Operating Modes by Using States” (Stateflow)

Version History

Introduced in R2020a

jc_0740: Limitation on use of exit state action

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

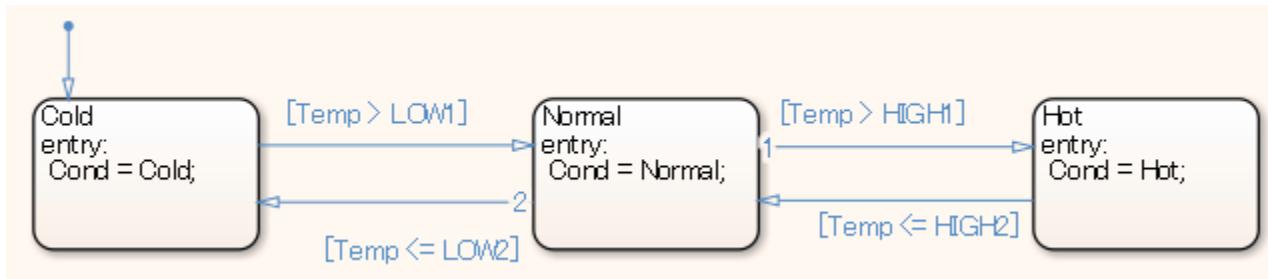
Sub ID a

State action type `exit(ex)` shall not be used.

Custom Parameter

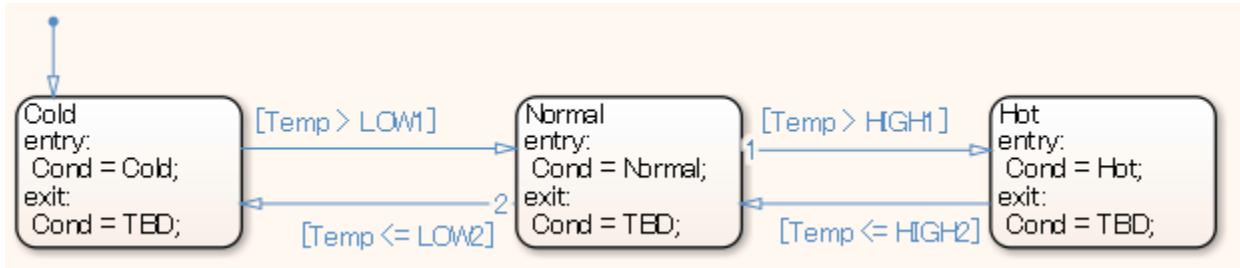
Not Applicable

Example — Correct



Example — Incorrect

This example illustrates how the model behavior in Stateflow Chart is misinterpreted. It appears that TBD is output when state action type `exit(ex)` is used, but it is in fact being overwritten by the state action type `entry` of the transition destination state. It is not outputted by the Stateflow Chart.



Rationale

Sub ID a:

- Execution timing can be difficult to understand when state action type `exit(ex)` is used in combination with a conditional action, a transition action, or state action type `entry(en)`. This can result in misinterpretation of the model behavior.

Verification

Model Advisor check: "Check if state action type 'exit' is used in the model" (Simulink Check)

Last Changed

R2020a

See Also

- "Exit a State" (Stateflow)

Version History

Introduced in R2020a

jc_0741: Timing to update data used in state chart transition conditions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a1, a2, b

MATLAB Versions

All

Rule

Sub ID a1

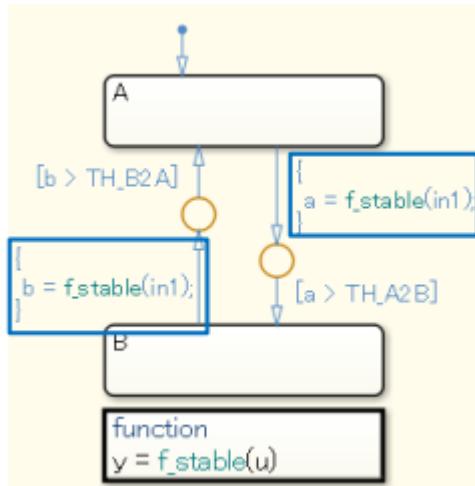
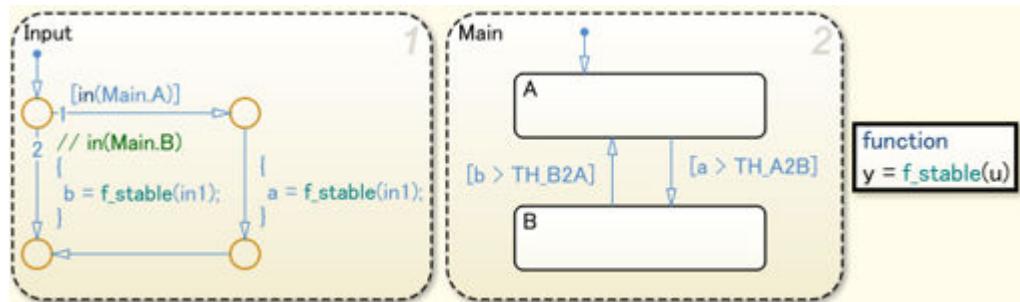
Variables that are used in a state transition condition shall not use state action du to perform an update.

Custom Parameter

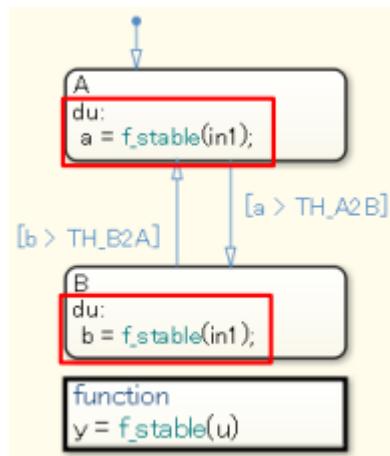
Not Applicable

Exceptions

This rule applies exclusively to writing in a state chart with a transition source. The update is not prohibited for parallel states and outside of the chart.

Example — Correct**Example — Correct (for exception)****Example — Incorrect**

State action du is used to perform an update.

**Sub ID a2**

Variables that are used in a state transition condition shall use state action du to perform an update.

Custom Parameter

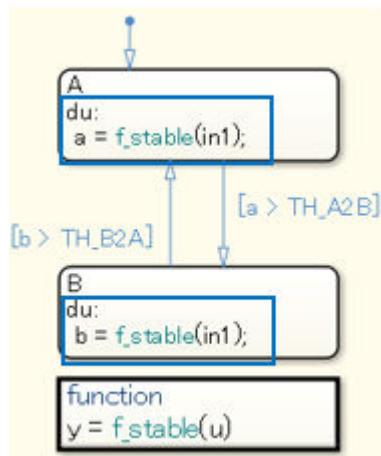
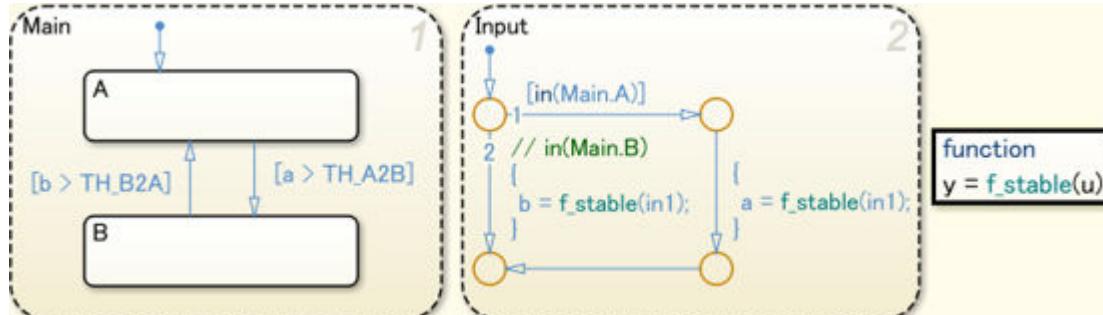
Not Applicable

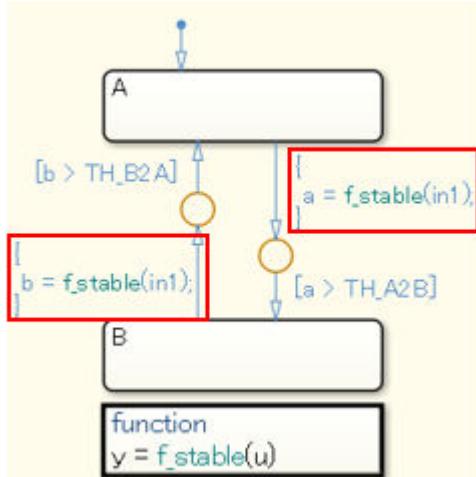
Exceptions

This rule applies exclusively to writing in a state chart with a transition source. The update is not prohibited for parallel states and outside of the chart.

Example — Correct

State action du is used to perform an update.

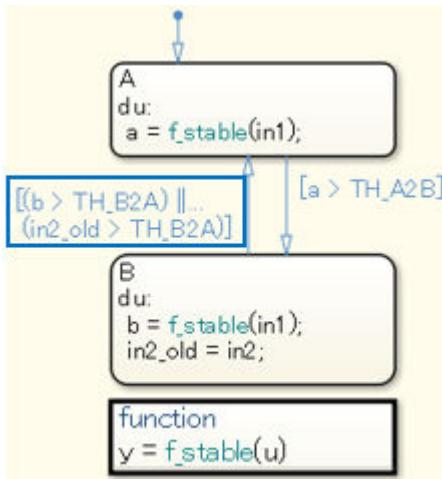
**Example — Correct (for exception)**

Example — Incorrect**Sub ID b**

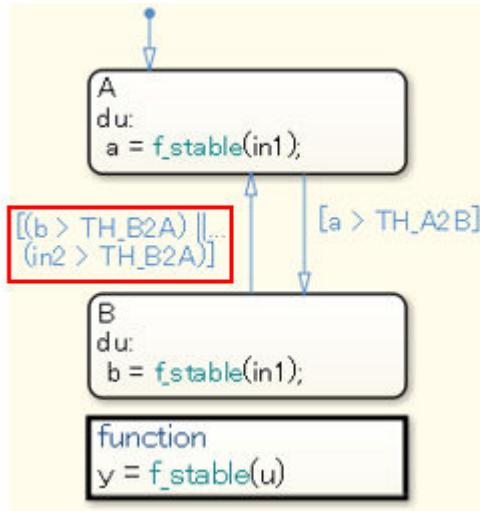
When the data referenced in a transition condition is updated to du as its source state, mixing updated and non-updated data in the same transition condition is prohibited, except for constants and parameters.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

State action du is used to perform an update. There is a mixture of data to update with State action du(b) and data not updated in2.



Rationale

Sub ID a1:

- The execution order of the transition condition and implement of `during` can be difficult to understand, which increases the risk of errors.

Verification

Model Advisor check: "Check updates to variables used in state transition conditions" (Simulink Check)

Last Changed

R2024b

See Also

- "Transition Between Operating Modes" (Stateflow)
- "Represent Operating Modes by Using States" (Stateflow)
- "Execution of a Stateflow Chart" (Stateflow)

Version History

Introduced in R2020a

jc_0772: Execution order and transition conditions of transition lines

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

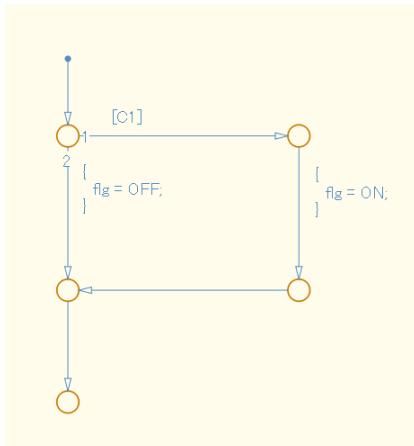
All transition paths shall be executable by setting configuration parameter:

- (R2011b to R2016a) **Set Transition shadowing** to error.
- (R2016b and later) Set **Unreachable execution path** to error.

Custom Parameter

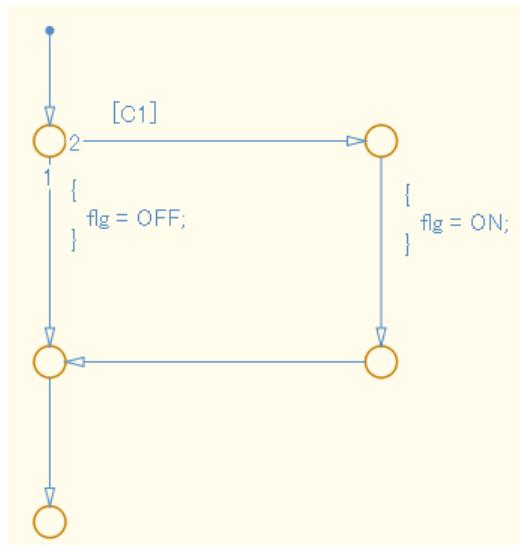
Not Applicable

Example — Correct



Example – Incorrect

Execution order 1 is an unconditional transition and conditional expression [C1] is described in execution condition 2.



Rationale

Sub ID a:

- An unconditional transition that is in a position other than the last in the execution order causes the subsequent transition to be a dead path, which results in unintended simulation behavior.

Verification

Model Advisor check: "Check usage of transition conditions in Stateflow transitions" (Simulink Check)

Last Changed

R2020a

See Also

- “Transition Between Operating Modes” (Stateflow)
 - “Types of Chart Execution” (Stateflow)
 - “Execution of a Stateflow Chart” (Stateflow)

Version History

Introduced in R2020a

jc_0753: Condition actions and transition actions in Stateflow

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

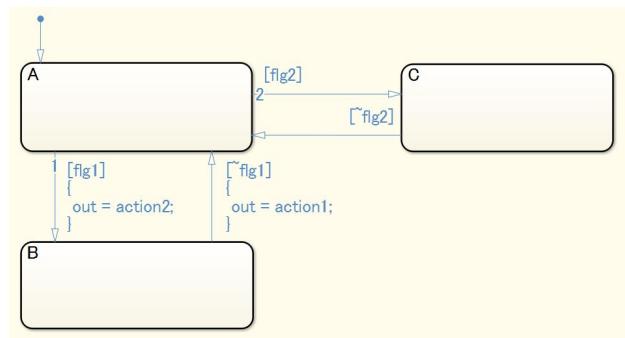
Transition actions shall not be used in a Stateflow Chart.

Custom Parameter

Not Applicable

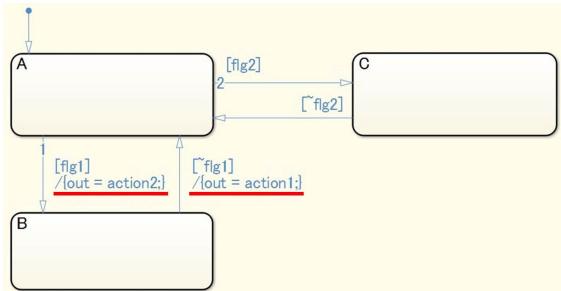
Example — Correct

Only a condition action is used.



Example — Incorrect

A transition action is used.



Sub ID b

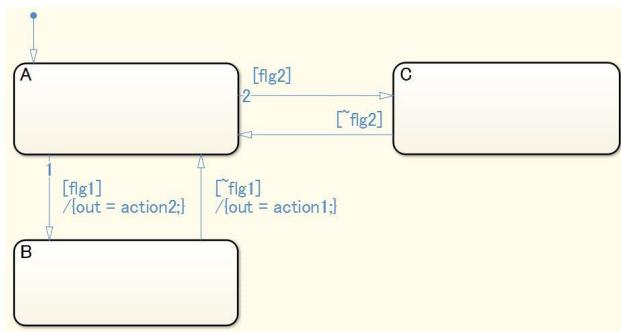
Condition actions and transition actions shall not be combined in the same Stateflow Chart.

Custom Parameter

Not Applicable

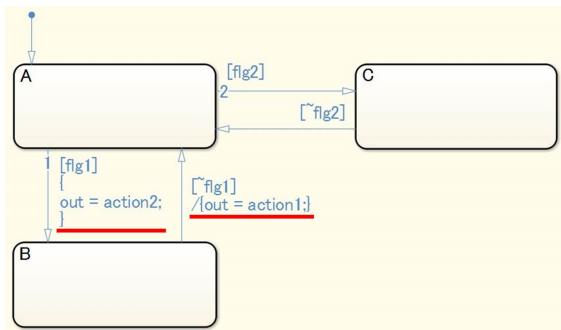
Example — Correct

Either a condition action or a transition action can be used. (The following diagram illustrates a transition action.)



Example — Incorrect

Includes both a condition action and a transition action.



Rationale

Sub ID a:

- Prevents confusion with a condition action, thus improving readability.

Sub ID b:

- A condition action executes upon entering a transition. A transition action executes after determining whether it can transition to the next state. Adherence to the rule prevents confusion between a conditional action and a transition action.

Verification

Model Advisor check: "Check condition actions and transition actions in Stateflow" (Simulink Check)

Last Changed

R2024b

See Also

- "Evaluate Outer Transitions with Condition and Transition Actions" (Stateflow)

Version History

Introduced in R2020a

jc_0711: Division in Stateflow

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1/a2
- JMAAB — a1/a2

MATLAB Versions

All

Rule

Sub ID a1

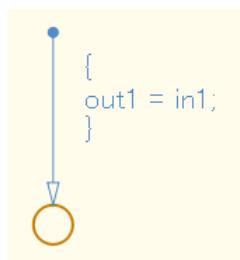
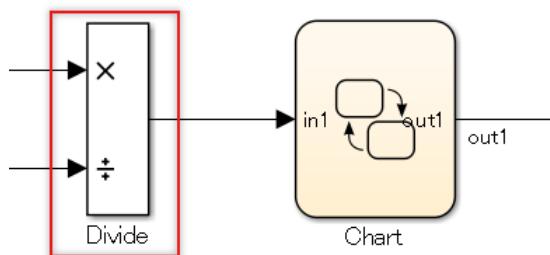
Variables, constants, or parameters in Stateflow Chart shall not be used to perform division operations.

Custom Parameter

Not Applicable

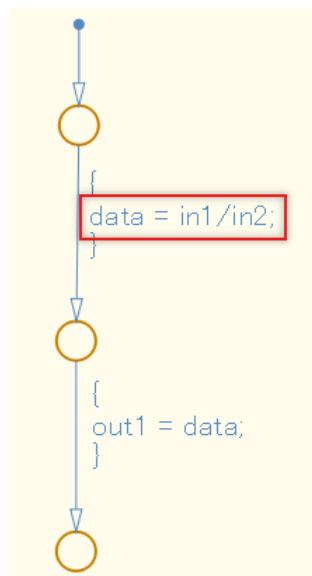
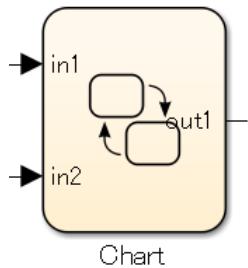
Example — Correct

Division is performed outside of the Stateflow Chart.



Example — Incorrect

Division occurs within the Stateflow Chart.

**Sub ID a2**

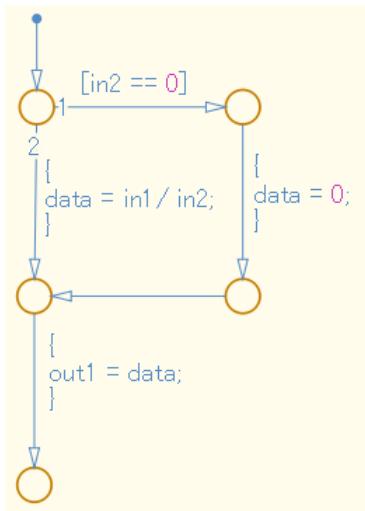
When division occurs in a Stateflow Chart, the process shall prevent division by zero.

Custom Parameter

Not Applicable

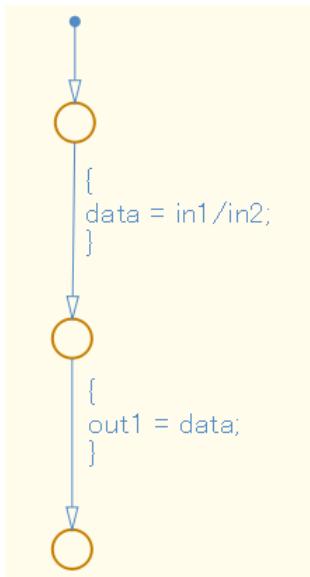
Example — Correct

The process is defined to prevent division by zero.



Example – Incorrect

The process does not prevent division by zero.



Rationale

Sub ID a1, a2:

- Deviation from the rule can cause unintended operation and code generation results.

Verification

Model Advisor check: Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

See Also

- “Fixed-Point Operations in Stateflow Charts” (Stateflow)

Version History

Introduced in R2020a

db_0127: Limitation on MATLAB commands in Stateflow blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a, b
- JMAAB — a, b

MATLAB Versions

All

Rule

Sub ID a

Built-in MATLAB functions shall not be used in Stateflow blocks.

Supplement

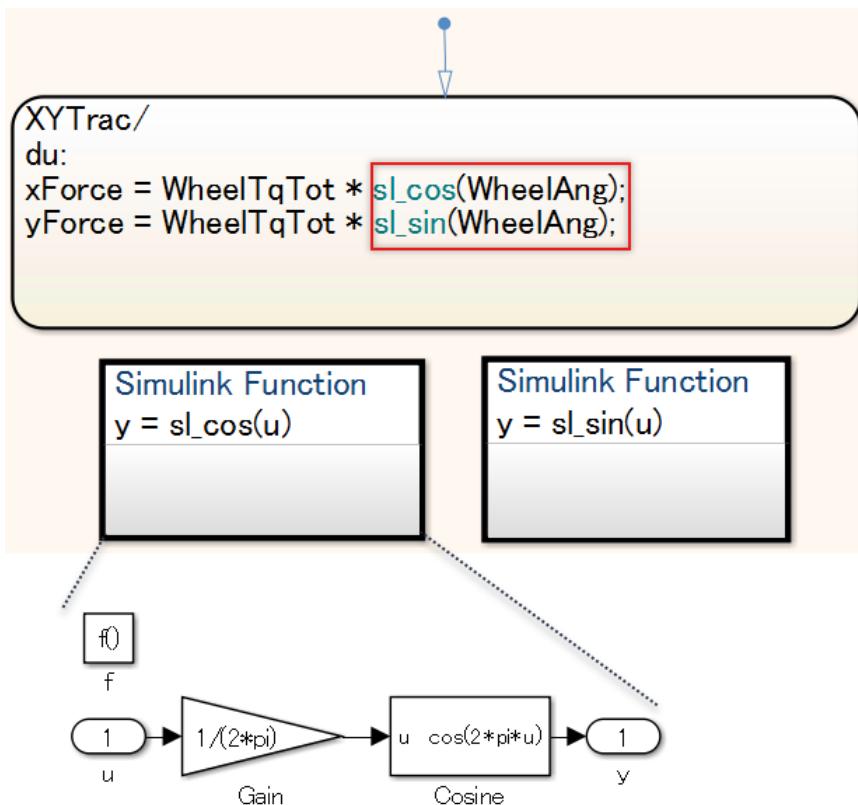
Applicable for C charts

Custom Parameter

Not Applicable

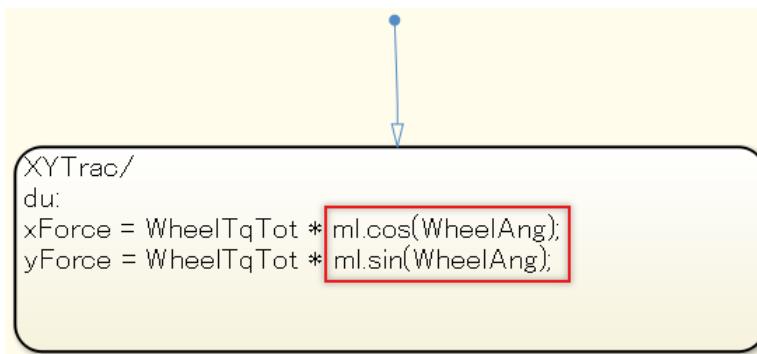
Example — Correct

MATLAB commands are not used in Stateflow blocks.



Example — Incorrect

A MATLAB command is used in the Stateflow block.



Sub ID b

When a built-in MATLAB function is used in Stateflow blocks, it shall be accessed only by using MATLAB Function.

Supplement

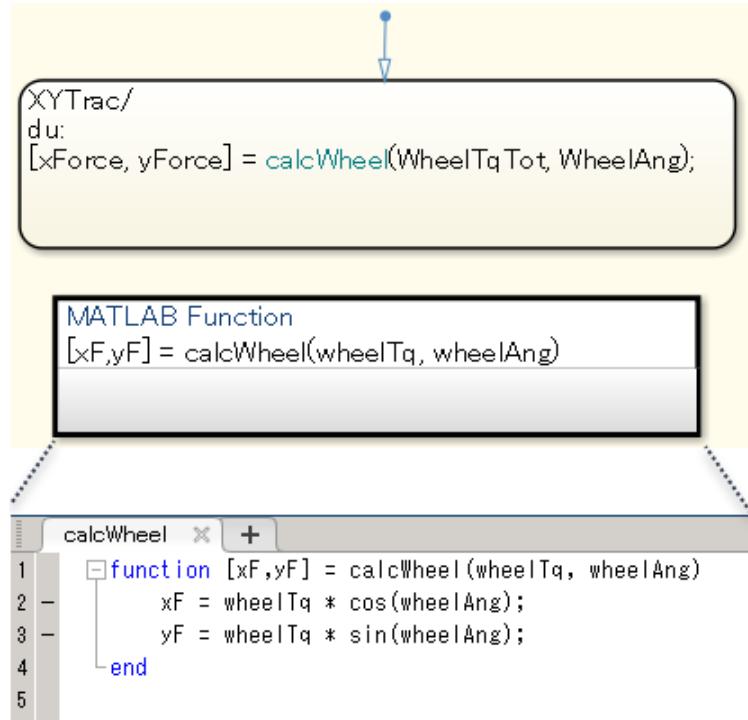
Applicable for C charts

Custom Parameter

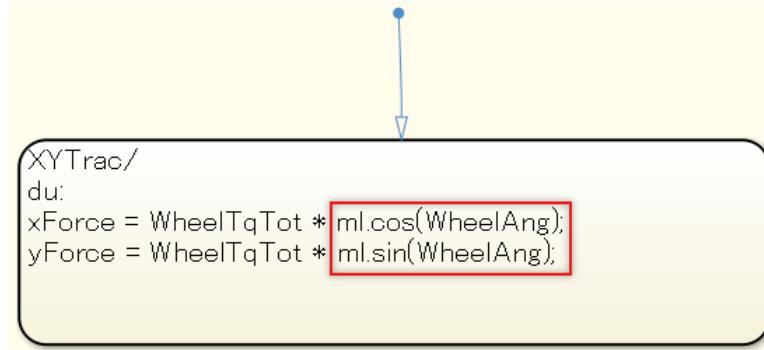
Not Applicable

Example — Correct

The MATLAB command is accessed by using the MATLAB Function block.

**Example — Incorrect**

A MATLAB Function block is not used for a MATLAB command.

**Rationale**

Sub ID a:

- Not all built-in MATLAB functions are supported for code generation. As a result, code may not be generated for these unsupported MATLAB functions.

Sub ID b:

- Not all built-in MATLAB functions are supported for code generation. As a result, code may not be generated for these unsupported MATLAB functions.
- Readability improves when C and MATLAB action languages are described separately.

Verification

Model Advisor check: "Check for MATLAB expressions in Stateflow charts" (Simulink Check)

Last Changed

R2024b

See Also

- MATLAB Function
- "Stateflow Programmatic Interface" (Stateflow)
- "Differences Between MATLAB and C as Action Language Syntax" (Stateflow)

Version History

Introduced in R2020a

jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

These equality comparison operators shall not be used in floating-point operands:

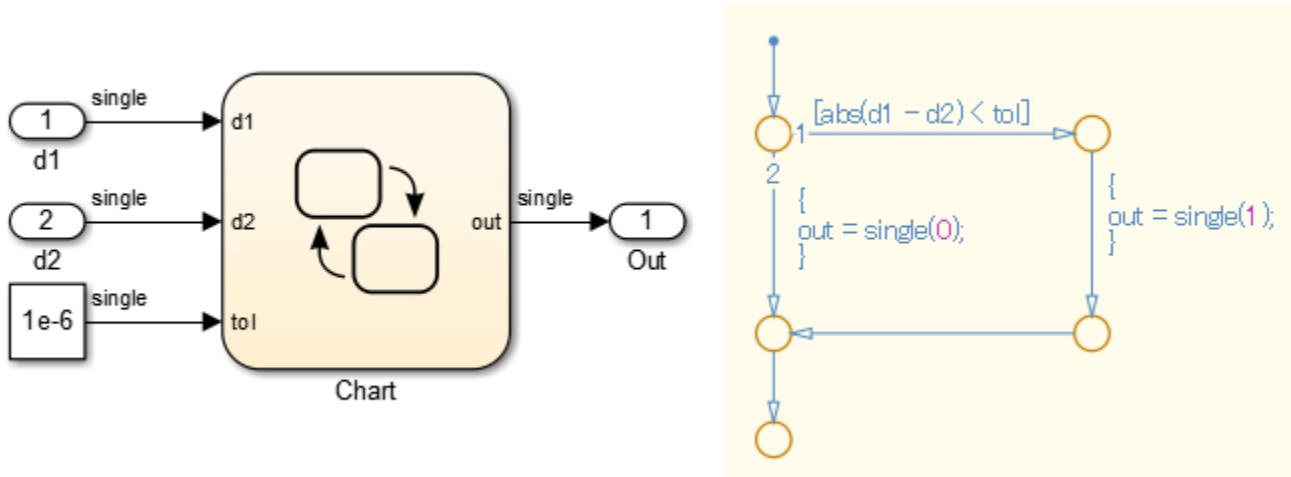
- ==
- !=
- ~=
- <>

Custom Parameter

Not Applicable

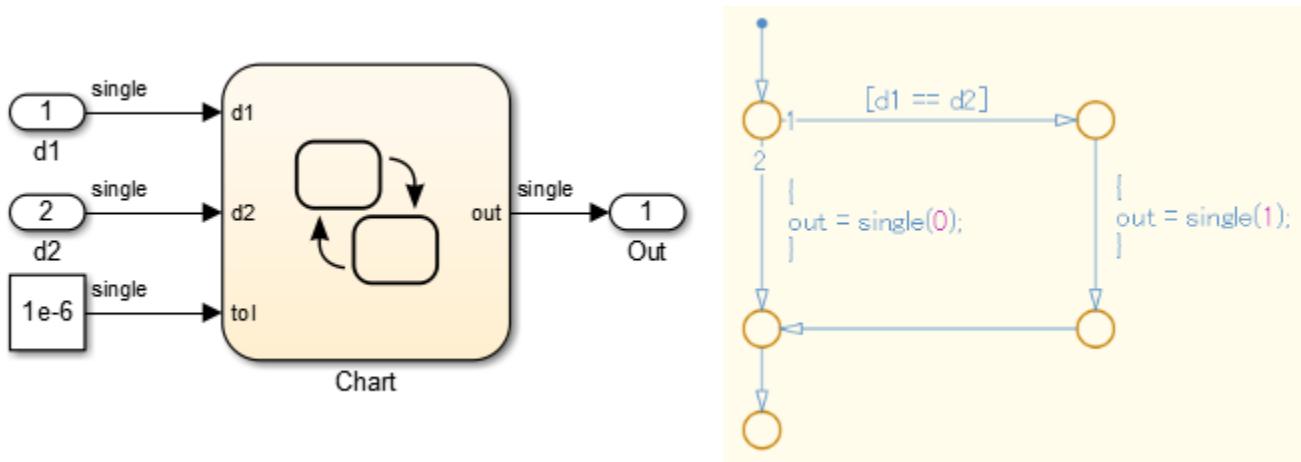
Example — Correct

Equality comparison operators are not used in floating-point operands.



Example — Incorrect

Equality comparison operator `==` is used in floating-point operands.



Rationale

Sub ID a:

- Due to the nature of the floating-point data type, as it contains an error, the result of the equivalence comparison operation may be false when it was expected to be true.

Verification

Model Advisor check: "Check usage of floating-point expressions in Stateflow charts" (Simulink Check)

Last Changed

R2024b

See Also

- “Fixed-Point Data in Stateflow Charts” (Stateflow)
- “Relational Operations”

Version History

Introduced in R2020a

na_0001: Standard usage of Stateflow operators

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b1/b2/b3, c

MATLAB Versions

All

Rule

Sub ID a

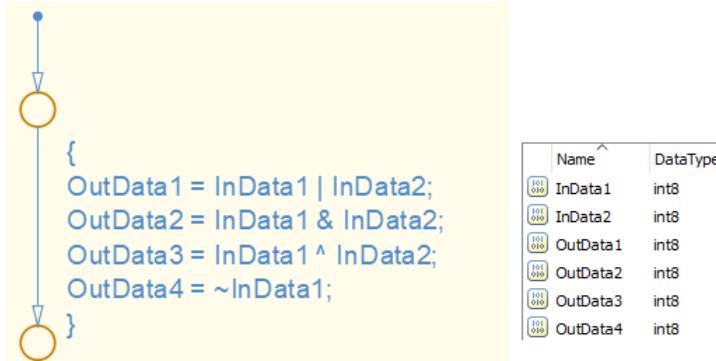
When Stateflow Chart property **Action Language** is set to C, operators ($\&$, $|$, $^$, \sim) shall be used only for bit operations.

Custom Parameter

Not Applicable

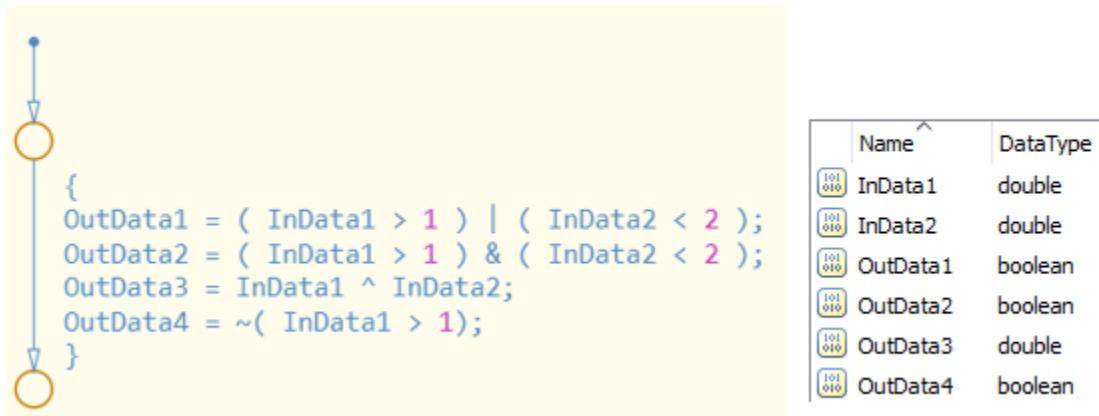
Example — Correct

Operators ($\&$, $|$, $^$, \sim) are used for bit operations.



Example — Incorrect

Operators ($\&$, $|$, $^$, \sim) are not used for bit operations.



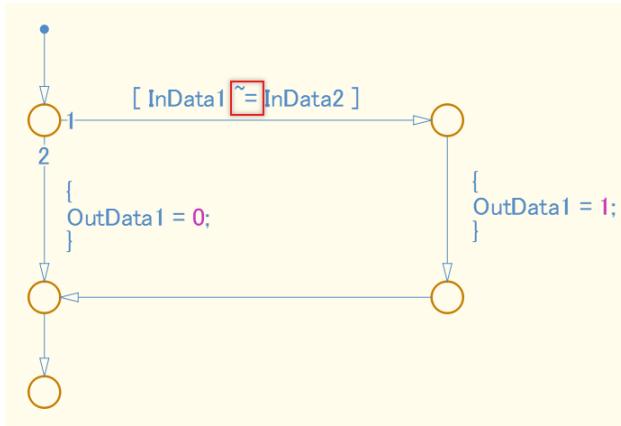
Sub ID b1

When Stateflow Chart property **Action Language** is set to C, operator `~` shall be used for inequality operations.

Custom Parameter

Not Applicable

Example — Correct

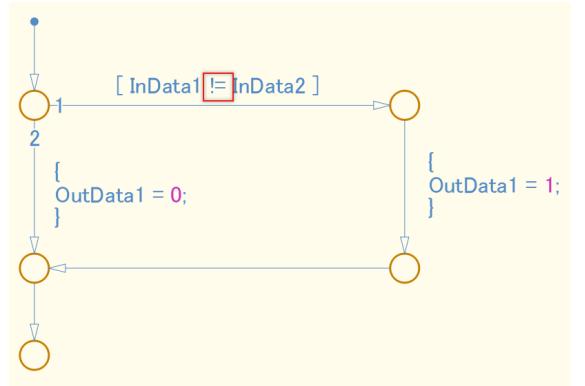


Sub ID b2

When Stateflow Chart property **Action Language** is set to C, operator `!=` shall be used for inequality operations.

Custom Parameter

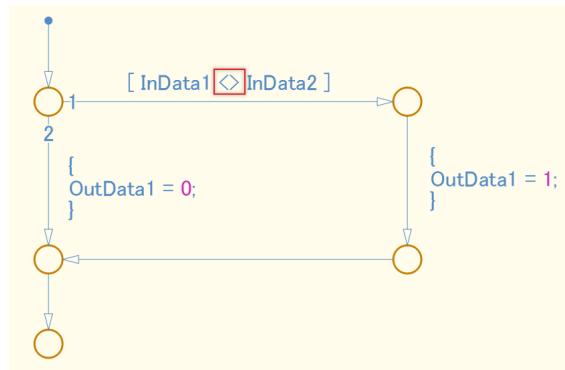
Not Applicable

Example — Correct**Sub ID b3**

When Stateflow Chart property **Action Language** is set to C, operator `<>` shall be used for inequality operations.

Custom Parameter

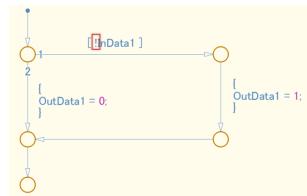
Not Applicable

Example — Correct**Sub ID c**

When Stateflow Chart property **Action Language** is set to C, operation `!` shall be used for logical negation.

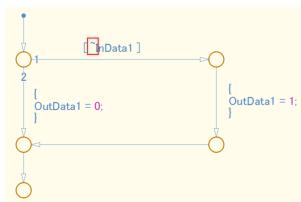
Custom Parameter

Not Applicable

Example — Correct

Example — Incorrect

An operator other than ! should be used for logical negation.



Rationale

Sub ID a:

- When either of these Stateflow Chart properties are set as follows:
 - Action Language** is set to MATLAB
 - Action Language** is set to C and **Enable C-bit operations** is selected
- && and &, || ,and | , have the same calculation function. However, when && and & or || and | are combined in the same chart, it can be difficult to determine whether these are separate calculation functions or the same calculation function.

Sub IDs b1, b2, b3:

- Consistent use of equality operators improves readability.

Sub ID c:

- Consistent use of logical negation operators improves readability.
- When **C-bit operations are enabled** is selected, the function of the ! operator remains the same and is not affected by logic changes that result from changing the setting.

Verification

Model Advisor check: “Check Stateflow operators” (Simulink Check)

Last Changed

R2020a

See Also

- “Modify the Action Language for a Chart” (Stateflow)
- “Differences Between MATLAB and C as Action Language Syntax” (Stateflow)

Version History

Introduced in R2020a

jc_0655: Prohibition of logical value comparison in Stateflow

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

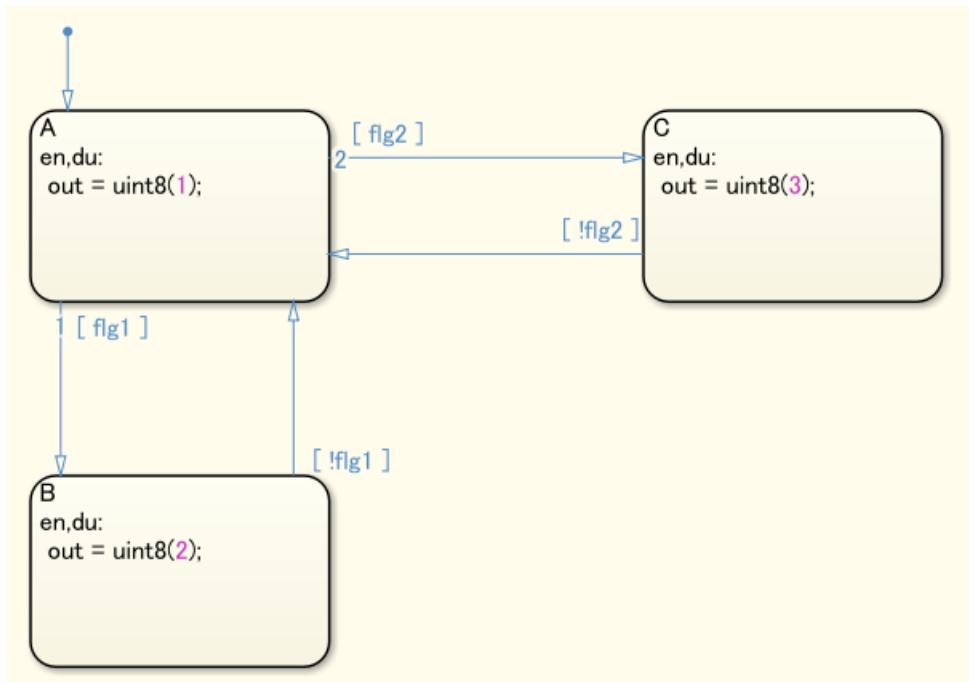
Logical constants shall not be compared to each other.

Custom Parameter

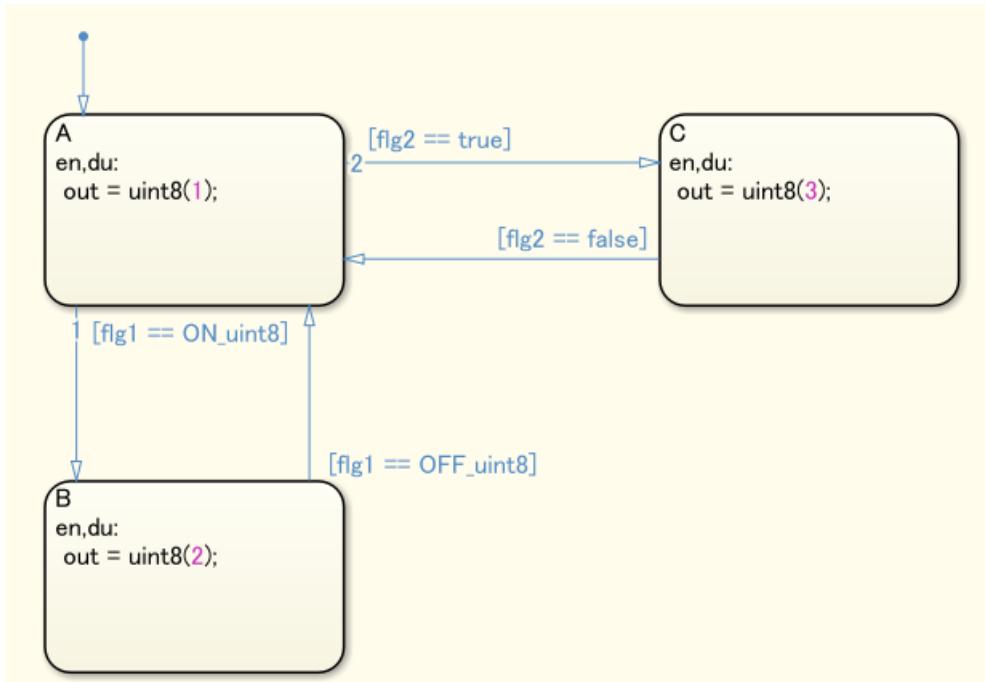
Not Applicable

Example — Correct

Logical constants are not compared to each other.

**Example — Incorrect**

Logical constants are compared to each other.

**Rationale**

Sub ID a:

- Readability improves with consistent use of boolean-valued `signal==true(boolean type constant)` or `(boolean-valued signal)` for logical signal condition expressions.
- Prevents redundancy in the model.
- Deviation from the rule can cause unexpected issues.

Verification

Model Advisor check: "Check prohibited comparison operation of logical type signals" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0451: Use of unary minus on unsigned integers

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

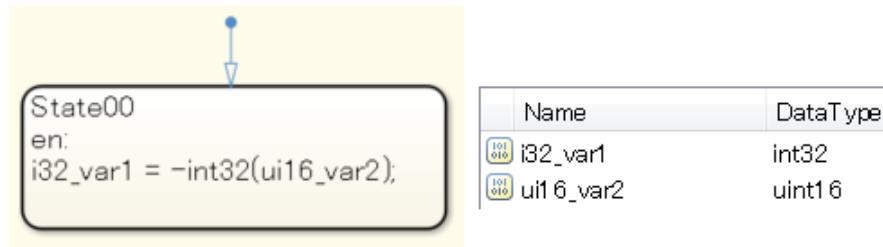
Sub ID a

Unary minus shall not be used on unsigned integers.

Custom Parameter

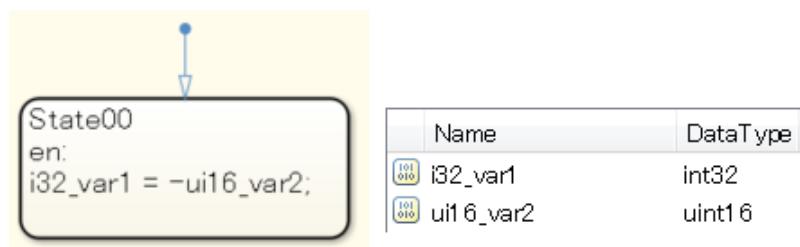
Not Applicable

Example — Correct



Example — Incorrect

Negative values cannot be input into 16-bit environments. (Negative values can be input into 32-bit environments.)



Rationale

Sub ID a:

- As the results are depend on the execution environment, unintended results can occur.

Verification

Model Advisor check: "Check usage of unary minus operations in Stateflow charts" (Simulink Check)

Last Changed

R2020a

See Also

- "Data Types Supported by Simulink"
- Unary Minus

Version History

Introduced in R2020a

jc_0802: Prohibited use of implicit type casting in Stateflow

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

All operations, including substitution, comparison, arithmetic, etc., shall be performed between variables of the same data type.

The data type of the actual arguments and the formal arguments in a function call shall be the same.

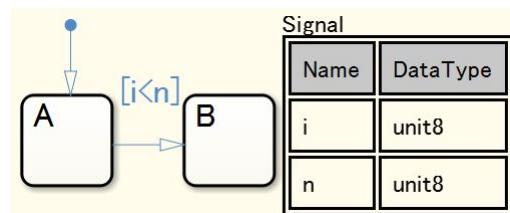
Custom Parameter

Not Applicable

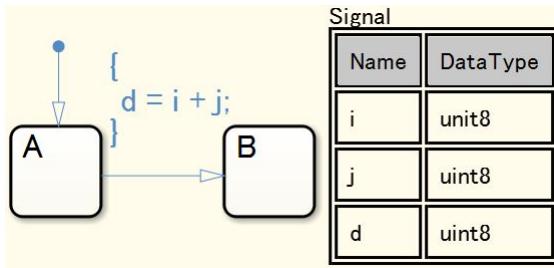
Example — Correct

Variables use the same data type for calculations.

Example: Comparison operation

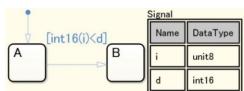


Example: Arithmetic operations and assignment operations (compound expressions)

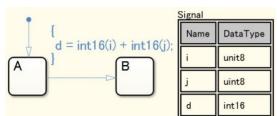


Variables have different data types but are explicitly typecast before calculation.

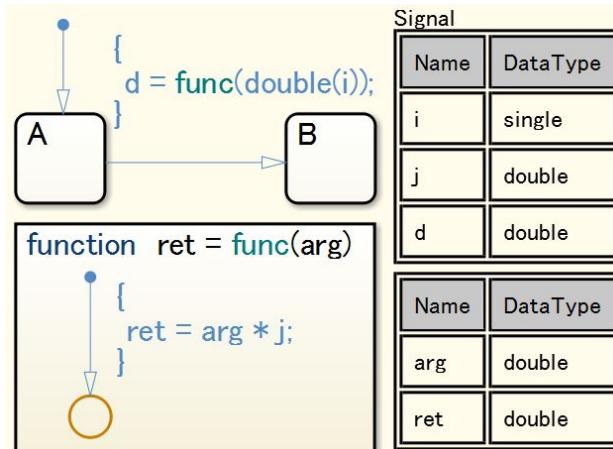
Example: Comparison operation



Example: Arithmetic operations and assignment operations (compound expressions)



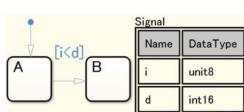
The data type of actual arguments and formal arguments in the function call are the same.



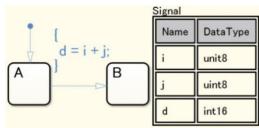
Example – Incorrect

Variables use different data types for calculations.

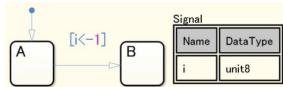
Example: Comparison operation



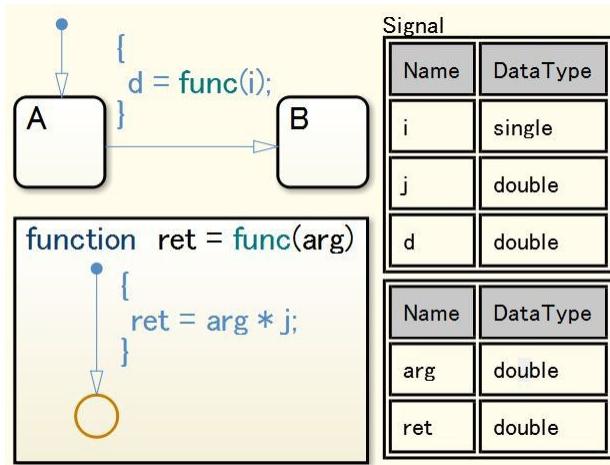
Example: Arithmetic operations and assignment operations (compound expressions)



Calculations are performed between unsigned integer type variables and signed integers.



The data type of actual arguments and formal arguments in the function call are different.



Rationale

Sub ID a:

- Implicit data type conversion can produce unexpected results.

Verification

Model Advisor check: "Check for implicit type casting in Stateflow" (Simulink Check)

Last Changed

R2020a

See Also

- "Type Cast Operations" (Stateflow)
- "Differences Between MATLAB and C as Action Language Syntax" (Stateflow)

Version History

Introduced in R2020a

jc_0803: Passing values to library functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1/a2, b1/b2, c1/c2,
- JMAAB — a1/a2, b1/b2, c1/c2, d1/d2

MATLAB Versions

All

Rule

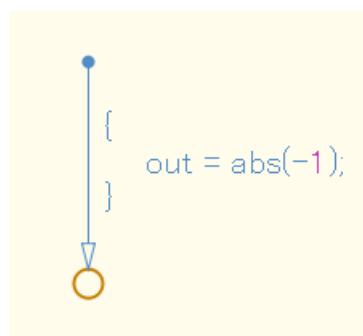
Sub ID a1

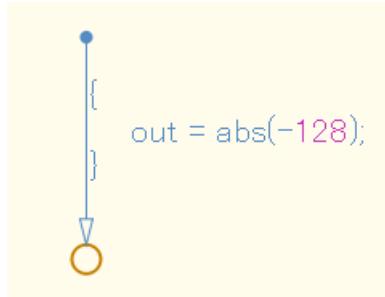
A minimum value for the signed integer type shall not be provided when using the `abs` library function.

Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect**Sub ID a2**

The `abs` library function shall not be used.

Custom Parameter

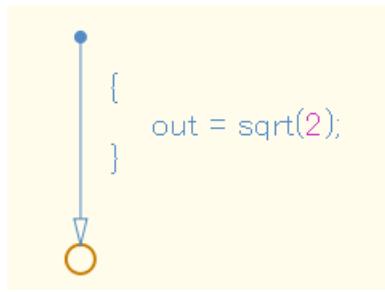
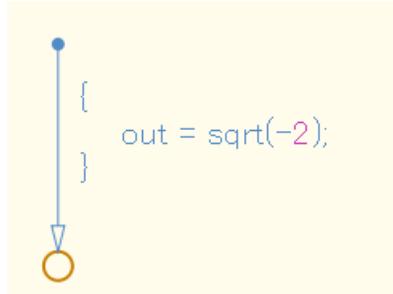
Not Applicable

Sub ID b1

A negative number shall not be entered when using the `sqrt` library function.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Sub ID b2**

The `sqrt` library function shall not be used.

Custom Parameter

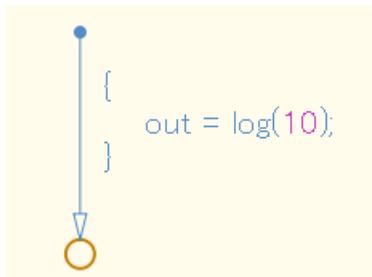
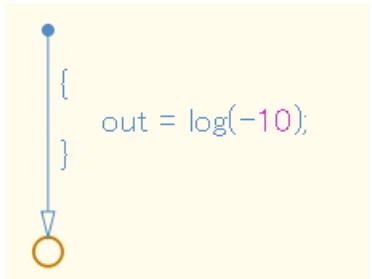
Not Applicable

Sub ID c1

A negative number shall not be entered when using the `log` and `log10` library functions.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Sub ID c2**

The `log` and `log10` library functions shall not be used.

Custom Parameter

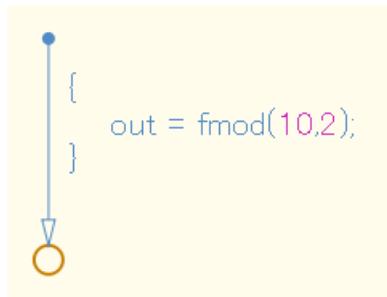
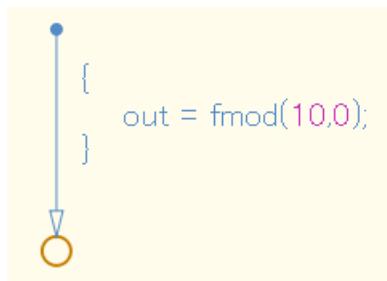
Not Applicable

Sub ID d1

Zero shall not be entered for the second argument when using the `fmod` library function.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect****Sub ID d2**

The `fmod` library function shall not be used.

Custom Parameter

Not Applicable

Rationale

Sub IDs a1, b1, c1, d1:

- The behavior of a library function when an invalid value has been passed is dependent on the processing system and may result in unintended behavior.

Sub IDs a2, b2, c2, d2:

- To avoid duplicate modeling of the same guard process in Simulink and Stateflow, use Simulink to perform arithmetic operations

Verification

Model Advisor check: Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

See Also

- “Library Development” (Embedded Coder)

Version History

Introduced in R2020a

Label Description

jc_0732: Distinction between state names, data names, and event names

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

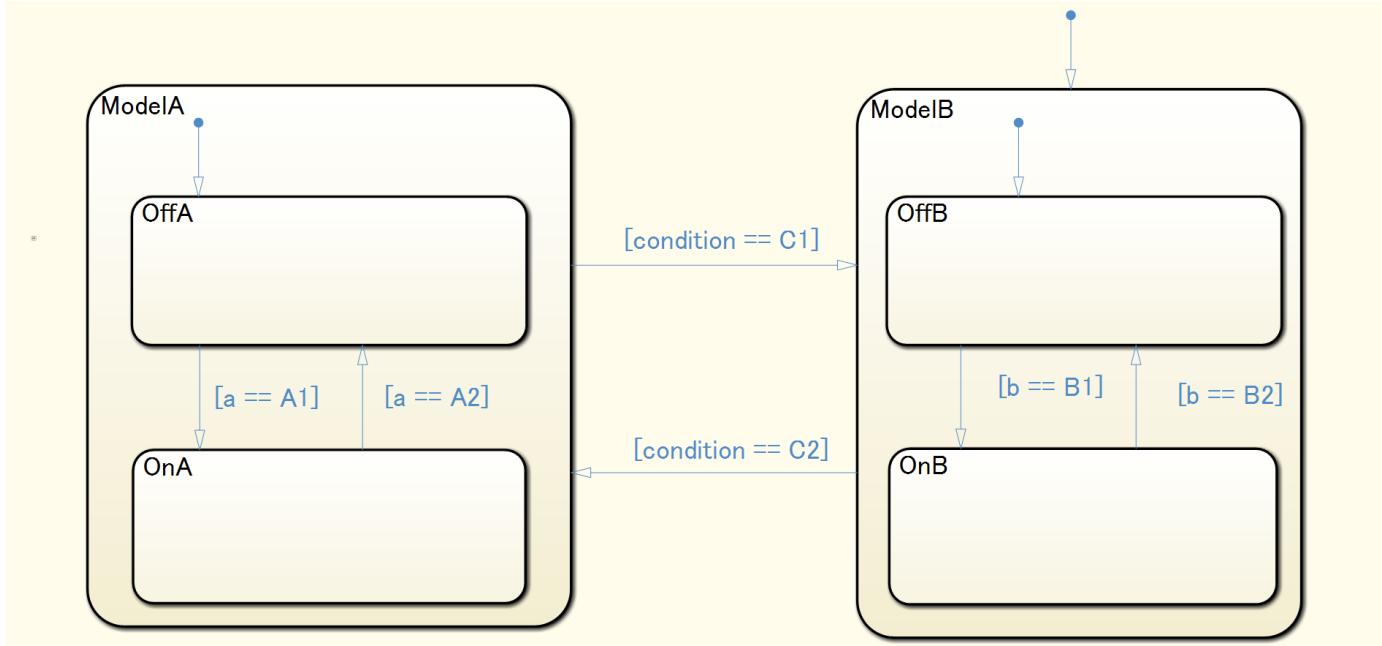
Rule

Sub ID a

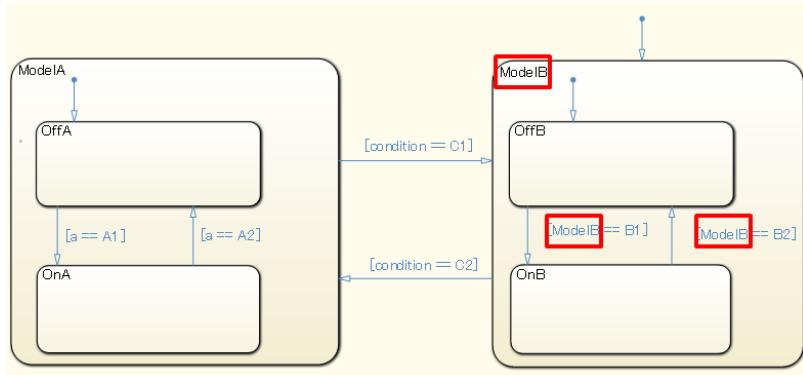
An identical name shall not be used for states, data (inputs and outputs, local data, constants, parameters, data store memory), or event names in a single Stateflow Chart.

Custom Parameter

Not Applicable

Example — Correct**Example — Incorrect**

Names are duplicated.

**Rationale**

Sub ID a:

- Using unique names prevent misunderstanding.

Verification

Model Advisor check: "Check uniqueness of Stateflow state, data and event names" (Simulink Check)

Last Changed

R2020a

Version History

Introduced in R2020a

jc_0730: Unique state name in Stateflow blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

State names in Stateflow Chart block shall be unique.

The content of linked atomic subcharts can be treated as another Stateflow Chart block.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Readability is impaired.
- Deviation from the rule can cause unintended code behavior.

Verification

Model Advisor check: “Check uniqueness of State names” (Simulink Check)

Last Changed

R2020a

See Also

- “Represent Operating Modes by Using States” (Stateflow)
- “Atomic Subcharts” (Stateflow)

Version History

Introduced in R2020a

jc_0731: State name format

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

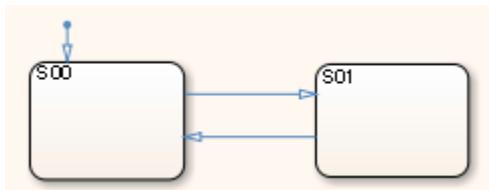
Sub ID a

The state name shall be followed by a new line that does not include a slash (/).

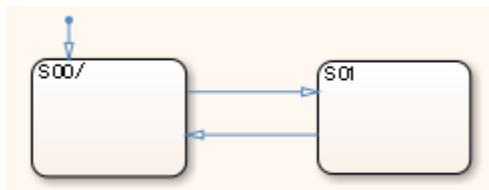
Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect



Rationale

Sub ID a:

- Readability improves when state names are described consistently.

Verification

Model Advisor check: “Check usage of State names” (Simulink Check)

Last Changed

R2020a

See Also

- “Represent Operating Modes by Using States” (Stateflow)

Version History

Introduced in R2020a

jc_0501: Format of entries in a State block

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

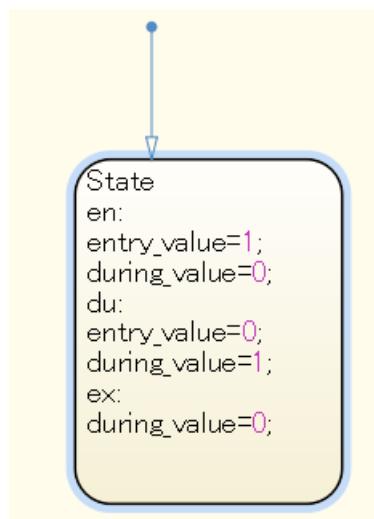
Sub ID a

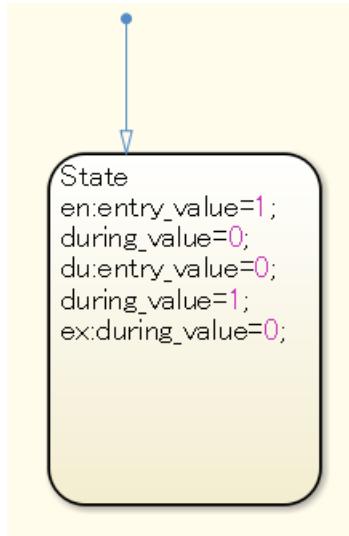
A state action statement shall not be written on the same line as a state action type.

Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect**Rationale**

Sub ID a:

- Readability is impaired.

Verification

Model Advisor check: "Check entry formatting in State blocks in Stateflow charts" (Simulink Check)

Last Changed

R2020a

See Also

- "Represent Operating Modes by Using States" (Stateflow)

Version History

Introduced in R2020a

jc_0736: Uniform indentations in Stateflow blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a, b, c

MATLAB Versions

All

Rule

Sub ID a

State action types shall not have blank spaces at the start of a line.

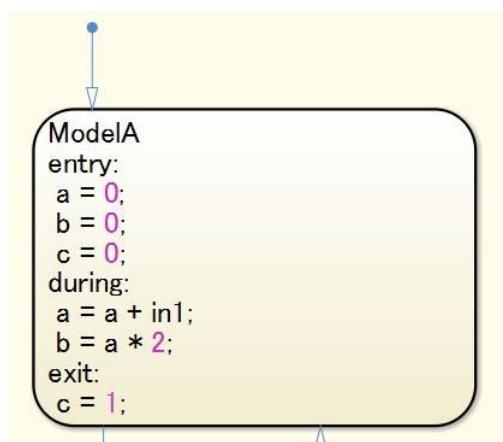
Executable statements shall have one single-byte space at the start of the line.

Custom Parameter

Number of single-byte spaces

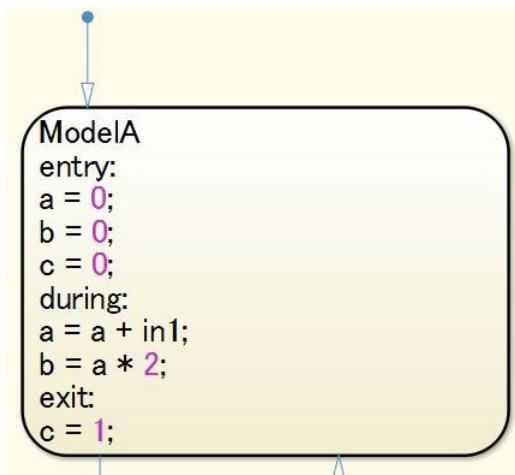
Example — Correct

Executable statements use one single-byte space at the start of the line.



Example — Incorrect

Executable statements do not have a single-byte space at the start of the line.

**Sub ID b**

A blank space shall not be entered before the following:

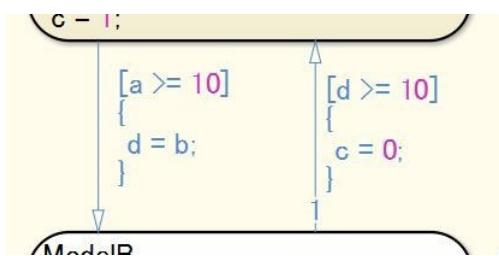
- [of a transition condition
- { of a condition action
- / of a transition action

Custom Parameter

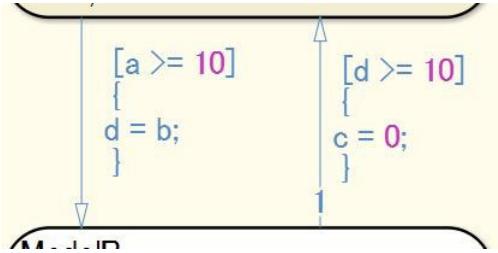
Not Applicable

Example — Correct

A blank space is not entered before the [and { of the transition label condition, condition action, and transition action.

**Example — Incorrect**

A blank space is entered before the [and { of the transition label condition, condition action, and transition action.



Sub ID c

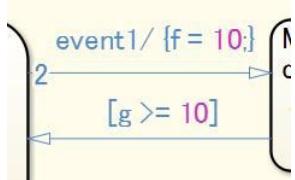
At least one single-byte space shall be entered after the / of a transition action.

Custom Parameter

Number of single-byte spaces

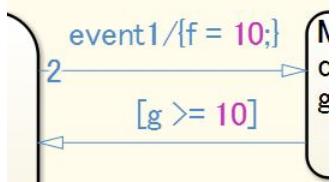
Example — Correct

Single-byte spaces are entered after the / of the transition action.



Example — Incorrect

There are no single-byte spaces after the / of the transition action.



Rationale

Sub ID a:

- Using uniform indents before the executable statement clarifies the link between the state action type of a state label and the execution statement, improving readability.

Sub ID b:

- Using uniform indents for transition conditions, condition actions, and transition actions improves readability.

Sub ID c:

- Consistent use of blank spaces improves readability.

Verification

Model Advisor check: “Check indentation of code in Stateflow states” (Simulink Check)

Last Changed

R2020a

See Also

- “Transition Between Operating Modes” (Stateflow)

Version History

Introduced in R2020a

jc_0770: Position of transition label

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a1/a2/a3

MATLAB Versions

All

Rule

Sub ID a1

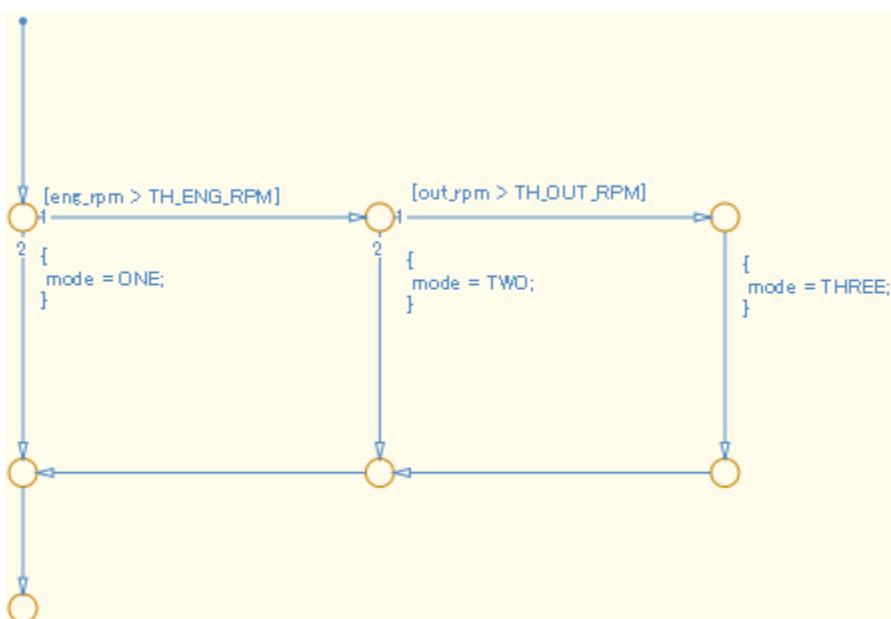
Transition labels are positioned near the source of the transition line.

Custom Parameter

Not Applicable

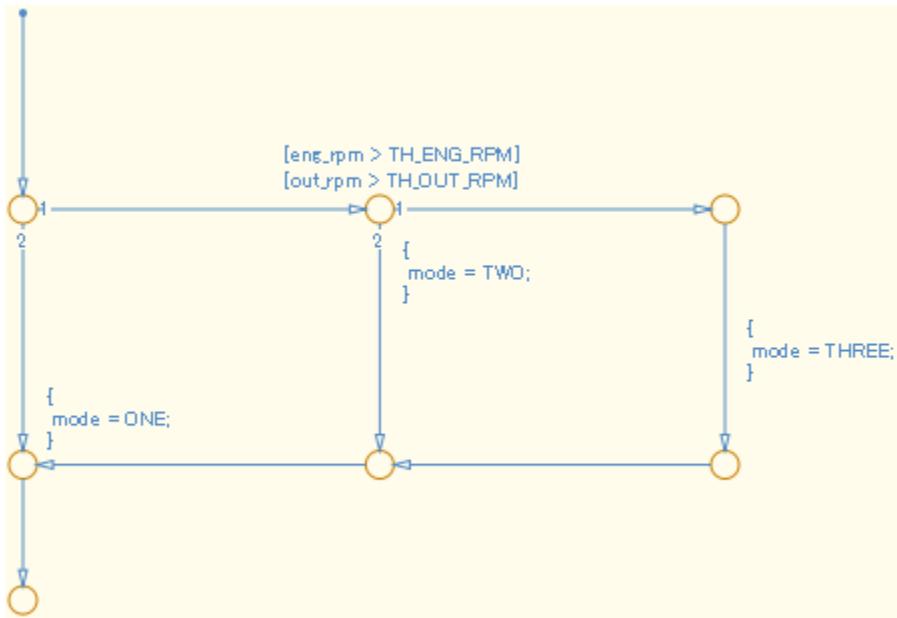
Example — Correct

Transition labels are positioned at the point of origin.



Example — Incorrect

The positioning of transition labels is inconsistent and do not correspond to the transition line.

**Sub ID a2**

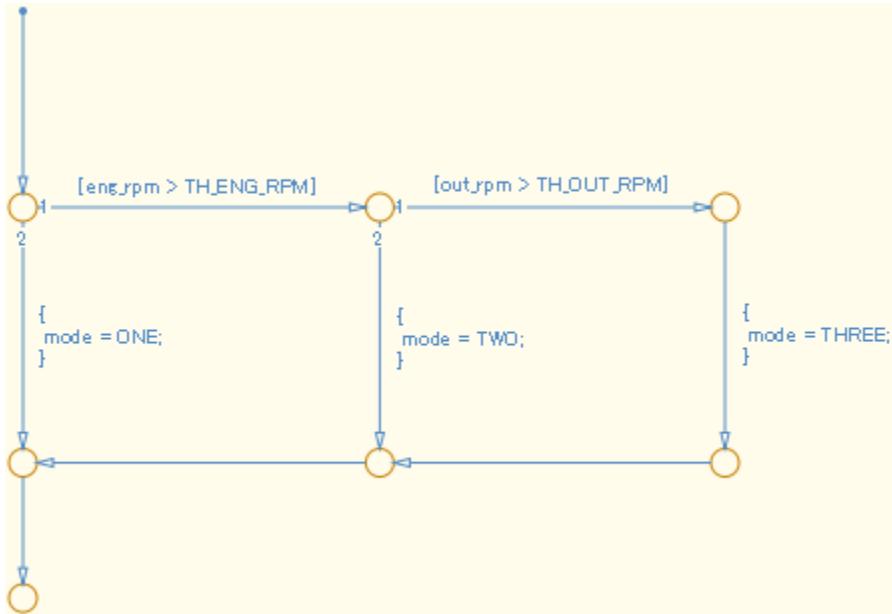
Transition labels are positioned near the center of the transition line.

Custom Parameter

Not Applicable

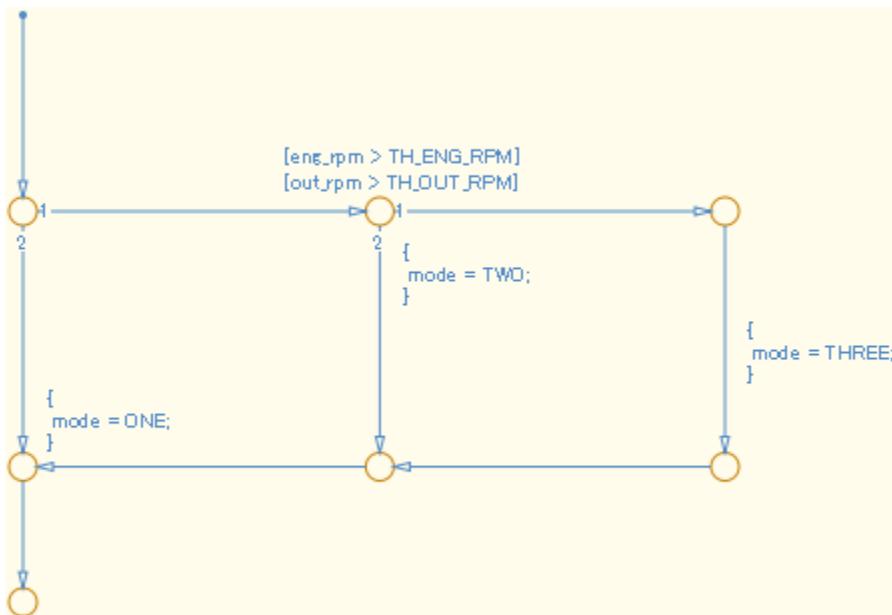
Example — Correct

Transition labels are positioned near the center of the transition line.



Example — Incorrect

The positioning of transition labels is inconsistent and do not correspond to the transition line.



Sub ID a3

Transition labels are positioned as follows:

- Transition labels including transition conditions shall be placed near the source of transition lines.
- Transition labels without transition conditions shall be placed near the destination of transition lines.

Exception

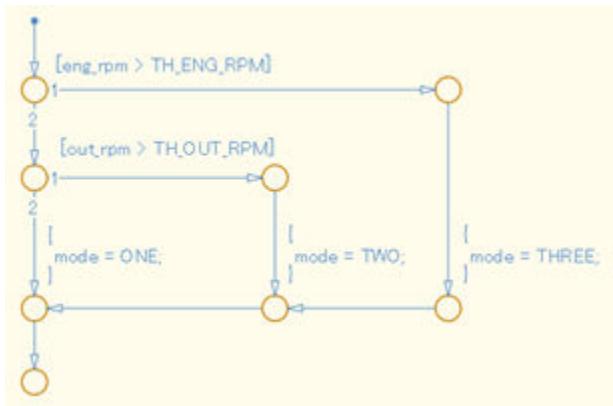
Not applicable for comment-only transition labels.

Custom Parameter

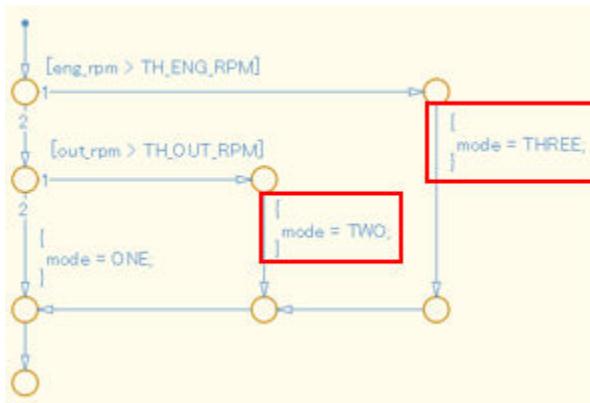
Not Applicable

Example — Correct

Transition labels including transition conditions are placed near sources of transition lines and those without transition conditions are placed near destinations of transition lines.

**Example — Incorrect**

Transition labels without transition conditions are not placed near destinations of transition lines.

**Rationale**

Sub IDs a1, a2:

- Consistent positioning of transition labels makes the correspondence between label and line easier to understand.

Verification

Model Advisor check: "Check placement of Label String in Transitions" (Simulink Check)

Last Changed

R2024b

See Also

- “Transition Between Operating Modes” (Stateflow)

Version History

Introduced in R2020a

jc_0771: Comment position in transition labels

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a1/a2
- JMAAB — a1/a2

MATLAB Versions

All

Rule

Sub ID a1

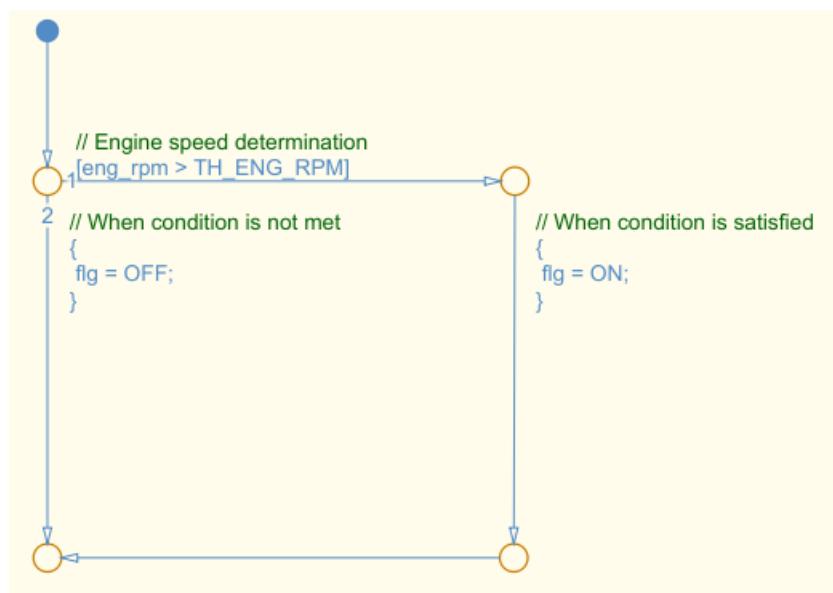
Comments in transition labels shall be positioned above transition conditions, condition actions, and Stateflow events.

Custom Parameter

Not Applicable

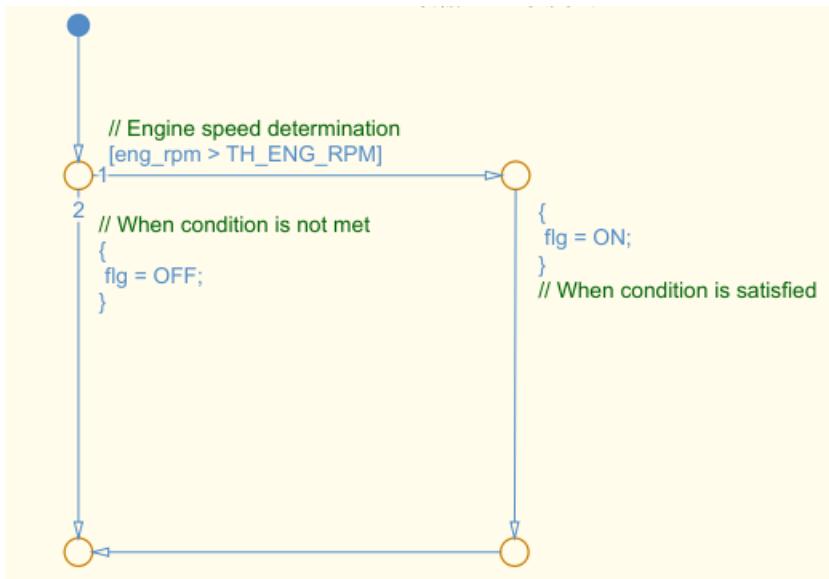
Example — Correct

The position of the comments in the transition label is uniform.



Example — Incorrect

The position of the comments in the transition label is inconsistent.

**Sub ID a2**

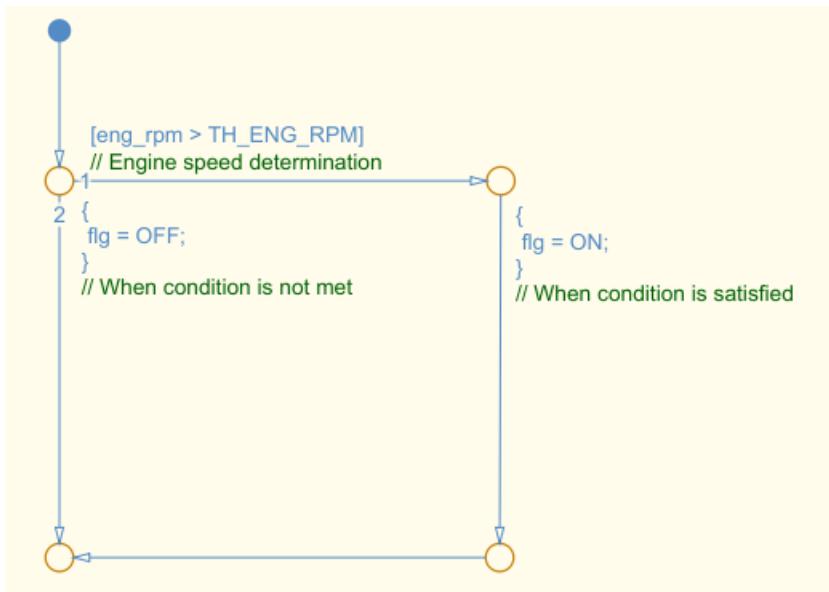
Comments in transition labels shall be positioned below transition conditions, condition actions, and Stateflow events.

Custom Parameter

Not Applicable

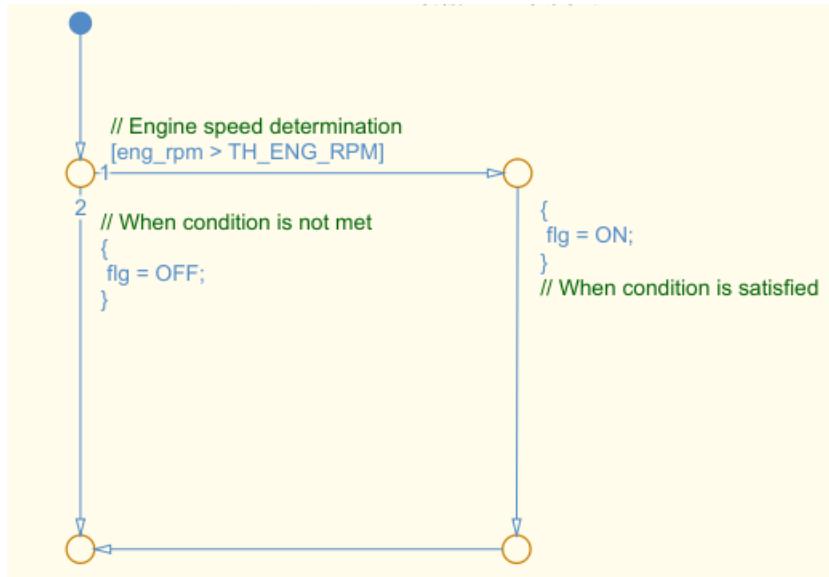
Example — Correct

The position of the comments in the transition label is uniform.



Example — Incorrect

The position of the comments in the transition label is inconsistent.



Rationale

Sub IDs a1, a2:

- Uniform positioning of comments in transition labels clarifies to which transition condition, condition action, transition action, or Stateflow event the label corresponds.

Verification

Model Advisor check: “Check position of comments in transition labels” (Simulink Check)

Last Changed

R2020a

See Also

- “Transition Between Operating Modes” (Stateflow)
- “Add Descriptive Comments in a Chart” (Stateflow)

Version History

Introduced in R2020a

jc_0752: Condition action in transition label

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Parentheses in condition actions shall use only curly brackets on a single line.

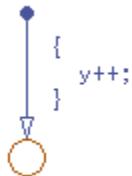
A new line shall start before and after curly brackets.

Custom Parameter

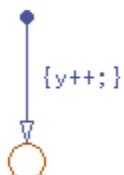
Not Applicable

Example — Correct

Note: The example is for a flow chart, but the rule also applies to state transitions.



Example — Incorrect



Rationale

Sub ID a:

- Clarifying condition actions improves readability.

Verification

Model Advisor check: "Check usage of parentheses in Stateflow transitions" (Simulink Check)

Last Changed

R2020a

See Also

- "Transition Between Operating Modes" (Stateflow)

Version History

Introduced in R2020a

jc_0774: Comments for through transition

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

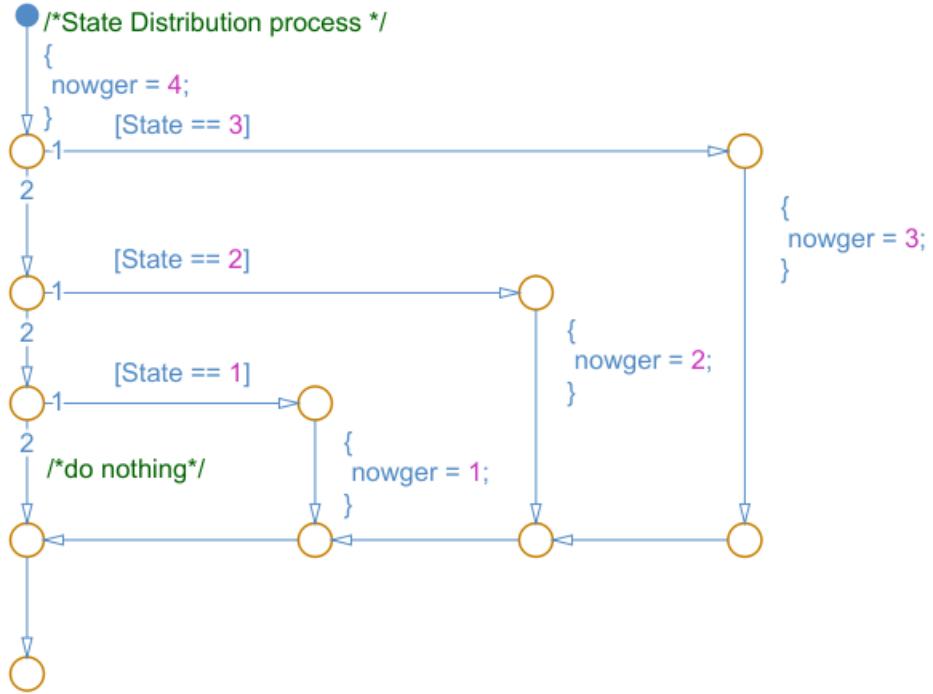
When there is no processing within an unconditional transition, a clarifying comment shall be written on the transition label.

Custom Parameter

Not Applicable

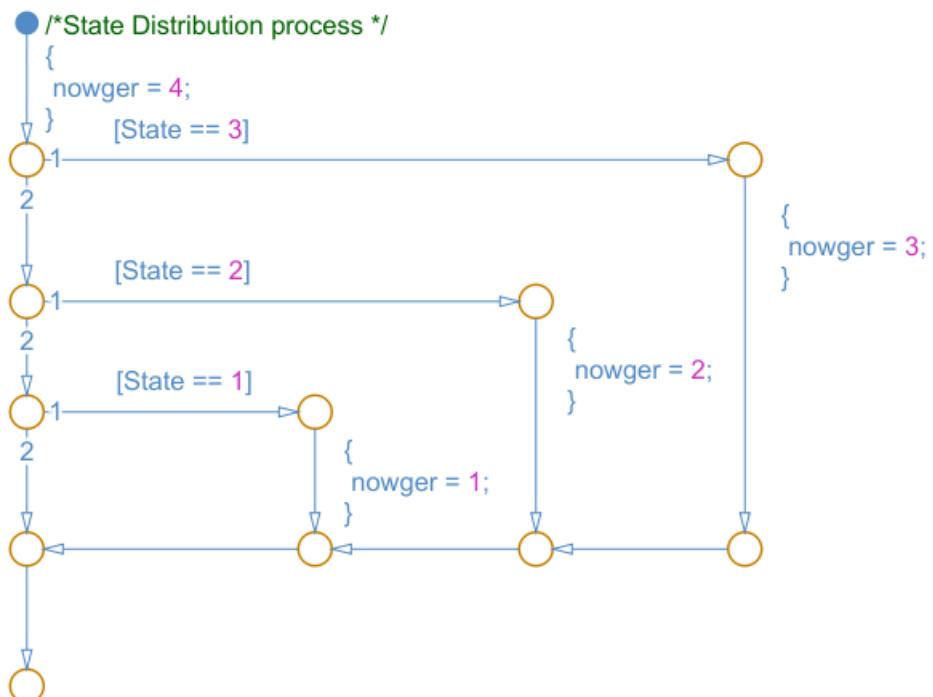
Example — Correct

The clarifying comment /*do nothing*/ is provided.



Example – Incorrect

A clarifying comment is not provided on the condition path, so it is difficult to determine whether the lack of action is intentional.



Rationale

Sub ID a:

- Clarifies that the processing is deliberately excluded.
- The comment that is added to a transition label is also included in the generated code.

Verification

Model Advisor check: "Check for comments in unconditional transitions" (Simulink Check)

Last Changed

R2020a

See Also

- "Add Descriptive Comments in a Chart" (Stateflow)
- "Transition Between Operating Modes" (Stateflow)

Version History

Introduced in R2020a

Miscellaneous

jc_0511: Return values from a graphical function

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — No recommendations
- JMAAB — a

MATLAB Versions

All

Rule

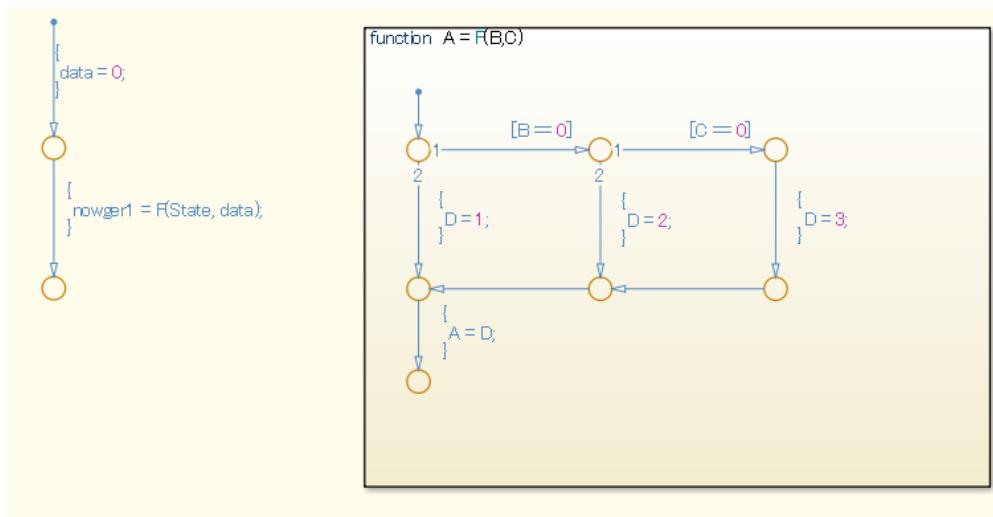
Sub ID a

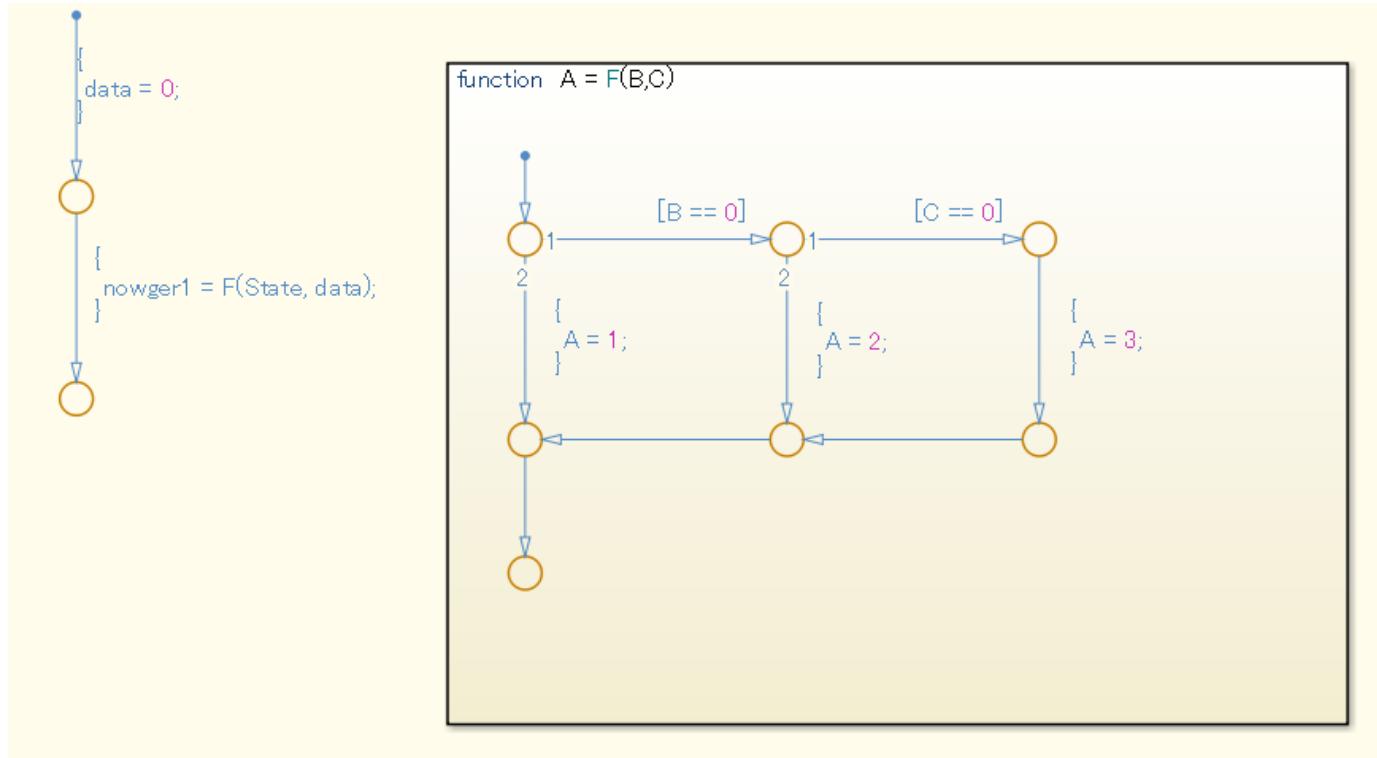
The return value for graphical functions shall be set in one place only.

Custom Parameter

Not Applicable

Example — Correct



Example — Incorrect**Rationale**

Sub ID a:

- Modifications to the output name is limited to prevent the changes from being missed or overlooked.

Verification

Model Advisor check: “Check return value assignments in Stateflow graphical functions” (Simulink Check)

Last Changed

R2020a

See Also

- “Create Flow Charts in Stateflow” (Stateflow)
- “Create Flow Charts by Using Pattern Wizard” (Stateflow)

Version History

Introduced in R2020a

jc_0804: Prohibited use of recursive calls with graphical functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

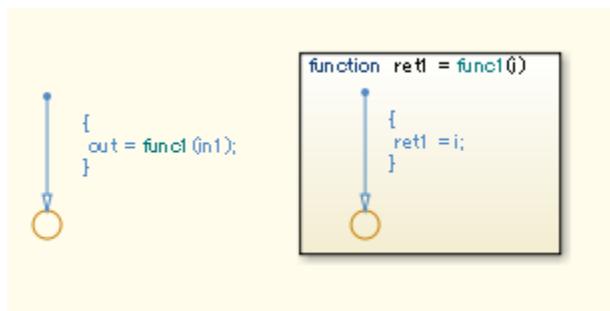
Calls from a graphical function to itself and calls between graphical functions shall be prohibited.

Custom Parameter

Not Applicable

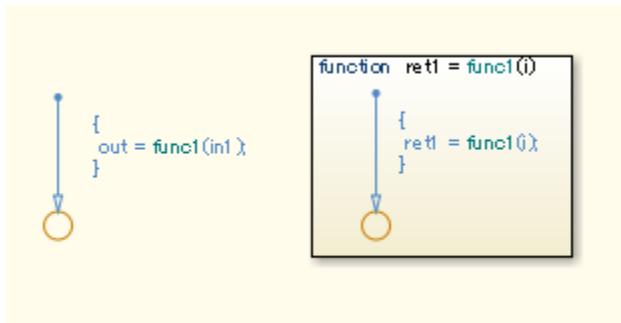
Example — Correct

Processing is performed within the graphical function.

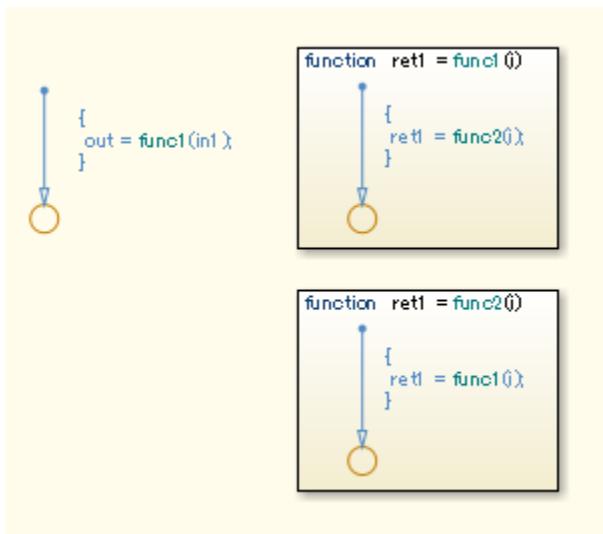


Example — Incorrect

The graphical function is calling itself.



Graphical functions are calling each other.



Rationale

Sub ID a:

- Readability decreases. Deviation from the rule can cause unintended overflows and infinite loops.

Verification

Adherence to this modeling guideline cannot be verified by using a Model Advisor check. For more information, see “Check usage of graphical functions in Stateflow” (Simulink Check).

Last Changed

R2020a

See Also

- “Reuse Logic Patterns by Defining Graphical Functions” (Stateflow)
- “How Stateflow Objects Interact During Execution” (Stateflow)

Version History

Introduced in R2020a

na_0042: Usage of Simulink functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

When using a Simulink Function block in a Stateflow Chart, one or more of the following conditions shall be met:

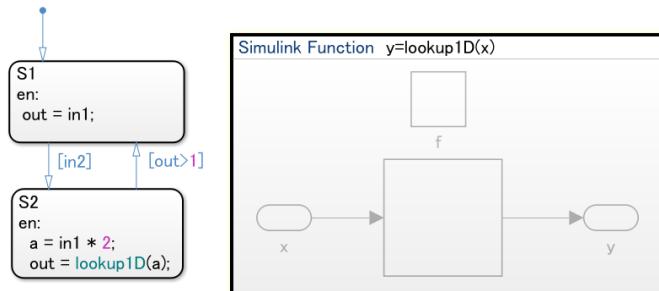
- Input/output variables shall use only local Stateflow Chart data and input data in the Simulink Function block.
- The Simulink Function block shall be called from multiple places in the Stateflow Chart.
- The Simulink Function block shall not be called at every time step.

Custom Parameter

Not Applicable

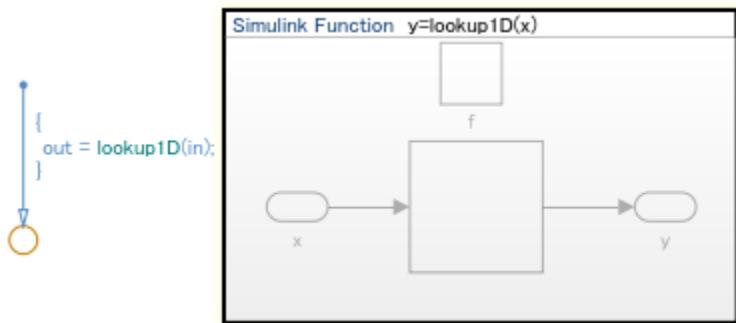
Example — Correct

The Simulink Function block `lookup1D` is not called from every time step and, therefore, can be used.



Example — Incorrect

The Simulink Function block `lookup1D` is called from every time step and, therefore, cannot be used (out is the Stateflow output data)



Rationale

Sub ID a:

- To improve model readability, the use of the Simulink Function block should be used with caution in charts.

Verification

Model Advisor check: "Check usage of Simulink function in Stateflow" (Simulink Check)

Last Changed

R2020a

See Also

- "Simulink Functions Overview"
- "Define a Simulink Function in a Model"

Version History

Introduced in R2020a

na_0039: Limitation on Simulink functions in Chart blocks

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

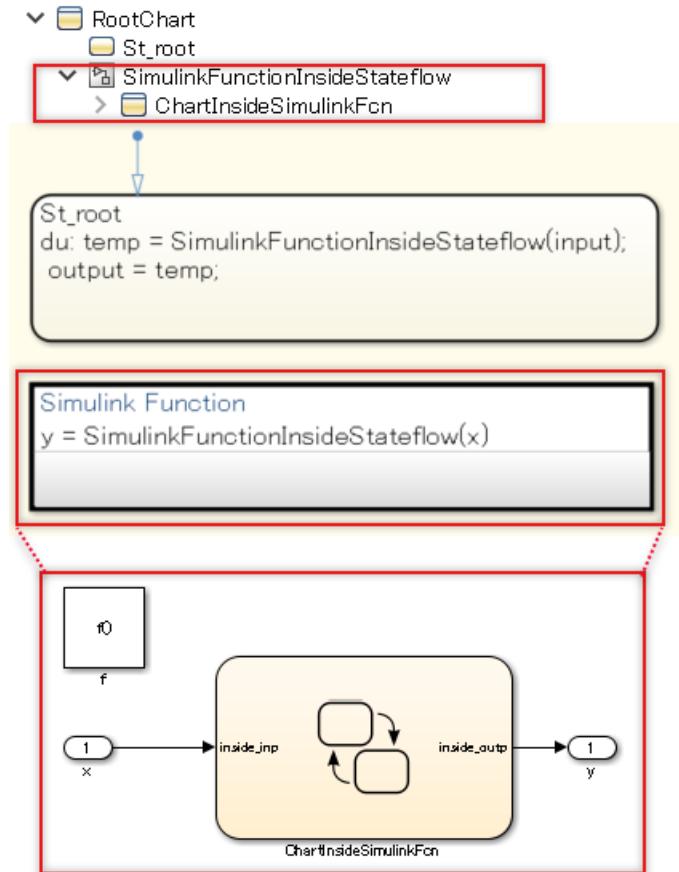
Sub ID a

Stateflow blocks shall not be used in Simulink Function blocks that are included in a Stateflow Chart.

Custom Parameter

Not Applicable

Example — Incorrect



Rationale

Sub ID a:

- Readability decreases and can result in design errors.

Verification

Model Advisor check: "Check use of Simulink in Stateflow charts" (Simulink Check)

Last Changed

R2020a

See Also

- “Simulink Functions Overview”
- “Define a Simulink Function in a Model”

Version History

Introduced in R2020a

MATLAB

- “MATLAB Appearance” on page 5-2
- “MATLAB Data and Operations” on page 5-7
- “MATLAB Usage” on page 5-15

MATLAB Appearance

na_0025: MATLAB Function header

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

The MATLAB Function block shall have a descriptive header.

Information in the header can include, but is not limited to:

- Function name
- Description of function
- Assumptions and limitations
- Description of changes from previous versions
- Lists of inputs and outputs

Custom Parameter

Not Applicable

Example

```
%% Function Name: NA_0025_Example_Header
%
% Description: An example of a header file
%
% Assumptions: None
%
% Inputs:
%     List of input arguments
%
% $Revision: R2020a$
% $Author: MathWorks Advisory Board (MAB)$
```

```
% $Date: November 20, 2019$  
%-----
```

Rationale

Sub ID a:

- Improves readability, model simulation, testability, and workflow.
- Code generation may not be possible.

Verification

Adherence to this modeling guideline cannot be verified by using a Model Advisor check.

Last Changed

R2020a

See Also

- “Implement MATLAB Functions in Simulink with MATLAB Function Blocks”

Version History

Introduced in R2020a

na_0018: Number of nested if/else and case statement

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

The number of nested levels in `if /else` and `case` statements shall be limited, typically to three levels.

Custom Parameter

Maximum nested levels

Rationale

Sub ID a:

- Improves readability
- Code generation may not be possible.

Verification

Model Advisor check: “Check nested conditions in MATLAB Functions” (Simulink Check)

Last Changed

R2020a

See Also

- “Loops and Conditional Statements”

Version History

Introduced in R2020a

MATLAB Data and Operations

na_0024: Shared data in MATLAB functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

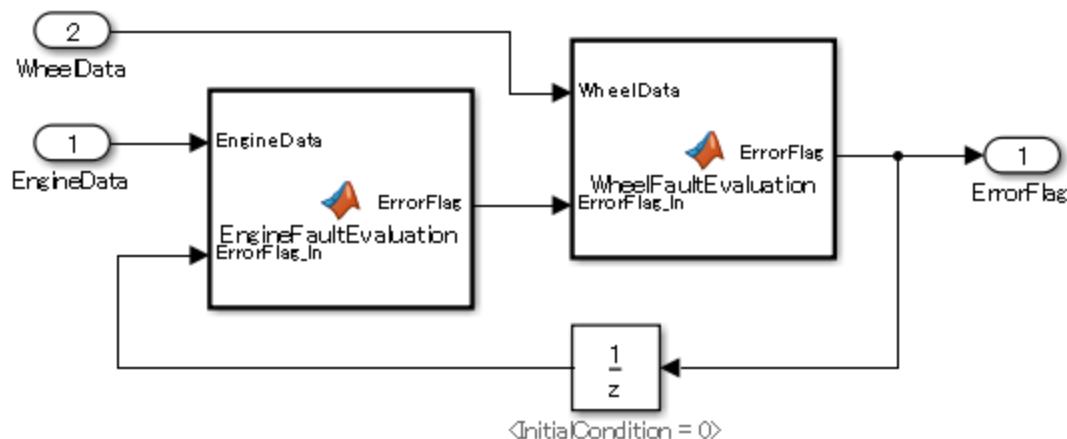
Sub ID a

Signal lines shall be used to connect data that is shared between MATLAB Function blocks.

Custom Parameter

Not Applicable

Example — Correct



```
function ErrorFlag = EngineFaultEvaluation(EngineData,ErrorFlag_In)
 %#codegen
 RMP_HIGH = 10000;
 RMP_LOW = 10;
```

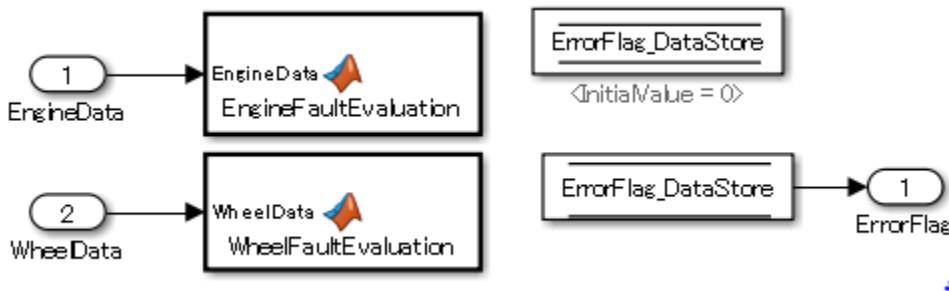
```

HIGHRPMFAULT = 2^1;
LOWRPMFAULT = 2^2;
ErrorFlag = ErrorFlag_In;
if EngineData > RPM_HIGH
    ErrorFlag = bitor(ErrorFlag,HIGHRPMFAULT);
end
if EngineData < RPM_LOW
    ErrorFlag = bitor(ErrorFlag,LOWRPMFAULT);
end

function ErrorFlag = WheelFaultEvaluation(WheelData,ErrorFlag_In)
 %#codegen
 SLIP_HIGH = 1000;
 WHEELSLIP = 2^3;
 ErrorFlag = ErrorFlag_In;
 if WheelData > SLIP_HIGH
     ErrorFlag = bitor(ErrorFlag,WHEELSLIP);
 end
end

```

Example — Incorrect



```

function EngineFaultEvaluation(EngineData)
 %#codegen
 global ErrorFlag_DataStore
 RMP_HIGH = 10000;
 RMP_LOW = 10;
 HIGHRPMFAULT = 2^1;
 LOWRPMFAULT = 2^2;
 if EngineData > RPM_HIGH
     ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,HIGHRPMFAULT);
 end
 if EngineData < RPM_LOW
     ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,LOWRPMFAULT);
 end

function WheelFaultEvaluation(WheelData)
 %#codegen
 global ErrorFlag_DataStore
 SLIP_HIGH = 1000;
 WHEELSLIP = 2^3;
 if WheelData > SLIP_HIGH
     ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,WHEELSLIP);
 end

```

end

Rationale

Sub ID a:

- When a data store is used, the readability of the data flow decreases and can lead to errors in the update reference timing.

Verification

Model Advisor check: "Check MATLAB code for global variables" (Simulink Check)

Last Changed

R2020a

See Also

- "Implement MATLAB Functions in Simulink with MATLAB Function Blocks"
- "Signal Lines"
- "Resolve Signal Objects for Output Variables"

Version History

Introduced in R2020a

na_0031: Definition of default enumerated value

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Method `getDefaultValue()` shall be used to explicitly define the default value of an enumeration.

Custom Parameter

Not Applicable

Example — Correct

See “Specify a Default Enumerated Value”

```
classdef BasicColors < Simulink.IntEnumType
    enumeration
        Red(0)
        Yellow(1)
        Blue(2)
    end
    methods (Static)
        function retVal = getDefaultValue()
            retVal = BasicColors.Blue;
        end
    end
end
```

Example — Incorrect

```
classdef(Enumeration) BasicColors < Simulink.IntEnumType
    enumeration
        Red(0)
        Yellow(1)
        Blue(2)
```

```
    end  
end
```

Rationale

Sub ID a:

- When an enumerated type does not have a clearly defined a default value, the first enumeration string that is described will be defined as the default, which may not be as intended.

Verification

Model Advisor check: "Check usage of enumerated values" (Simulink Check)

Last Changed

R2020a

See Also

- “Use Enumerated Data in Simulink Models”
- “Define Enumerated Data Types” (Stateflow)
- “Instantiate Enumerations”

Version History

Introduced in R2020a

na_0034: MATLAB Function block input/output settings

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

The data type in the model explorer shall be defined for input and output to the MATLAB Function block.

Custom Parameter

Not Applicable

Rationale

Sub ID a:

- Defining the data type for input and output to the MATLAB Function block helps prevent simulation errors and unexpected behavior.

Verification

Model Advisor check: "Check input and output settings of MATLAB Functions" (Simulink Check)

Last Changed

R2020a

See Also

- “Implement MATLAB Functions in Simulink with MATLAB Function Blocks”

Version History

Introduced in R2020a

MATLAB Usage

na_0016: Source lines of MATLAB Functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

The length of MATLAB functions shall be limited. The recommended limit is 60 lines of code. Sub-functions can use an additional 60 lines of code.

This restriction applies to MATLAB functions that reside in the Simulink block diagram and external MATLAB files with a .m extension.

Custom Parameter

Maximum effective lines of code per function

Rationale

Sub ID a:

- Improves readability and workflow
- Code generation may not be possible.

Verification

Model Advisor check: "Check lines of code in MATLAB Functions" (Simulink Check)

Last Changed

R2020a

See Also

- “Integrate MATLAB Functions in a Stateflow Charts” (Stateflow)
- “Implement MATLAB Functions in Simulink with MATLAB Function Blocks”

Version History

Introduced in R2020a

na_0017: Number of called function levels

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

The number of sub-function levels shall be limited, typically to three levels.

MATLAB Function blocks that reside in the Simulink block diagram level counts as the first level, unless it is simply a wrapper for an external MATLAB file with a .m extension. This includes functions that are defined within the block and those in separate .m files.

Exclusions

The following function types are excluded from the number of levels:

- Standard utility functions, such as built-in functions `sqrt` or `log`
- Commonly used custom utility functions

Custom Parameter

Maximum function call levels

Rationale

Sub ID a:

- Improves readability and testability

Verification

Model Advisor check: "Check the number of function calls in MATLAB Function blocks" (Simulink Check)

Last Changed

R2020a

See Also

- “Implement MATLAB Functions in Simulink with MATLAB Function Blocks”
- “Integrate MATLAB Functions in a Stateflow Charts” (Stateflow)
- “Exponents and Logarithms”

Version History

Introduced in R2020a

na_0021: Strings in MATLAB functions

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

Assignment statements for strings shall not be used in MATLAB functions.

Custom Parameter

Not Applicable

Example — Incorrect

An assignment statement for strings is being used in the MATLAB function.

```
function y = fcn(u)
%#codegen

str = 'A';

for i = 1:u
    str = [str 'B'];
end

if strcmp(str, 'ABB')
    y = int16(1);
else
    y = int16(0);
end
end
```

Rationale

Sub ID a:

- MATLAB functions store strings as character arrays. As a result, storing strings of different lengths in the same variable does not support dynamic memory allocation, which prevents the strings from being stored.

Consider using enumerated types when a string is used in a Switch Case block

Verification

Model Advisor check: "Check usage of character vector inside MATLAB Function block" (Simulink Check)

Last Changed

R2020a

See Also

- “Implement MATLAB Functions in Simulink with MATLAB Function Blocks”
- “Integrate MATLAB Functions in a Stateflow Charts” (Stateflow)

Version History

Introduced in R2020a

na_0022: Recommended patterns for Switch/Case statements

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — Not supported

MATLAB Versions

All

Rule

Sub ID a

Switch/Case statements shall use constant values for Case arguments.

Input variables shall not be used in Case arguments.

Custom Parameter

Not Applicable

Example — Correct

```
function outVar = NA_0022_Pass(SwitchVar)
 %#codegen
 switch SwitchVar
 case Case_1_Parameter % Parameter
     outVar = 0;
 case NA_0022.Case_2 % Enumerated Data Type
     outVar = 2;
 otherwise
     outVar = 10;
 end
end
```

Example — Incorrect

```
function outVar = NA_0022_Fail(Case_1,Case_2,Case_3,SwitchVar)
 %#codegen
 switch SwitchVar
 case Case_1
```

```
        outVar = 1;
case Case_2
        outVar = 2;
case Case_3
        outVar = 3;
otherwise
        outVar = 10;
end
end
```

Rationale

Sub ID a:

- Improves model simulation and testability.
- Code generation may not be possible.

Verification

Model Advisor check: "Check usage of recommended patterns for Switch/Case statements" (Simulink Check)

Last Changed

R2020a

See Also

- "Loops and Conditional Statements"

Version History

Introduced in R2020a

jc_0801: Prohibited use of the /* and */ comment symbols

Guideline Publication

Control Algorithm Modeling Guidelines - Using MATLAB, Simulink, and Stateflow

- Version 6.0

Sub ID Recommendations

- NA-MAAB — a
- JMAAB — a

MATLAB Versions

All

Rule

Sub ID a

As comment symbols /* and */ are automatically assigned in the generated code, the symbol shall not be used in:

- Code Generate Templates (.cgt) files
- mpt.Signal description
- mpt.Parameter description

Custom Parameter

Not Applicable

Example — Incorrect

Included in a .cgt file.

```

ert_code_template_NG.cgt + 
%%
%% Custom file banner section (optional)
%%
<FileBanner style="classic">
File: %<FileName>
/*
Code generated for Simulink model '%<ModelName>'.
*/
Model version : %<ModelVersion>

```

コード生成

```

/*
* File: mo0004a_NG.c
* /*
* Code generated for Simulink model 'mo0004a_NG'.
* */
* Model version : 1.10

```

Included in the mpt.Signal description.

説明:
/*sig1 : input signal*/

```

/* Exported data definition */
const volatile real32_T TWO = 2.0F;      /* */ 
volatile real32_T sig1;                  /* */ 

```

コード生成

Rationale

Sub ID a:

- Since comment symbols /* and */ are automatically assigned in the generated code, comments can be unintentionally nested and behave differently than expected.

Verification

Model Advisor check: "Check for use of C-style comment symbols" (Simulink Check)

Last Changed

R2020a

See Also

- “Code Generation Template (CGT) Files” (Embedded Coder)
- “MPT Data Object Properties” (Embedded Coder)
- “Specify Comment Style for C/C++ Code” (Embedded Coder)
- “Add Custom Comments for Variables in the Generated Code” (Embedded Coder)

Version History

Introduced in R2020a

Considerations

- “Considerations for Determining Guideline Operation Rules” on page 6-2
- “Considerations for Applying Guidelines to a Project” on page 6-5

Considerations for Determining Guideline Operation Rules

Prior to selecting the modeling guidelines to adopt for your project, it is important that you consider various aspects of your project and models, such as:

- “Process Definition and Development Environment” on page 6-2
- “MATLAB and Simulink Versions” on page 6-2
- “MATLAB and Simulink Settings” on page 6-2
- “Usable Blocks” on page 6-3
- “Using Optimization and Configuration Parameters” on page 6-3

Process Definition and Development Environment

The model base development that utilizes simulation is suitable for developing a safe product. However, this does not mean that a system is safe simply because the design can be simulated. While high quality control and functions is necessary, the process definition and development environment being used is equally important. The foundation for a safe system is determined at the start of the project, long before development begins.

MATLAB and Simulink Versions

The version of MATLAB and Simulink that is used at each development stage is determined at the start of the project. That version must be used by everyone during that development stage.

Different MATLAB versions can be used for different stages in the development process. For example, you can generate and verify the code in R2017b and then use Simulink Design Verifier™ to develop test cases in R2020a.

It is necessary to regularly check the bug report published by MathWorks, which are available on the MathWorks website at <https://www.mathworks.com/support/bugreports>. Depending on the bug, a version change may be required; a decision that can be reversed if necessary. During this evaluation, it is important to consider risk from both:

- Malfunctions that result from a bug
- Result from upgrading the version

It is necessary to always have a process that allows adaptation to the latest version and to appropriately evaluate and judge what is the safest option.

MATLAB and Simulink Settings

MATLAB and Simulink settings shall adhere to the project. It is important that Simulink settings that affect appearance are applied consistently across the project.

Options to be unified include:

- Simulink environment settings:
 - New model standard font settings (block, line, annotation)
- Mask (Edit mask):

- Icons and Ports
- Information display:
 - Library links
 - Sample Time
 - (Block) Sorted execution order
 - (Signals and ports) Wide Non-scalar Lines
 - (Signals and ports) Port data types

See guidelines:

- na_0004: Simulink model appearance settings
- db_0043: Model font and font size

Usable Blocks

There are many blocks in Simulink, however, not all are suitable for all aspects of a project. For example, only some blocks are suitable for generating production-quality code. Or, depending on the block, a function using a combination of basic blocks can be represented by using one block. Usable blocks and design should be defined and limited to the requirements and specifications of the project.

Significantly limiting the number of available blocks can cause adverse effects, such decreased readability due to variation within the descriptions for the same function, decreased code efficiency, and increased user libraries.

You must register custom blocks in the project's user library.

See guideline db_0143: Usable block types in model hierarchy for defining usable blocks

Using Optimization and Configuration Parameters

It is important to consider how you are using optimization options and configuration parameters for your project.

Optimization Parameters

Optimization options significantly affect generated code. Closely evaluate and apply the optimization options with regards to how they impact the security and safety considerations for your project or product.

As an example of how optimization parameters can impact a process:

For embedded automotive products, it is critical that processing time is fast and RAM/ROM requirement are minimal. To accommodate these requirements, optimization parameters are applied on the **Conditional Input Branch Execution** pane. These optimization parameters improve the computation rate by executing only where the condition holds during execution of the conditional branch by using the Switch block.

In contrast, for the aviation industry, the **Conditional Input Branch Execution** pane is disabled because stabilizing the execution speed is key. Calculation on both sides is preferred in order to maintain a stable computation time, even if calculation is needed only on the side where the condition holds.

Configuration Parameters

Consider these configuration parameters:

Hardware Implementation Settings

Describes model system hardware characteristics, including products and test hardware configuration setup for simulation and code generation. Configure these parameters so they are compatible with the microcomputer that the project uses. Unintended utility functions can be inserted if signed integer division rounding is undefined.

Model Reference Settings

Specified when using model references. Refers to options to include other models in this model, options to include this model in another model, and build options of simulation and code generation targets.

Simulation Target Setting

Configures a simulation target of a model with MATLAB Function, Stateflow Chart (Stateflow), or Truth Table (Stateflow) blocks.

High-Integrity Configuration Settings

For additional information about the high-integrity configuration settings, see “Configuration Parameter Considerations” in the “High-Integrity System Modeling” guidelines.

Code Generation Configuration Settings

For additional information the about the code generation configuration settings, see the “Code Generation” modeling guidelines.

Considerations for Applying Guidelines to a Project

It is important that you consider the following when applying modeling guidelines to a project:

- “Using the Model Analysis Process When Applying Guidelines” on page 6-5
- “Adoption of the Guideline Rule and Process Settings” on page 6-5
- “Setting the Guideline Rule Application Field and the Clarifying the Exclusion Condition” on page 6-5
- “Parameter Recommendations in the Guidelines” on page 6-6
- “Verifying Adherence to the Guidelines” on page 6-6
- “Modifying Adherence to the Guidelines” on page 6-6

Using the Model Analysis Process When Applying Guidelines

Model design specification should be defined prior to reviewing the guidelines. Doing so makes the process of determining which guidelines to apply and the implementation of the guidelines more efficient.

For example, the analysis of a simple model can use function `sldiagnostics` to investigate how often a specific block is used. Adjust the operation rules list by specifying blocks that are frequently used and those that are not.

Furthermore, reusability at a later stage is improved by adding rules that:

- Unify description styles
- Anticipate in advance the man-hours needed to correct models
- Measuring tendencies, such as where to place blocks that have feedback status variables (Unit Delay block), whether the Unit Delay block should be inside or outside the subsystem, or whether the Abs block should be set on the output side of the subsystem, and if it should process at the input side after receiving a signal.

Adoption of the Guideline Rule and Process Settings

At the start of the project, it should be determined which guidelines apply to each development process. The guidelines should be evaluated and applied so that they correspond with the development process. Considerations may include questions such as:

- Will the guideline be applied only at the code generation stage?
- Will the adopted guideline rule change for each process stage?

Setting the Guideline Rule Application Field and the Clarifying the Exclusion Condition

The field to which the guidelines apply must be determined. For example, guidelines can be:

- Limited to a model that represents the AUTOSAR field of application
- Applied to a general software field, such as where models implement interrupts (add processes that prohibit interruption during calculation).

- Specific to fields where general engineers edit the models. The intention of these rules is to ensure that the models are easily understandable in those fields.

Note Specialized fields can be excluded from the constraints of these guidelines by limiting the scope and applying unique set of guidelines that are specific in this environment.

Specialized fields, such as those where modelers design custom library blocks, are not typically targeted by these guidelines.

Furthermore, when having a control model that is operated with Rapid Control Prototyping (RCP), the entire model should not be set as a target; instead, the field needs to be limited. It is necessary to generate the code and review the areas that are implemented in the built-in microcomputer as well as the areas that are not. These guidelines do not apply to control models such as those scheduler models that are made solely for RCP and are not implemented, or for interface sections with blocks that correspond to drivers such as CAN and PWM signals for operating actual machines.

Parameter Recommendations in the Guidelines

Guidelines should not be adopted as they are written without further evaluation.

Implementation of guideline rules and parameter recommendations should be evaluated to determine the impact on the project and the development processes being used. In addition, consideration needs to be taken as to the effect on other guidelines and how applying custom parameters can affect simulation or code generation.

Verifying Adherence to the Guidelines

At the beginning of a project, it is important to determine how and when the project will be evaluated to ensure adherence to the guidelines.

The decision whether to use an automated checking mechanism (third party or internal) or perform manual checks is very important. Also, the stage at which the checks occur, as well as developing a system for revising the check rule criteria, is important.

Automated checking can significantly reduce the time required for review. It is recommended that an additional, manual review also be performed by a skilled person, even if everything can be checked automatically.

Modifying Adherence to the Guidelines

The decision to apply a guideline or a rule can change. When doing so, it is important to specify a process and procedure for determine the root cause of the request and evaluate the potential impact the change can have on the project and the organization.

When evaluating the change request, first listen to the needs of the modeler and determine the root cause of the request. When the request is based on the user not understanding block usage or a guideline rule, training should occur instead of revising the rule.

The procedure to relax the rules as needed should be implemented when there are restrictions due to company objectives and control specifications or hardware (such as microcomputers).

Using Simulink and Stateflow

Understanding Model Architecture

When evaluating the modeling guidelines for your project, it is important that you understand the architecture of your controller model, such the function/subfunction layers, schedule layer, control flow layer, section layer, and data flow layer.

Hierarchical Structure of a Controller Model

This section provides a high-level overview of the hierarchical structuring in a basic model, using a controller model as an example. This table defines the layer concepts in a hierarchy.

	Layer concept	Layer purpose
Top Layer	Function layer	Broad functional division
	Schedule layer	Expression of execution timing (sampling, order)
Bottom Layer	Sub function layer	Detailed function division
	Control flow layer	Division according to processing order (input → judgment → output, etc.)
	Selection layer	Division (select output with Merge) into a format that switches and activates the active subsystem
	Data flow layer	Layer that performs one calculation that cannot be divided

When applying layer concepts:

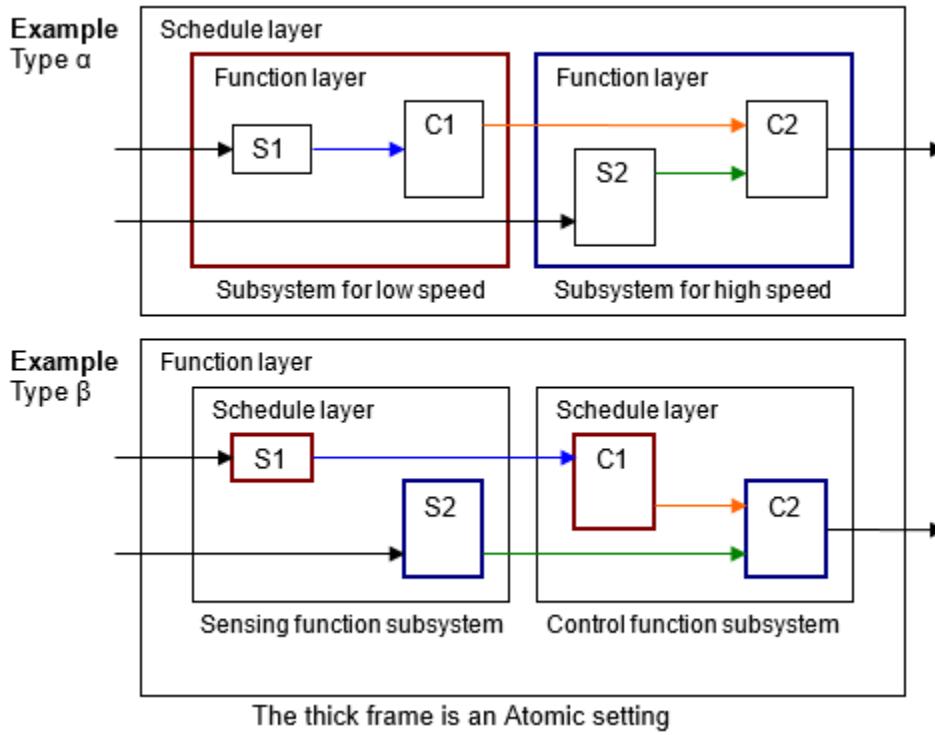
- Layer concepts shall be assigned to layers and subsystems shall be divided accordingly.
- When a layer concept is not needed, it does not need to be allocated to a layer.
- Multiple layer concepts can be allocated to one layer.

When building hierarchies, division into subsystems for the purpose of saving space within the layer shall be avoided.

Top Layer

Layout methods for the top layer include:

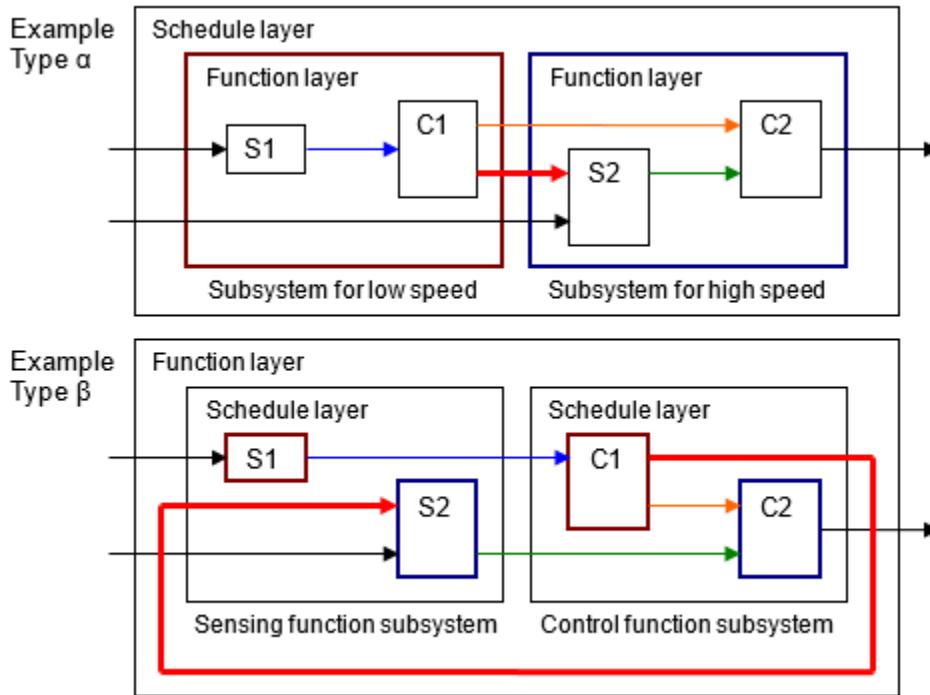
- Simple control model — Represents both the function layer and schedule layer in the same layer. Here, function is execution unit. For example, a control model has only one sampling cycle and all functions are arranged in execution order.
- Complex control model Type α — The schedule layer is positioned at the top. This method makes integration with the code easy, but functions are divided, and the readability of the model is impaired.
- Complex control model Type β — Function layers are arranged at the top and schedule layers are positioned below the individual function layers.



Function Layers and Sub-Function Layers

When modeling function and sub-function layers:

- Subsystems shall be divided by function, with the respective subsystems representing one function.
- One function is not always an execution unit so, for that reason, the respective subsystem is not necessarily an atomic subsystem. In the type β example below, it is more appropriate for a function layer subsystem to be a virtual subsystem. Algebraic loops are created when these change into atomic subsystems.
- Individual functional units shall be described.
- When the model includes multiple large functions, consider using model references for each function to partition the model.



Schedule Layers

When scheduling layers:

- System sampling intervals and execution priority shall be set. Use caution when setting multiple sampling intervals. In connected systems with varying sampling intervals, ensure that the system is split for each sampling interval. This minimizes the RAM needed to store previous values in the situation where the processing of signals values differs for fast cycles and slow cycles.
- Priority ranking shall be set. This is important when designing multiple, independent functions. When possible, computation sequence for all subsystems should be based on subsystem connections.
- Two different types of priority rankings shall be set, one for different sampling intervals and the other for identical sampling rates.

There are two types of methods that can be used for setting sampling intervals and priority rankings:

- For subsystems and blocks, set the block parameter **sample time** and block properties **priority**.
- When using conditional subsystems, set independent priority rankings to match the scheduler.

Patterns exist for many different conditions, such as the configuration parameters for custom sampling intervals, atomic subsystem settings, and the use of model references. The use of a specific pattern is closely linked to the code implementation method and varies significantly depending on the status of the project. Models that are typically affected include:

- Models that have multiple sampling intervals
- Models that have multiple independent functions
- Usage of model references

- Number of models (and whether there is more than one set of generated code)

For the generated code, affected factors include:

- Applicability of a real-time operating system
- Consistency of usable sampling intervals and computation cycles to be implemented
- Applicable area (application domain or basic software)
- Source code type: AUTOSAR compliant - not compliant - not supported.

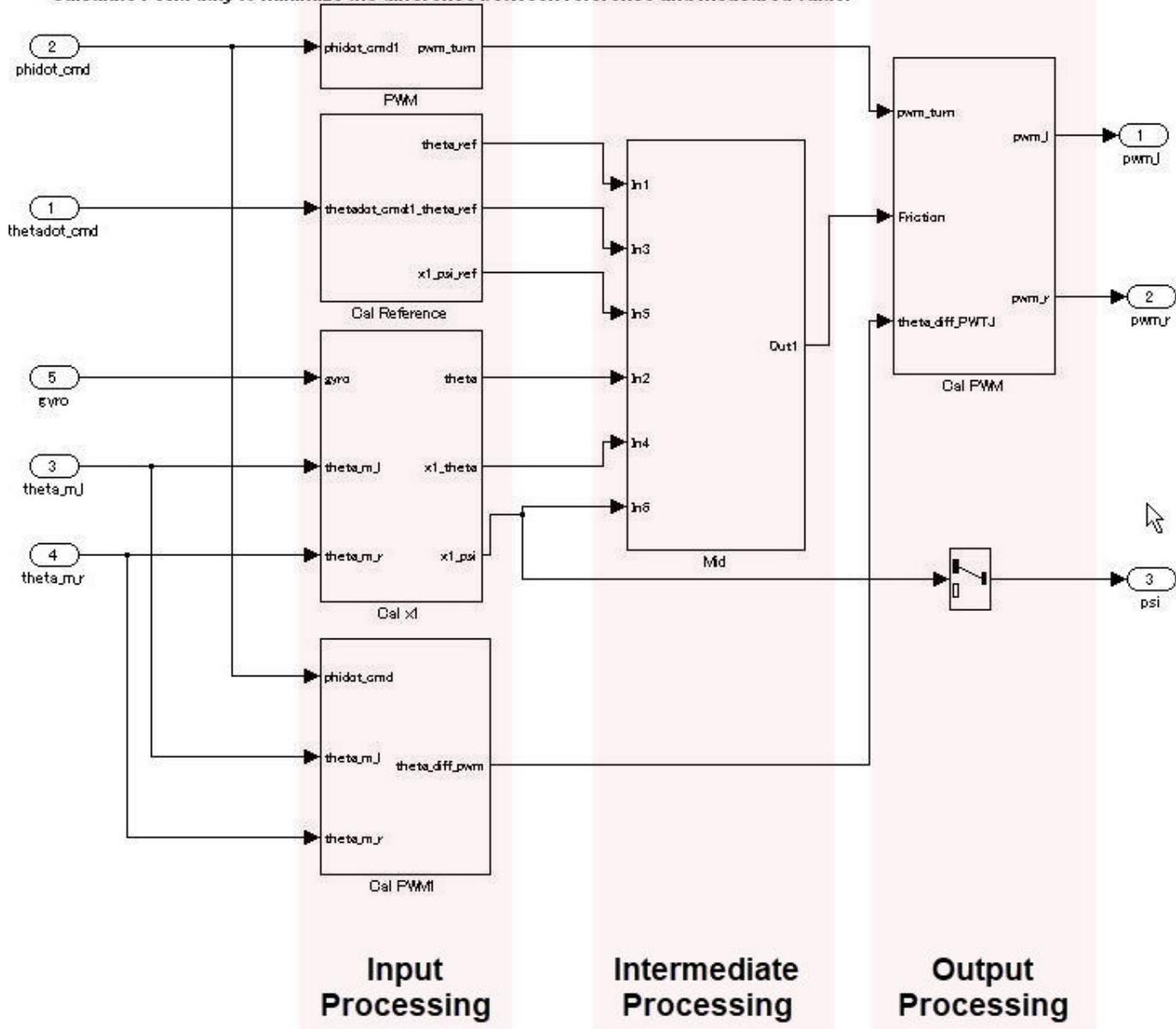
Control Flow Layers

In the hierarchy, the control layer expresses all input processing, intermediate processing, and output processing by using one function. The arrangement of blocks and subsystems is important in this layer. Multiple, mixed small functions should be grouped by dividing them between the three largest stages of input processing, intermediate processing and output processing, which forms the conceptual basis of control. The general configuration occurs close to the data flow layer and is represented in the horizontal line. The difference in a data flow layer is its construction from multiple subsystems and blocks.

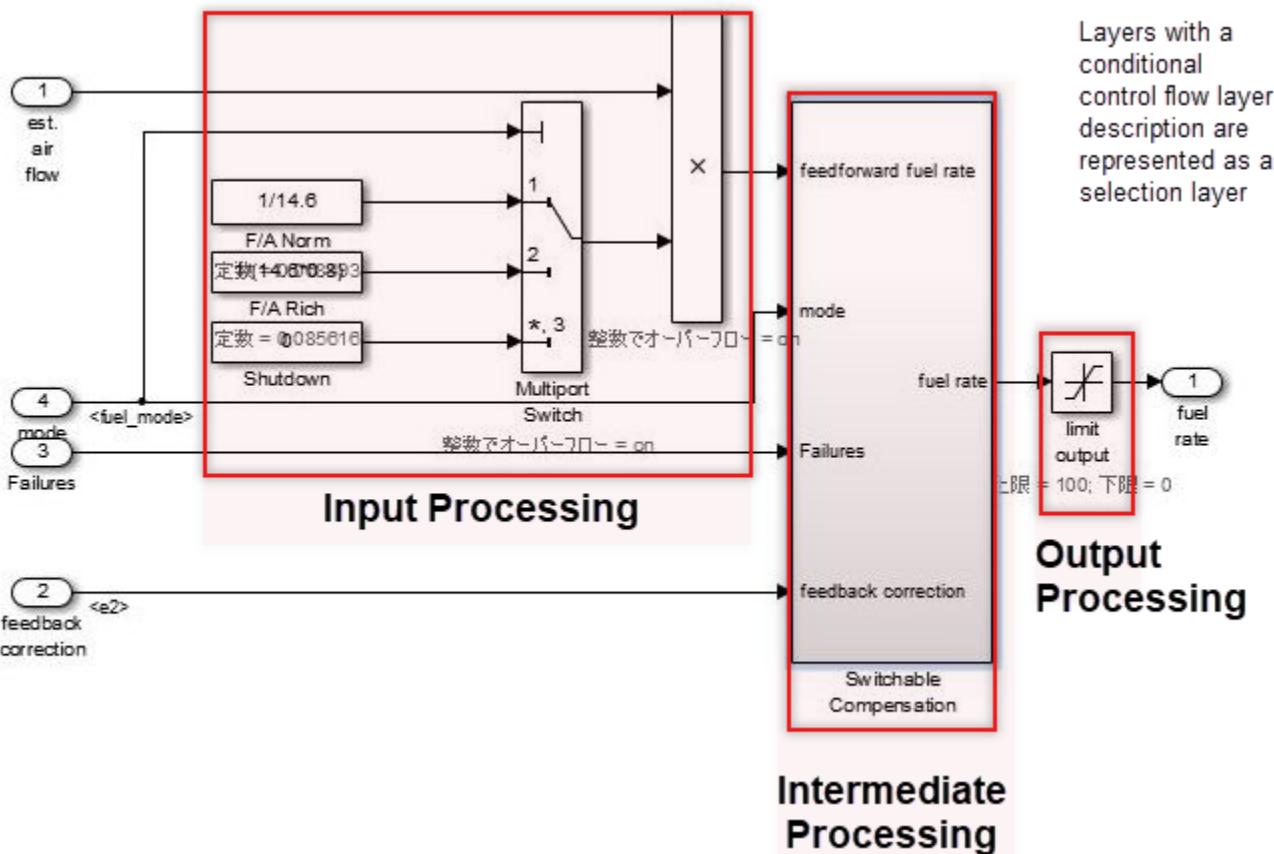
In control flow layers, the horizontal direction indicates processing with different significance; blocks with the same significance are arranged vertically.

NXTway-GS Controller

Calculate PWM duty to minimize the difference between reference and measured value.



Block groups are arranged horizontally and are given a provisional meaning. Red borders, which signify the delimiter for processing that is not visible, correspond to objects called virtual objects. Using annotations to mark the delimiters makes it easier to understand.



Control flow layers can co-exist with blocks that have a function. They are positioned between the sub-function layer and the data flow layer. Control flow layers are used when:

- The number of blocks becomes too large
- All is described in the data flow layer
- Units that can be given a minimum partial meaning are made into subsystems

Placement in the hierarchy organizes the internal layer configuration and makes it easier to understand. It also improves maintainability by avoiding the creation of unnecessary layers.

When the model consists solely of blocks and does not include a mix of subsystems, if the horizontal layout can be split into input/intermediate/output processing, it is considered a control flow layer.

Selection Layers

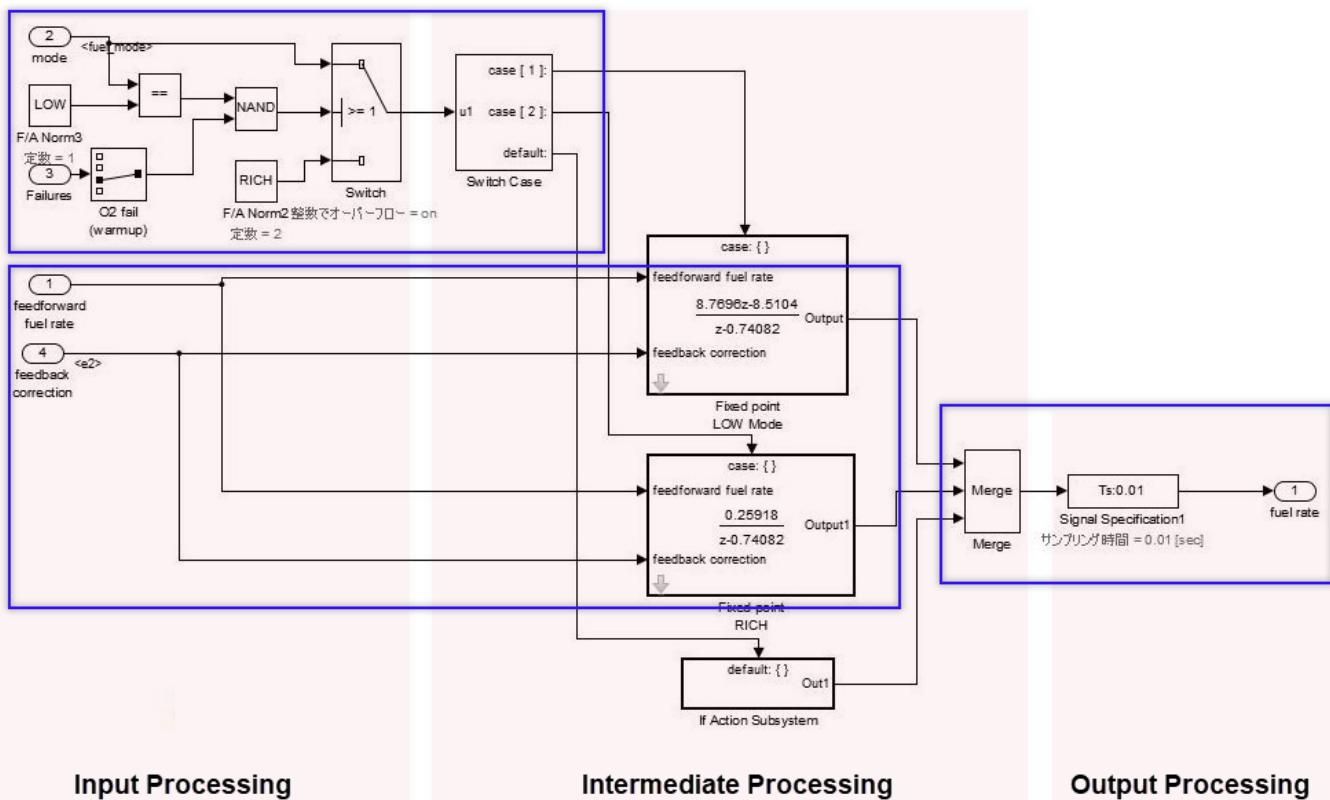
When modeling selection layers:

- Selection layers should be written vertically or side-by-side. There is no significance to which orientation is chosen.
- Selection layers shall mix with control flow layers.

When a subsystem has switch functions that allow only one subsystem to run depending on the conditional control flow inside the red border, it is referred to as a selection layer. It is also described as a control flow layer because it structures input processing/intermediate processing (conditional control flow)/output processing.

In the control flow layer, the horizontal direction indicates processing with different significance. Parallel processing with the same significance is structured vertically. In selection layers, no significance is attached to the horizontal or vertical direction, but they show layers where only one subsystem can run. For example:

- Switching coupled functions to run upwards or downwards, changing chronological order
- Switching the setting where the computation type switches after the first time (immediately after reset) and the second time
- Switching between destination A and destination B



Data Flow Layers

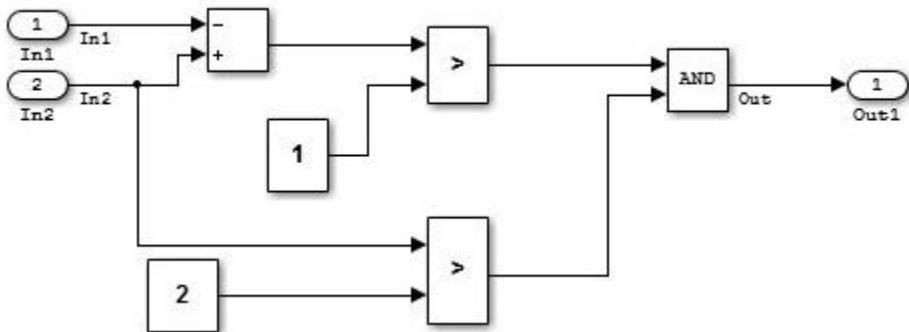
A data flow layer is the layer below the control flow layer and selection layer.

A data flow layer represents one function as a whole; input processing, intermediate processing and output processing are not divided. For instance, systems that perform one continuous computation that cannot be split.

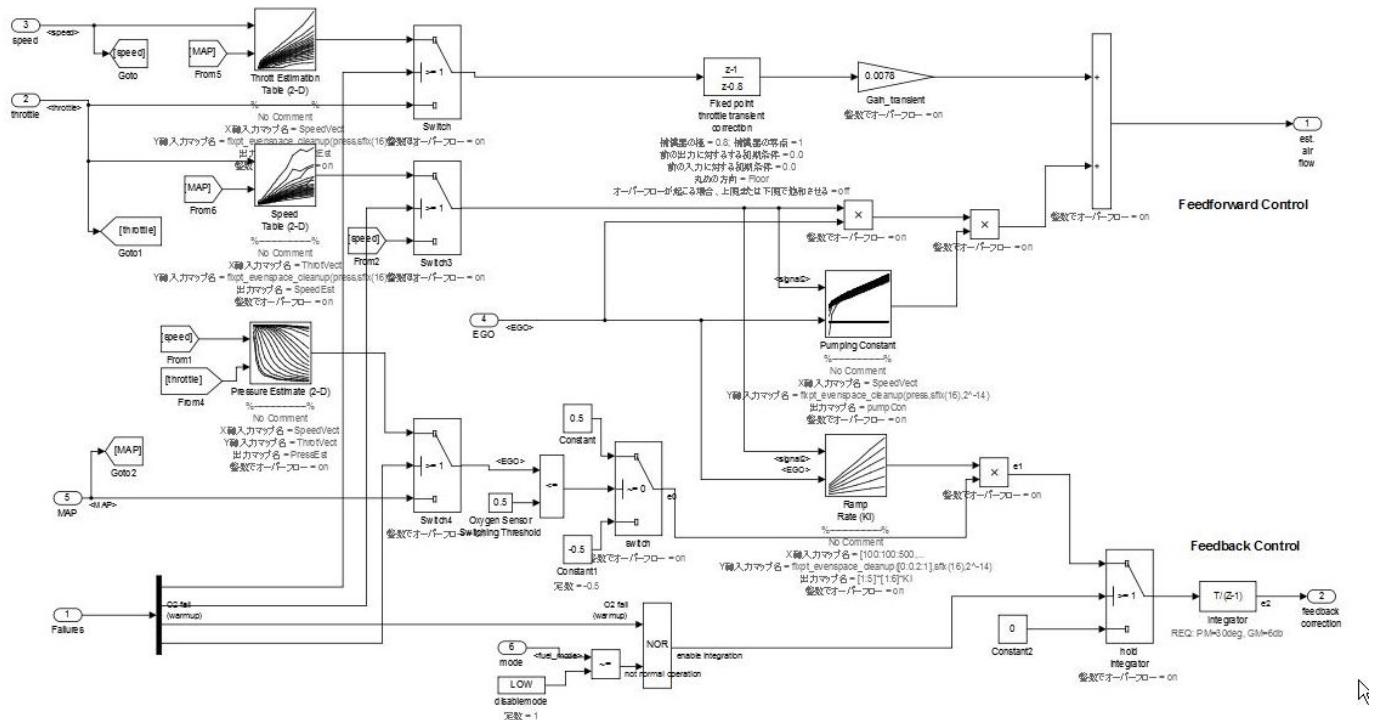
Data flow layers cannot coexist with subsystems apart from those where exclusion conditions apply. Exclusion conditions include:

- Subsystems where reusable functions are set
- Masked subsystems that are registered in the Simulink standard library
- Masked subsystems that are registered in a library by the user

Example of a simple data flow layer.



Example of a complex data flow layer.



When input processing and intermediate processing cannot be clearly divided as described above, they are represented as a data flow layer.

A data flow layer becomes complicated when both the feed forward reply and feedback reply from the same signal are computed at the same time. Even when the number of blocks in this type of cases is large, the creation of a subsystem should not be included in the design when the functions cannot be clearly divided. When meaning is attached through division, it should be designed as a control flow layer.

Relationship Between Simulink Models and Embedded Implementation

Running an actual micro controller requires embedding the code that is generated from the Simulink model into the micro controller. This requirement affects the configuration Simulink model and is dependent on:

- The extent to which the Simulink model will model the functions
- How the generated code is embedded
- The schedule settings on the embedded micro controller

The configuration is affected significantly when the tasks of the embedded micro controller differs from those modeled by Simulink.

Scheduler Settings in Embedded Software

The scheduler in embedded software has single-task and multi-task settings.

Single-task schedule settings

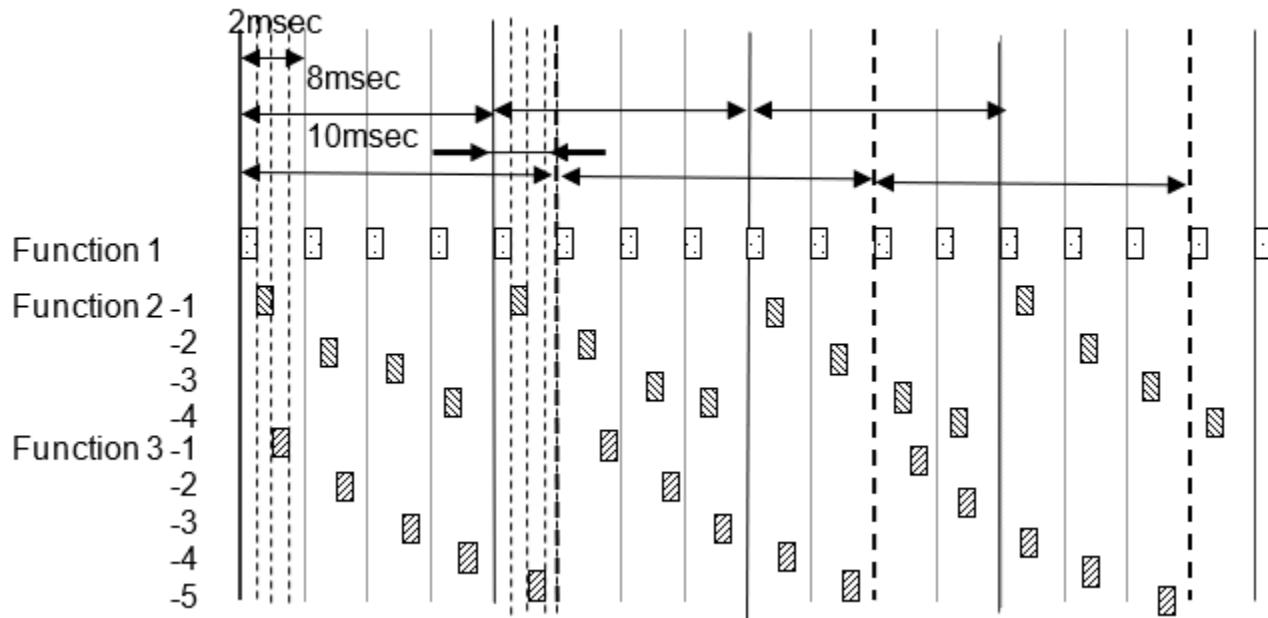
A single-task scheduler performs all processing by using basic sampling. Therefore, when processing of longer sampling is needed, the function is split so the CPU load is as evenly distributed as possible, and then processed using basic sampling. However, as equal splitting is not always possible, functions may not be able to be allocated to all cycles.

For example, basic sampling is 2 millisecond, and sampling rates of 2 millisecond, 8 millisecond and 10 millisecond exist within the model. An 8 millisecond function is executed once for every four 2 millisecond cycles, and a 10 millisecond function is executed once for every five. The number of executions is counted every 2 millisecond and the sampling function specified by this frequency is executed. Attention needs to be paid to the fact that the 2 millisecond, 8 millisecond and 10 millisecond cycles are all computed with the same 2 millisecond. Because all computations need to be completed within 2 millisecond, the 8 millisecond and 10 millisecond functions are split into several and adjusted so that all 2 millisecond computations are of an almost equal volume.

The following diagram shows the 8 millisecond function split into 4 and the 10 millisecond function split into 5.

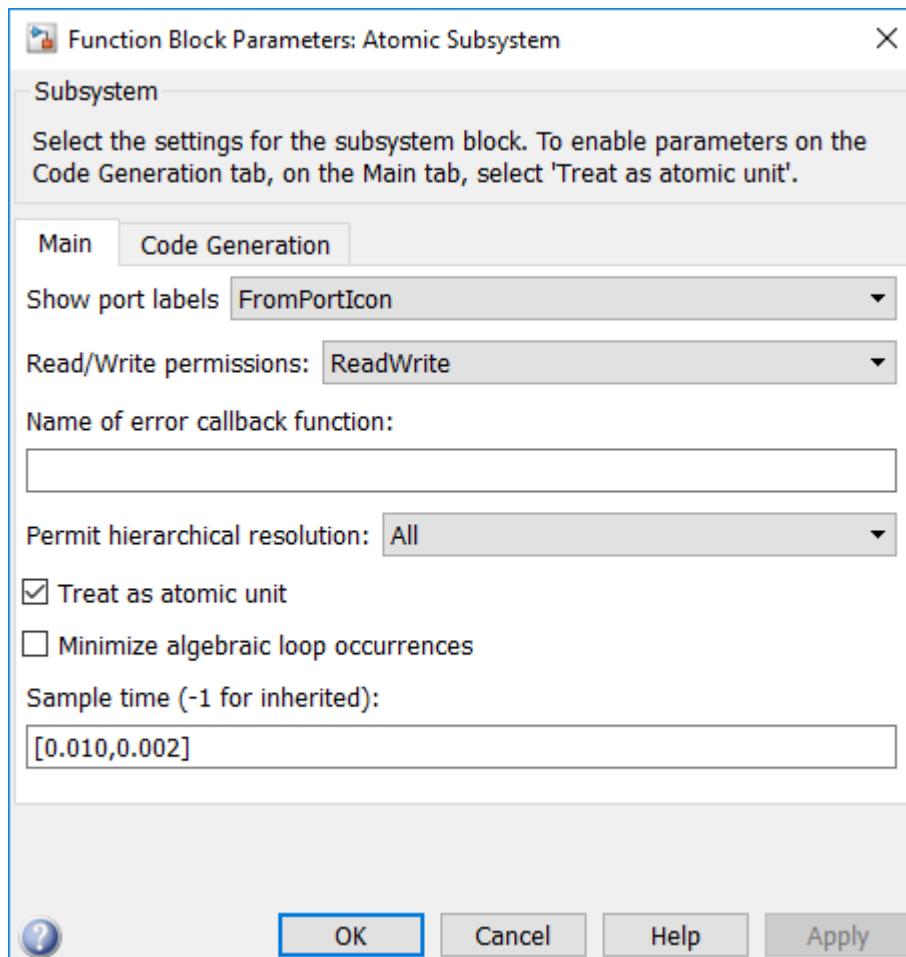
Functions	Fundamental frequency	Offset
	8millisecond	0millisecond
2-2	8millisecond	2millisecond
2-3	8millisecond	4millisecond
2-4	8millisecond	6millisecond
3-1	10millisecond	0millisecond
3-2	10millisecond	2millisecond
3-3	10millisecond	4millisecond
3-4	10millisecond	6millisecond
3-5	10millisecond	8millisecond

All computations must be contained within the 2 msec cycle



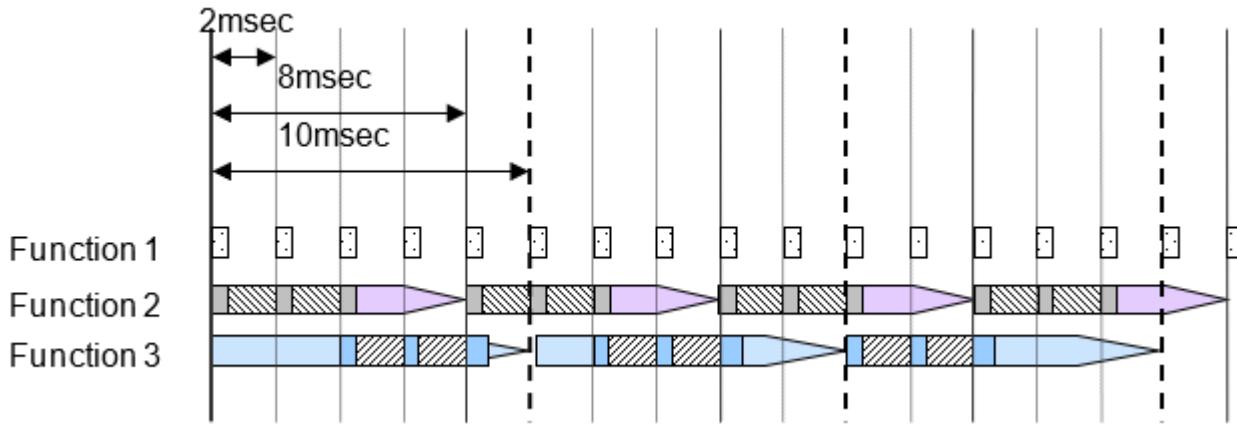
To set frequency-divided tasking:

- 1 Clear configuration parameter Treat each discrete rate as a separate task.
- 2 For the Atomic Subsystem block parameter **Sample time**, enter the sampling period offset values. A subsystem for which a sampling period can be specified is referred to as an atomic subsystem.



Multi-task scheduler settings

Multi-task sampling is executed by using a real-time OS that supports multi-task sampling. In single-task sampling, equalizing the CPU load is not done automatically, but a person divides the functions and allocates them to the appointed task. In multi-task sampling, the CPU performs the computations automatically in line with the current status; there is no need to set detailed settings. Computations are performed and results are output starting from the task with the highest priority, but the task priorities are user-specified. Typically, fast tasks are assigned highest priority. The execution order for this task is user-specified.



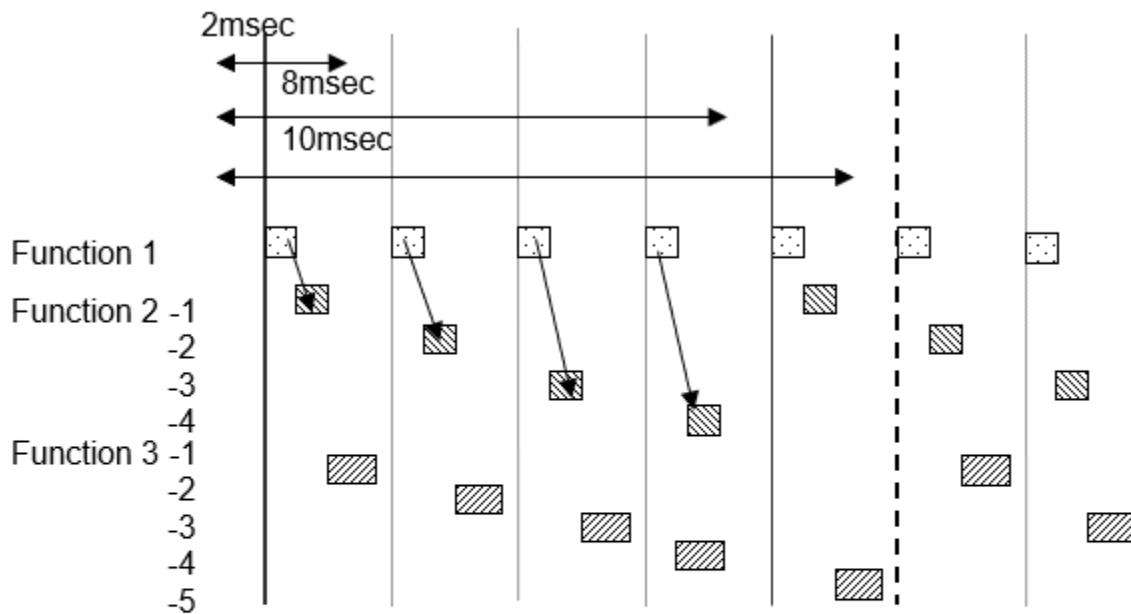
It is important that computations are completed within the cycle, including slow tasks. When the processing of a high priority computation finishes and the CPU is available, the computation for the system with the next priority ranking begins. A high priority computation process can interrupt a low priority computation, which is then aborted so the high priority computation process can execute first.

Effect of Connecting Subsystems with Sampling Differences

If subsystem B with a 20 millisecond sampling interval uses the output of subsystem A with a 10 millisecond sampling interval, the output result of subsystem A can change while subsystem B is computing. If the values change partway through, the results of subsystem B's computation may not be as expected. For example, a comparison is made in subsystem B's first computation with the subsystem A output, and the result is computed with the conditional judgment based on this output. At this point, the comparison result is true. It is then compared again at the end of subsystem B; if the output from A is different, then the result of the comparison can be false. Generally, in this type of function development it may happen that the logic created with true, true has become true, false, and an unexpected computation result is generated. To avoid this type of malfunction, when there is a change in task, output results from subsystem A are fixed immediately before they are used by subsystem B as they are used in a different RAM from that used by the subsystem A output signals. In other words, even if subsystem A values change during the process, the values that subsystem B are looking at is in a different RAM, so no effect is apparent.

When a model is created in Simulink and a subsystem is connected that has a different sampling interval in Simulink, Simulink automatically reserves the required RAM.

However, if input values are obtained with a different sampling interval through integration with hand-coded code, the engineer who does the embedding work should design these settings. For example, in the RTW concept using AUTOSAR, different RAMs are all defined at the receiving and exporting side.



If Function 2 uses the computation results of Function 1, computation results for Function 1 do not change during computation for Functions 2-1, 2-2, 2-3, but there is a possibility that Functions 2-1, 2-2, 2-3 use different values that have been computed on the respective different time axes.

Allocate different RAM for signal values with a different rates

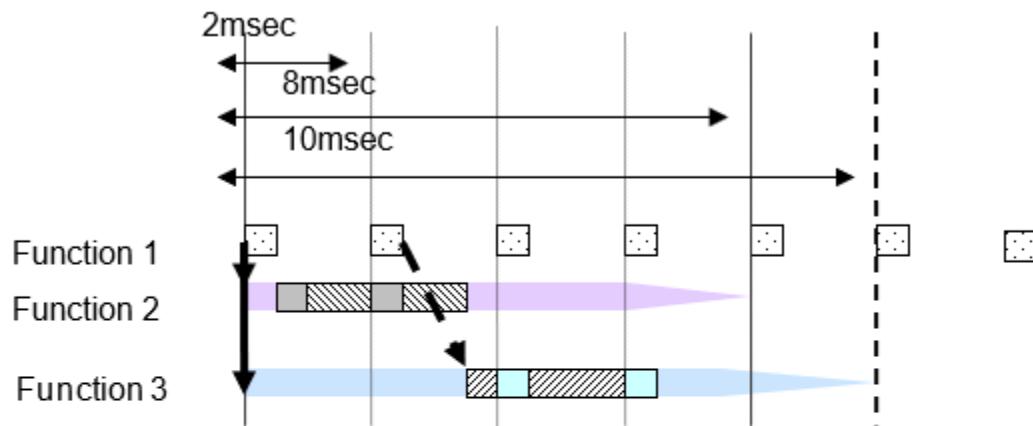
Single-task scheduler settings

Signal values are the same within the same 2 millisecond cycle, but when there are different 2 millisecond cycles, the computation value differs from the preceding one. When Function 2-1 and 2-2 uses signal A of Function 1, be aware that 2-1 and 2-2 uses results from different times.

Multi-task scheduler settings

For multi-task, you cannot specify at what point to use the computation result to use. With multi-task, always store signals for different tasks in new RAM.

Before new computations are performed within the task, all values are copied.



Do not immediately use values that are being updated.

The value should be held at the beginning of the task.

If Function 2 uses the computation results of Function 1, it is possible that computation results from Function 1 will replace them while Function 2 is computing. For that reason, computation results that vary at the point when computation starts for each sampling are typically stored in a different RAM.

Using Simulink and Stateflow in Modeling

When using Stateflow, Simulink is required for inputs, outputs, and structuring. Stateflow alone can perform a variety of formula processing. When using Simulink, complex state variables can be realized through methods, such as using the Switch Case block.

Either Simulink or Stateflow can be used to model specific parts of control, however, the application of either product in the development workflow is based on the user's understanding of the underlying algorithms and, ultimately, comes down to the organization to determine which tool is best suited for their needs. Determining whether Simulink or Stateflow should be used for design should be determined by a group of people in accordance with the task. Whether implementation in Stateflow is done by using state transitions or with flow charts should also be specified.

In most cases, Stateflow is less efficient with regards to RAM. Therefore, Simulink has an advantage in computations that use simple formulas. In addition, Simulink is more advantageous for situations where state variables are operated with simple flip-flops and the Relay block. When evaluating whether to use Simulink or Stateflow in a project, these topics should be taken into consideration:

- Increasing RAM: There must always be a RAM available for visualization of Stateflow inputs, outputs and internal variables.
- Equation error handling: When general computational formulas are used internally, the user designs ways to prevent overflow.
- Splitting and separating functions: When performing calculations that use Simulink outside of Stateflow, there is a possibility that they may split, thus reducing readability. There are also times where readability may improve. This can be difficult to judge.

There are cases where Stateflow has more efficient code than Simulink for optimum expressions that are close to code, but most of these result in a model that is difficult to understand. If code already exists, it is more advantageous to use S-functions instead of Stateflow modelling. Stateflow can note computations where specific arrangements are specified, or computations using for-loops, more efficiently than Simulink, but in recent years it has also become convenient to use MATLAB language for descriptions. If needed, consider using MATLAB language for modelling.

For Stateflow models, when dealing with states as described below, readability improves by describing them as state transitions:

- Different output values are output for identical inputs.
- Multiple states exist (as a guide, three or more).
- States with meaningful names instead of just numbers.
- Inside a state, initialization (first time) and differentiation during execution (after the second time) is required.

For instance, in flip-flop circuits, different values are outputted for inputs. State variables are limited to 0 and 1. However, a meaningful name cannot be added to each state simply by retaining boolean type numbers. There is also no distinction between initialization and execution within the state. Thus, only one flip-flop applies out of the four above, so Simulink is more beneficial.

In Stateflow, situations that can be represented as states are implemented as state transitions and conditional branches that are not states are implemented as flow charts. Truth tables are classified as a conditional branch implementation method. When designing states as state transitions by using Stateflow, **Classic** should be selected as the state machine type so that it is implemented as software into the control system's embedded micro controller.

HDL Coder™ is supported by Stateflow. When using HDL Coder, Mealy or Moore must be selected; Moore mode is more appropriate when protection is required against internal electric leaks.

Note HDL Coder use cases are not described in these guidelines.

Simulink Functionality

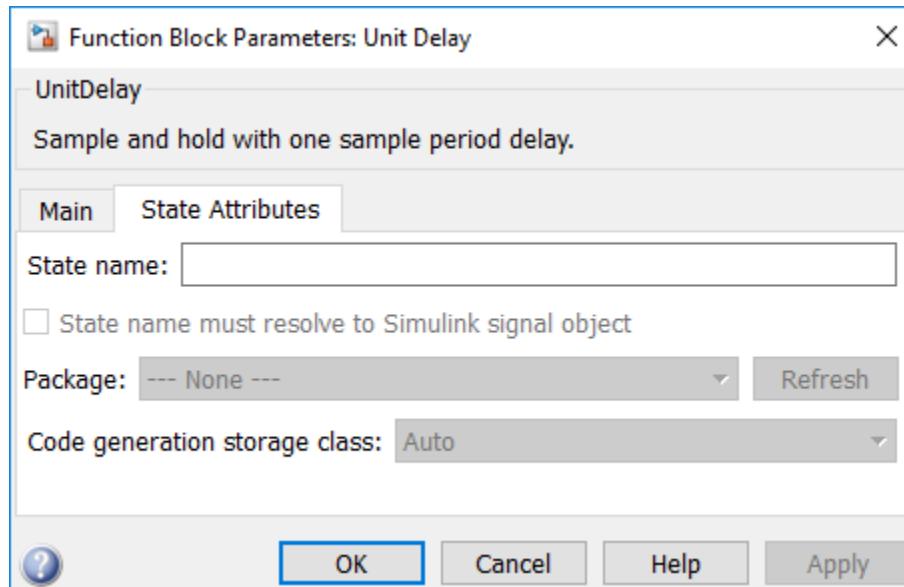
This section provides information about using Simulink for modeling.

Blocks with State Variables

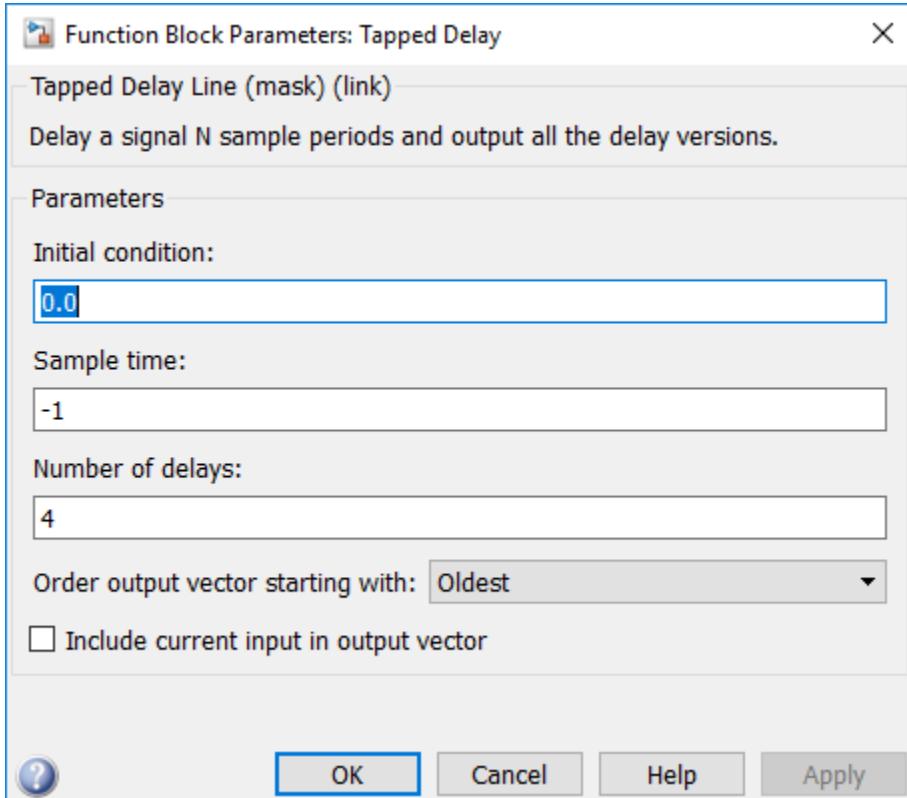
Blocks with state variables are primarily grouped into Simulink and discrete types.

For most of these blocks, the user can set the state attributes and initial values by using the block parameters. A conditional subsystem can have state variables, depending on the structure pattern.

In this example, the Unit Delay block has state attributes.



In this example, the Tapped Delay block does not have state attributes.

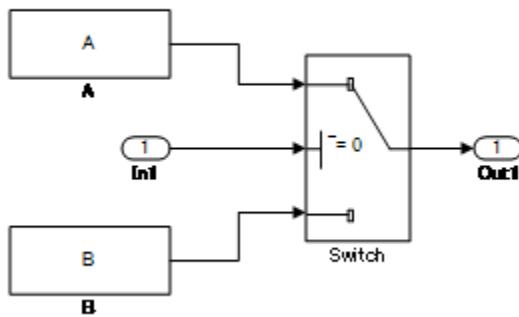


See guideline: jc_0640

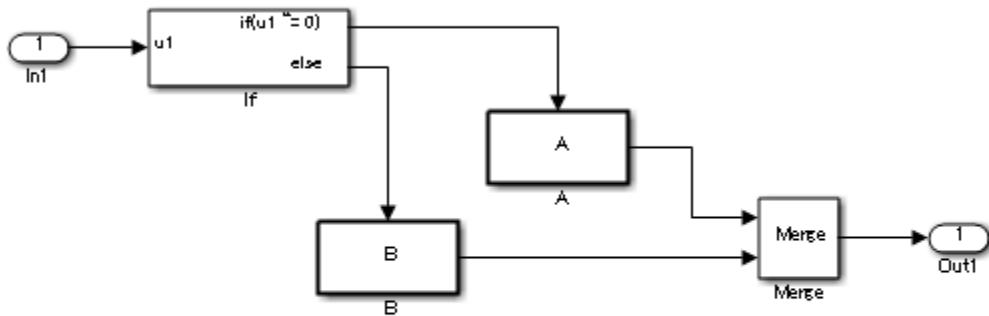
Branch Syntax with State Variables

The Switch block and conditional subsystems behave differently when state variables are used.

Depending on the configuration setting, when any state variable exists, the Switch block generally executes subsystem A when the condition of the control port is satisfied. If the condition is not satisfied, it executes only subsystem B without calculating subsystem A. However, when the subsystem A contains a state variable, calculation for the state variable within the subsystem A is processed even when the conditions of the control port are not satisfied.



In the conditional subsystem, subsystem A is calculated when the condition is satisfied. When it is not satisfied, subsystem B is calculated instead of subsystem A, regardless of the existence of any state variables in subsystem A.



The reset action in a recalculation can be specified by using the {Action Port} setting.

The behavior of subsystem A when using a Switch block and a conditional control flow is listed in the following tables. Familiarize yourself with these behaviors to determine which structure, the Switch block, or conditional subsystem is most suitable for the intended purpose.

This table shows the behavior of subsystem A.

Control port condition State variables	(in subsystem A)	Switch	Conditional subsystem
Hold	No	Executed	Executed
	Yes		
Not hold	No	Not executed	Not executed
	Yes	Minimally-processed *Executed calculations related to the state variables	

This table provides the initialization timing of subsystem A.

Action Port	Initialize

Switch	—	First time only
Conditional subsystem	Hold	First time only
	Reset	At returned by condition

See guidelines:

- jc_0656: Usage of Conditional Control blocks
- jc_0657: Retention of output value based on conditional control flow blocks and Merge blocks

Subsystems

A subsystem is used for compiling various blocks and subsystems.

Subsystems can also be used for other purposes. Usage methods that are not functional subsystems include:

- Mask display of the subsystem is used to describe the outline or display fixed form documents, such as "classified"
- The open functions (callback functions in the block properties) of the subsystem is used for running several tools or displaying explanatory text separate from the model
- Subsystems whose setting have changed to a mask subsystem (a subsystem that was simply set to NoReadOrWrite) by a user with administrative rights to make a change, but other users cannot see the content.

These non-typical subsystems are outside of the scope of the guidelines and, if excluded, should be put on an exclusion list managed within the project.

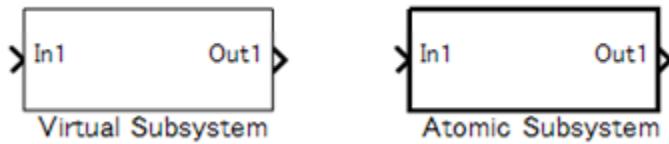
See guidelines:

- jc_0201: Usable characters for subsystem names
- jc_0243: Length restriction for subsystem names
- db_0143: Usable block types in model hierarchy
- db_0144: Use of subsystems
- db_0141: Signal flow in Simulink models
- jc_0653: Delay block layout in feedback loops
- jc_0171: Clarification of connections between structural subsystems
- jc_0602: Consistency in model element names
- jc_0081: Import and Outport block icon display
- db_0081: Unconnected signals and blocks

Atomic Subsystems and Virtual Subsystems

There are two types of subsystems: Virtual subsystems and Atomic subsystems. The primary difference between these subsystems is whether the subsystem is treated as a single execution unit. The virtual subsystem is the default subsystem block.

In a model, the border for a Virtual subsystem is thin as compared the border for the Atomic subsystem, which is thick and bold.

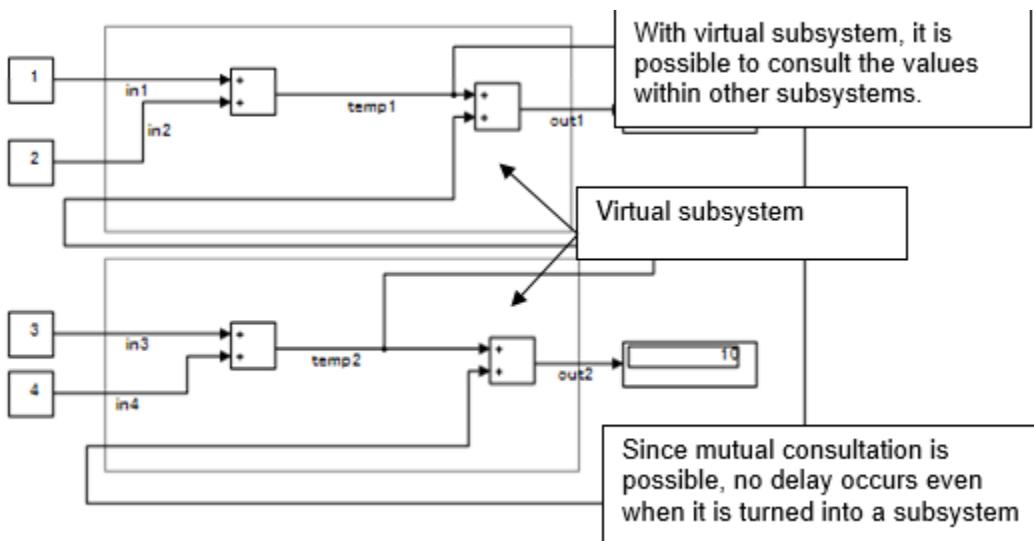


Virtual Subsystems

A block that provides a visual representation is known as a "virtual block". For example, a Mux block that compiles several signal lines, a From block that hands out the signal, and a Goto block that corresponds to a virtual block. Since the subsystem block in the default setting only constitutes a visual hierarchical structure, these blocks are considered virtual blocks. The subsystem is referred to as a virtual subsystem.

Consider a subsystem that consults an external calculation result within a subsystem, as shown in the following example. This system is calculated from these four equations.

- $\text{temp1} = \text{in1} + \text{in2}$
- $\text{temp2} = \text{in3} + \text{in4}$
- $\text{out1} = \text{in1} + \text{in2} + \text{temp2}$
- $\text{out2} = \text{temp1} + \text{in3} + \text{in4}$



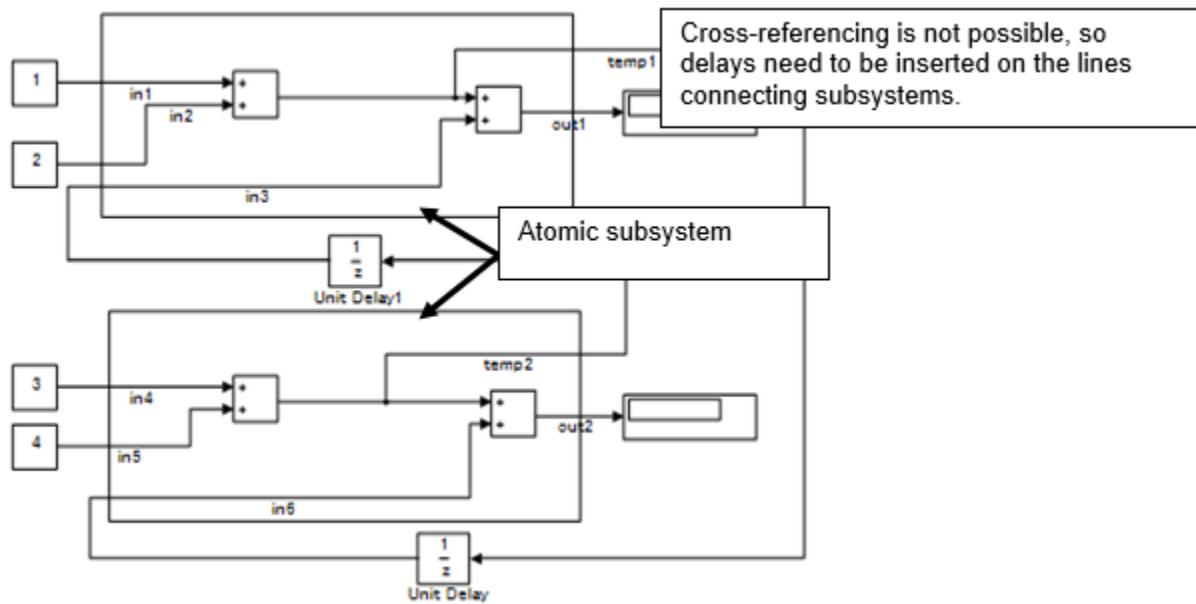
Atomic Subsystems

An atomic subsystem is detached from the external system and is not subject to cross-border optimization. Atomic subsystems do not use the results of the internal calculations of each subsystem. Therefore, interim output value will use a calculation result that is delayed by a session.

- $\text{temp1} = \text{in1} + \text{in2}$
- $\text{temp2} = \text{in4} + \text{in5}$
- $\text{out1} = \text{in1} + \text{in2} + \text{in3}$
- $\text{out2} = \text{in4} + \text{in5} + \text{in6}$

- $in3 = temp2$
- $in6 = temp1$

Atomic subsystems prohibit the direct referencing of the interim calculation results to other subsystems.



Notes on atomic subsystems:

- Atomic subsystems can select C-source function settings.
- As explained above, the internal section of an atomic subsystem will become encapsulated (objectified).
- Depending on the relationship before and after, a static RAM section should be secured inside the subsystem for the output signal.
- Atomic subsystems (including the addition of function settings) should be used with caution. Factor setting will not simply have a factor name inserted within a C code. It should be acknowledged that it is described as a mathematically independent system and the conditions under which an atomic subsystem can be used should be reviewed.
- Include the relationship with the structure layer; it is necessary to determine an operation rule per project and to determine its relationship with the guideline rules.

Signal Name

Signals can be named and are referred to as signal names. When a signal is named, that signal name is displayed as a label. Updates to labels are reflected in the signal name and are also displayed.

The signal name can be propagated to a signal line via a branched signal line or port block and displayed as a signal name.

See guidelines:

- jc_0222: Usable characters for signal and bus names

- jc_0245: Length restriction for signal and bus names

Code can be generated by associating a signal name with a signal object (Simulink object or mpt object). Type setting is configured through the data dictionary, setting of the storage class is optional. The recommended data type settings for these blocks include:

- For Import blocks, set **data type** to auto.
- For Outport blocks, **data type** to auto.
- For Sum blocks, set the output **data type** to Inherit via back propagation.

See guideline jc_0644: Type setting.

Vector Signals/Path Signal

Individual scalar signals that compose a vector shall have common functions, data type, and units.

Signals that do not fulfill the conditions as a vector can only be grouped as a bus signal. The Bus Selector block shall be used only with bus signal inputs. It shall not be used to extract a scalar signal from a vector signal.

The following table is an example of a vector signal.

Types of vector	Size
Row vector	[1 n]
Column vector	[n 1]
Wheel speed subsystem	[1 wheel number]
Cylinder vector	[1 cylinder number]
Location vector based on a 2-dimensional coordination points	[1 2]
Location vector based on 3-dimensional coordination points	[1 3]

The following table is an example of a bus signal.

Bus type	Factor
Sensor bus	Force vectors
	Location
	Wheel speed vector [Θ_{lf} , Θ_{rf} , Θ_{lr} , Θ_{rr}]
	Acceleration
	Pressure
Controller bus	Sensor bus
	Actuator bus
Serial data bus	Circulating water temperature
	Engine speed, front passenger seat door open

See guidelines:

- na_0010: Usage of vector and bus signals
- jc_0222: Usable characters for signal and bus names
- jc_0245: Length restriction for signal and bus names
- db_0097: Position of labels for signals and buses
- jc_0630: Usage of Multiport Switch blocks
- jc_0659: Usage restrictions of signal lines input to Merge blocks

Enumerated Types

Enumerated type data refers to data that is restricted to a determined numerical value.

The type of blocks that can be used in an enumerated type in Simulink is limited.

To use an enumerated type, you must define the enumerate type by using .m file on MATLAB. For additional information about defining enumeration data types, see “Use Enumerated Data in Simulink Models”.

Stateflow Functionality

This section provides information about using Stateflow for modeling.

Operations Available for Stateflow

For additional information about the Stateflow operations, see “Operations for Stateflow Data” (Stateflow).

See guidelines:

- na_0001: Standard usage of Stateflow operators
- jc_0655: Prohibition of logical value comparison in Stateflow

Differences Between State Transition and Flow Chart

Stateflow can represent both a state transition and a flow chart.

Stateflow allows a flow chart to be designed within a state transition diagram.

An entry action is represented as a flow chart in a state, which starts from a default transition and moves to junctions through transition lines, as illustrated below. Starting from an internal transition line allows a *during* action to be represented in the flow chart.

A flow chart cannot maintain its active state between updates. As a result, a flow chart always ends at a terminating junction (a connective junction that has no valid outgoing transitions).

In contrast, a state transition diagram stores its current state in memory to preserve local data and active state between updates. As a result, state transition diagrams can begin executing where they left off in the previous time step. This means that state transitions are suitable for modeling reactive or supervisory systems that depend on history.

This table defines the start and end points for a flow chart and state transition diagram.

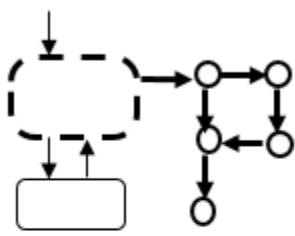
	Start point	End point
--	-------------	-----------

Flow chart	Default transition	All terminations from the state are connected to the connective junction.
State transition diagram	Default transition	Either termination should be connected to the state

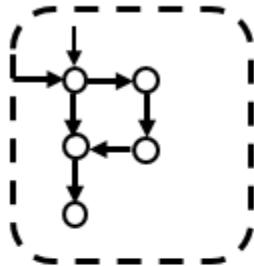
This illustration shows the difference between a general flow chart and state transition diagram

Flow Chart

Flow chart outside a state

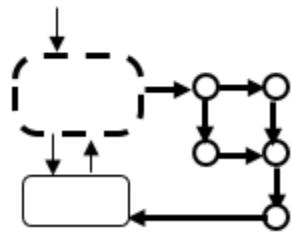


Flow chart inside a state

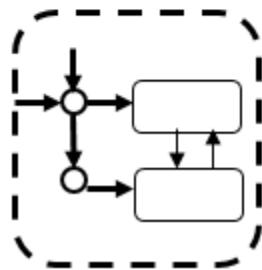


State Transition Diagram

State transition outside a state



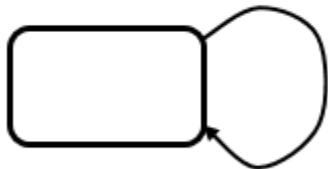
State transition inside a state



Mixture of flow charts and state transition diagrams with self-transition has more strict constraints.

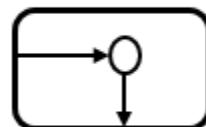
State Transition Diagram

Self transition outside a state



A self transition is formed outside a state and then reset after execution.

Self transition inside a state



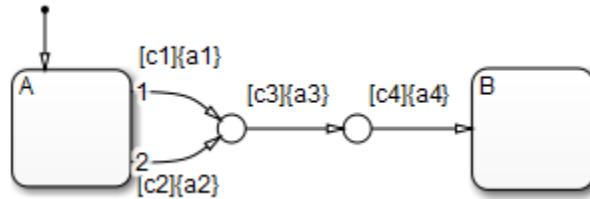
A self transition is formed inside a state and then reset using a during action.

See guidelines:

- db_0132: Transitions in flow charts
- jc_0752: Condition action in transition label

Backtrack

This example shows the behavior of transitions with junctions that force backtracking behavior in flow charts. The chart uses implicit ordering of outgoing transitions.



Initially, state A is active and transition conditions c1, c2, and c3 are true. Transition conditions c4 is false.

- 1 The chart root checks to see if there is a valid transition from state A.

There is a valid transition segment marked with the transition condition c1 from state A to a connective junction, therefore:

- a Transition condition c1 is true, so action a1 executes.
- b Transition condition c3 is true, so action a3 executes.
- c Transition condition c4 is not true and, therefore, the control flow backtracks to state A.

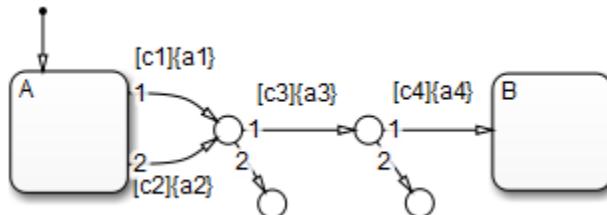
- 2 The chart root checks to see if there is another valid transition from state A.

There is a valid transition segment marked with the transition condition c2 from state A to a connective junction, therefore:

- a Transition condition c2 is true, so action a2 executes.
- b Transition condition c3 is true, so action a3 executes.

- c Transition condition c4 is not true and, therefore, the control flow backtracks to state A.
- 3 The chart goes to sleep.

To resolve this issue, consider adding unconditional transition lines to terminating junctions. The terminating junctions allow flow to end if either c3 or c4 is not true. This design leaves state A active without executing unnecessary actions.



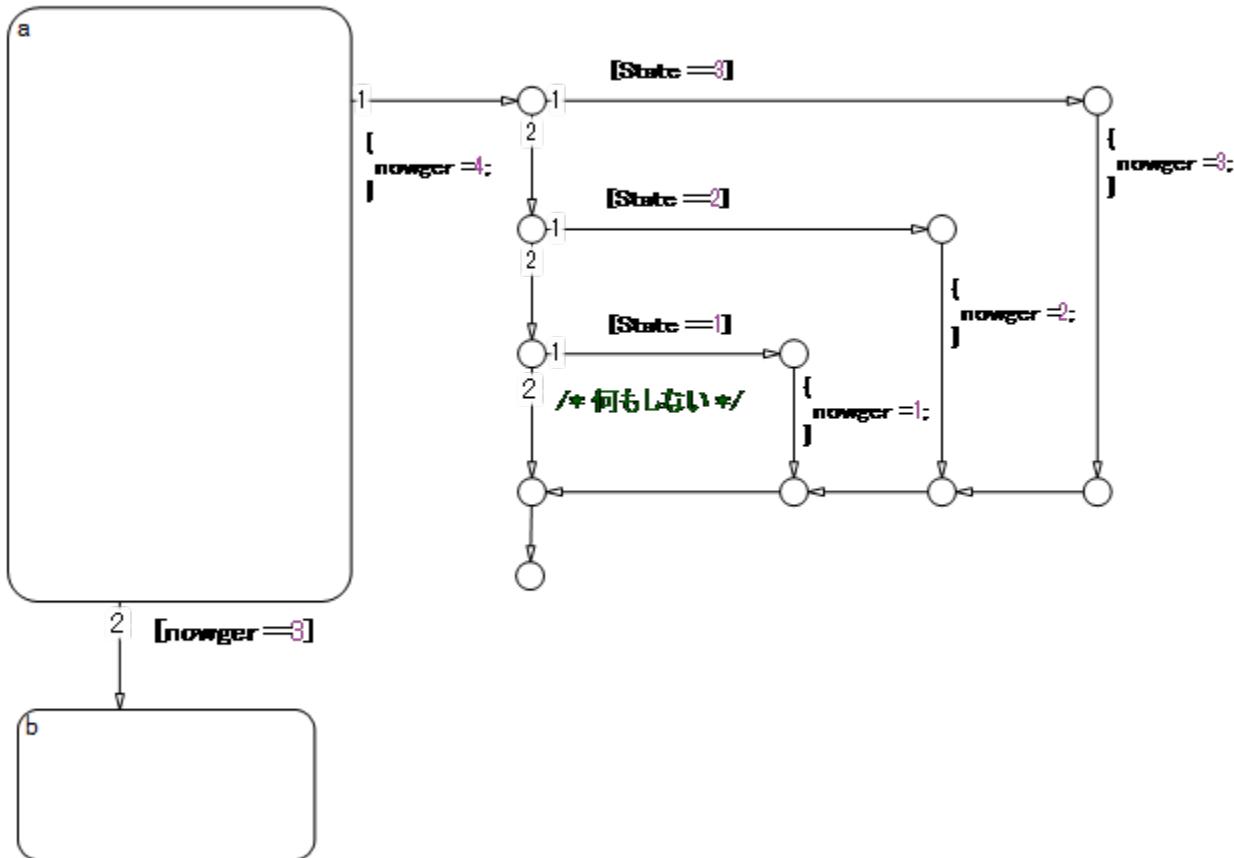
See guidelines:

- jc_0751: Backtracking prevention in state transition
- jc_0773: Unconditional transition of a flow chart

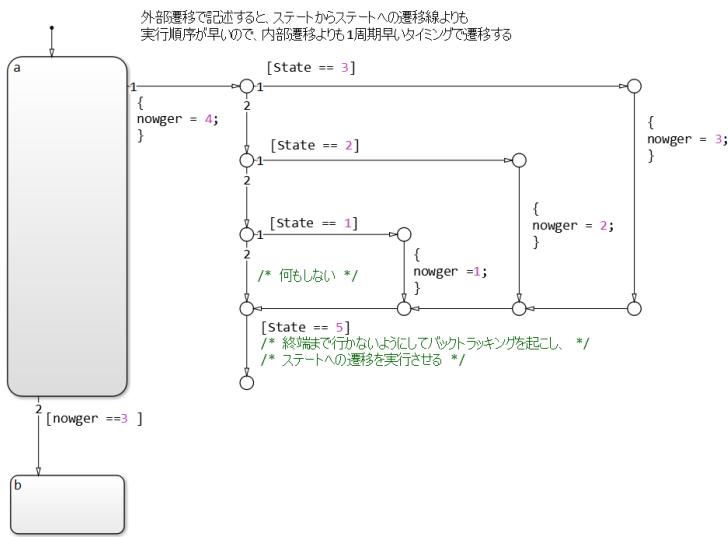
Flow Chart Outside the State

A flow chart associated with a state can be written inside or outside of the state; however, be attentive to the execution order and backtracking.

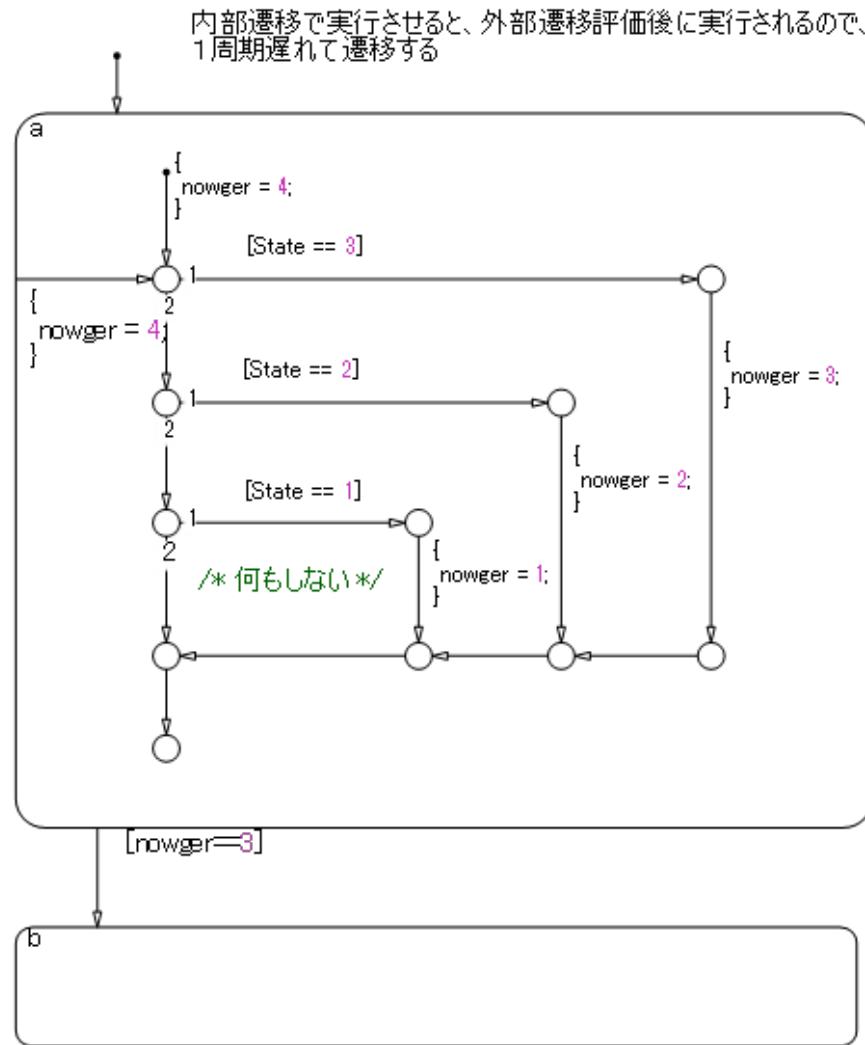
The following flow chart, which evaluates transition from a to b after executing the flow chart outside the state, appears to execute the transition within the same period as that of a newer calculation. However, the transition line to b is not evaluated if the termination point is reached by calculating the transition outside the state. This is a state transition diagram which always stays at a.



Done correctly, as shown below, the transition condition is not positioned at the termination of the external flow chart, allowing the transition line from a to b to be evaluated after the flow chart is executed. This enables the external flow chart to execute before the transition, and to be evaluated using the most recent value at the instant of the transition. Note that this chart contains a dead path where the transition condition will never hold, which can cause an error when the specification is changed in the future. Use this chart structure with caution.



In contrast, the following flow chart is inside a state, which means that the internal flow chart is always calculated when executing state a and can be described as an easily comprehensible structure without dead paths. However, it should be noted that, as a performance characteristic, when state a is executed, the transition from a to b is evaluated in the cycle following that in which the internal flow chart is calculated. Due to this characteristic, the timing of the execution of calculations and transitions for the external flow chart may be off. Use with caution.



See guidelines:

- jc_0751: Backtracking prevention in state transition
- jc_0773: Unconditional transition of a flow chart

Pointer Variables

This code sample is from model **sf_custom**. To open the model, enter the following on the MATLAB command line:

```
openExample('sf_custom')
```

In model **sf_custom**, click **Open my_header.h**.

```
#include "tmwtypes.h"

extern real_T my_function(real_T x);

/* Definition of custom type */
```

```

typedef struct {
    real_T a;
    int8_T b[10];
}MyStruct;

/* External declaration of a global struct variable */
extern MyStruct gMyStructVar;
extern MyStruct *gMyStructPointerVar;

```

In model sf_custom, click **Open my_function.c**.

```

#include "my_header.h"
#include <stdio.h>

/* Definition of global struct var */
MyStruct gMyStructVar;
MyStruct *gMyStructPointerVar=NULL;

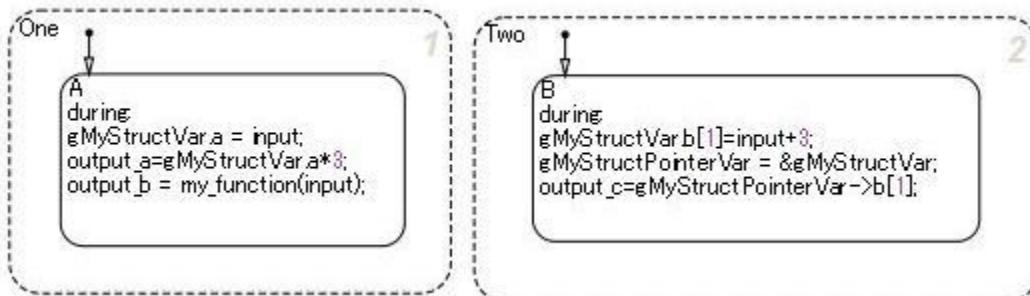
real_T my_function(real_T x)
{
    real_T y;

    y=2*x;

    return(y);
}

```

`gMyStructVar` is not defined in Stateflow. Typically, functions of `my_function` are called from C source for use in Stateflow. However, direct reference to global variables exposed by the C source is also available from Stateflow.



Initialization

This section provides information about using initialization values.

Initial Value Setting in Initialization

When a signal needs to be initialized, the initial values shall be set correctly.

When initial values are set inside a block, use an initial value list that includes annotations so you can visually confirm the initial values input.

Cases that require initial values include:

- When state variables are defined AND blocks that have state variables are used.
 - Use the internal block settings.
 - Use the external input values.
- When state variables are defined AND initial values are enabled for a block when a specific configuration is performed.
 - Set initial values in Merge blocks.
 - Use signals registered in the data dictionary.
- When signal settings (with RAM) have been defined that can be referenced from the outside.
 - Use signals registered in the data dictionary.

Initial Values of Signals Registered in the Data Dictionary

Set initial values for signals registered in the data dictionary.

- Discrete block groups, such as Unit Delay and Data Store Memory, have state variables.

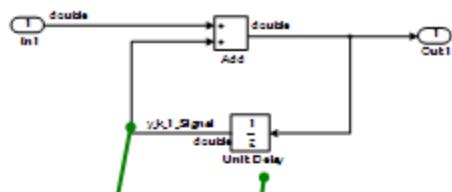
In the case of automatic code generation, the signal name, type, and initial value can be set for state variables by matching it to the signal in the data dictionary (associated with Simulink signal objects). When using a signal defined in the data dictionary for a state variable, the respective initial values should conform to the same value.

- When using a signal defined in the data dictionary for a state variable

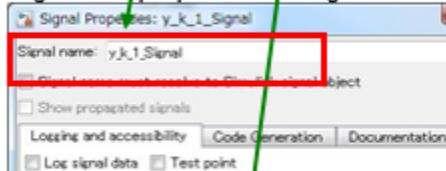
For discrete blocks, such as Unit Delay and Data Store Memory, settings are performed not when using signals defined in the data dictionary for the block output line, but for the state variables inside the block. Even when the signal name of the data dictionary is assigned to the signal line, RAM is reserved in duplicate, which is a waste of RAM.

Example — Correct

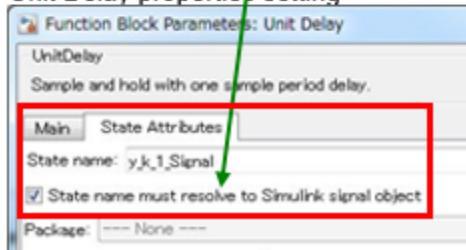
Signal is defined for the state variables inside the block. The signal name is defined and block parameter **State name must resolve to Simulink signal object** is selected.



Signal line properties setting

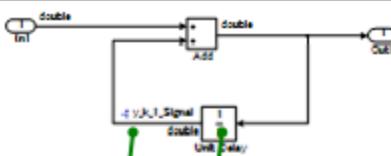


Unit Delay properties setting

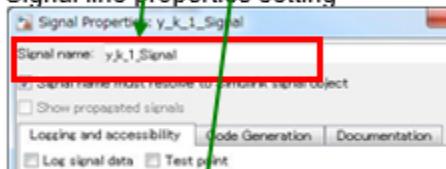


Example — Incorrect

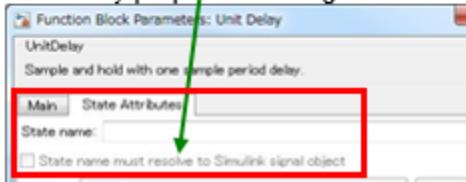
Signal is defined for the output signal of the block that has state variables. The signal name is defined and block parameter **State name must resolve to Simulink signal object** is not selected.



Signal line properties setting



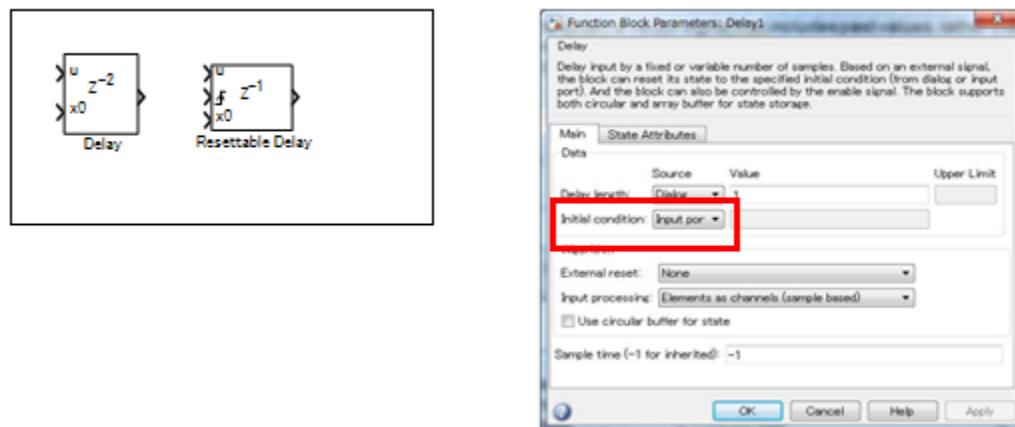
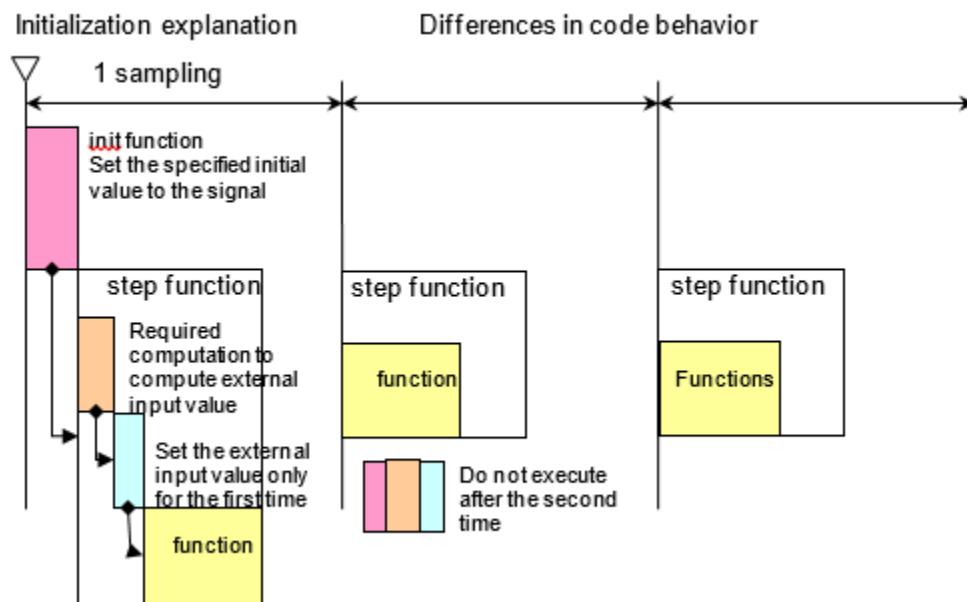
Unit Delay properties setting



Signal objects that are defined in the Workspace can be automatically associated with signal objects and signal names of the same name by using `disableimplicitsignalresolution(modelname)`. However, for state variables inside the block, they are associated with the state variables inside the block and the signal name of the same name. If a globally set signal is associated with two variables at the same time, it is better to perform settings so that the state variables inside a block and the signal label on the signal line have different names, otherwise the model cannot be simulated.

Block Whose External Input Value Is the Initial Value

When setting the initial value during initialization, the `init` function is called to set the signal to either the value inside of the block or to the initial value that is defined in the data dictionary. Next, the step function (the data flow executive function) is executed. Here, the external input value is set as the initial value. When modelling, be attentive to the execution functions and execution timing for initialization. This is demonstrated in the following image.



Initial Value Settings in a System Configuration That Would Enable Initialization Parameters

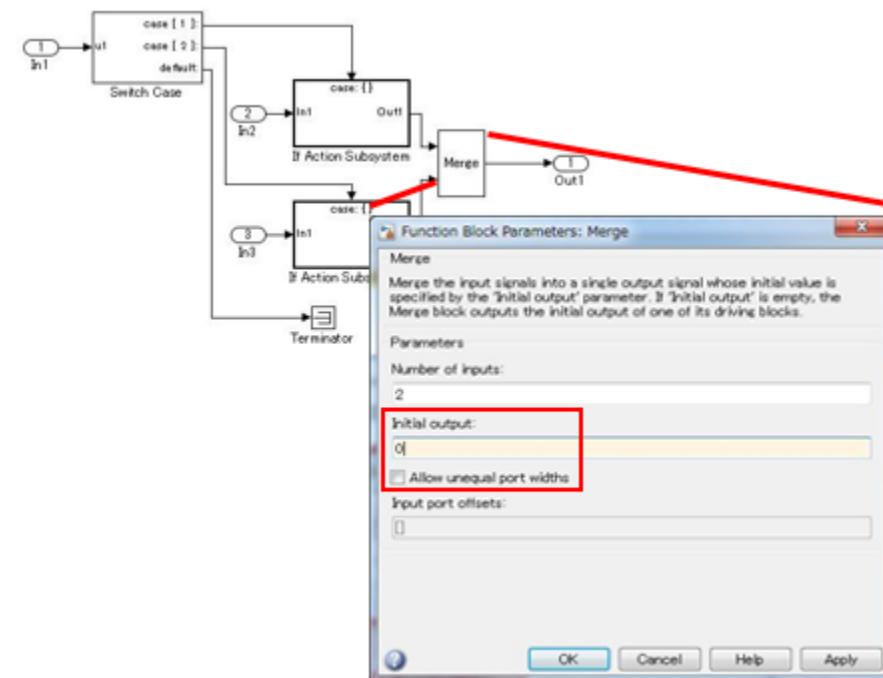
There are system configurations where, depending on their settings, initialization parameters are enabled for combinations of conditional subsystems and Merge blocks. When initial values are required in these combinations, either of the following modeling methods is performed:

- Set values in the Outport block
- Set in values the Merge block
- When an `mpt` signal is defined behind the Merge block, set the values in the `mpt` signal

The exception is when there are successive blocks with initial values and the settings for each block are not needed to clearly show the signal's initial value.

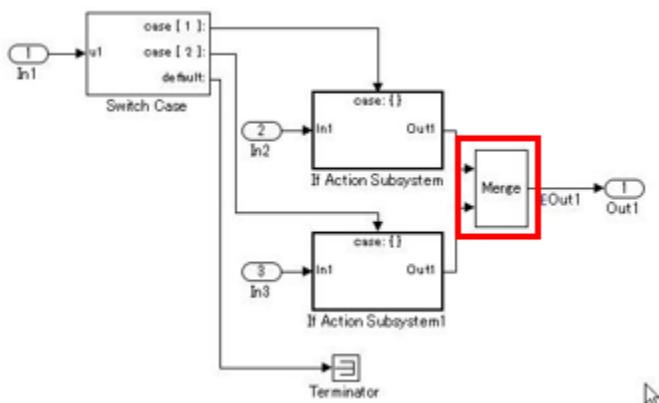
Example – Correct

Initial value set in the Merge block.



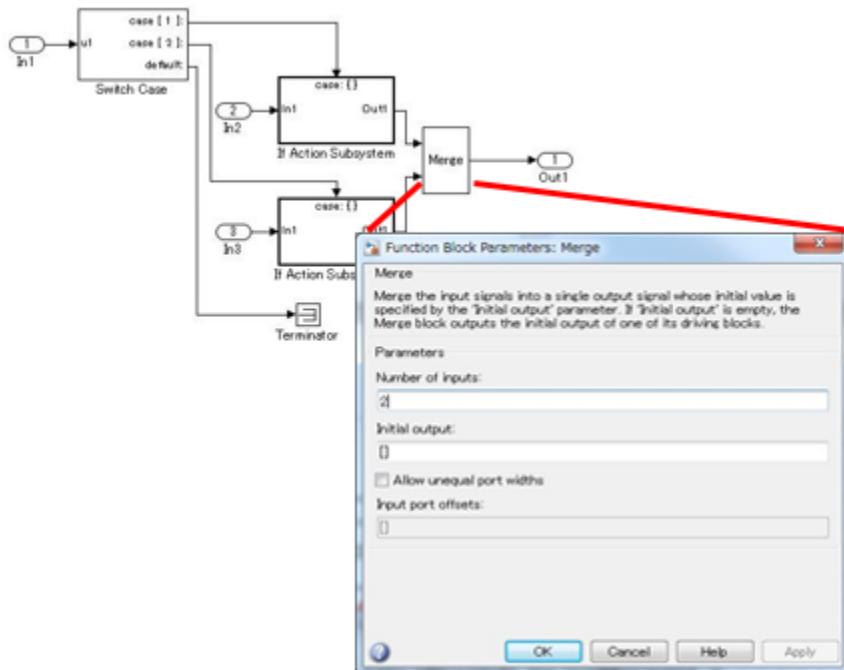
Example – Correct

Initial value set in `mpt` object.



Example — Incorrect

Despite the requirement for an initial value setting, it is not shown anywhere.



Modeling Knowledge

Usage Patterns

Simulink Patterns for If, elseif, else Constructs

These patterns shall be used for if, elseif, else constructs.

Function	Simulink pattern
<pre>if, elseif, else construct when the Switch block is used.</pre> <pre>if (If_Condition) { output_signal = If_Value; } else if (Else_If_Condition) { output_signal = Else_If_Value; } else { output_signal = Else_Value; }</pre>	<p>The diagram shows a Simulink model for an if-elseif-else construct. It uses a switch block with three inputs. The first input, labeled '1 In1', connects to a less-than-or-equal-to block (≤). The output of this block goes to a switch block. The second input, labeled '2 In2', connects to an equals block (=). The output of this block also goes to the same switch block. The third input, labeled '5', connects directly to the switch block. The switch block has three outputs: one for each branch and one for the else branch. These outputs then connect to logic blocks (inverted AND gates) which produce the final output signal.</p>
<pre>if, elseif, else construct when using and action subsystem</pre> <pre>if (Fault_1_Active & Fault_2_Active) { ErrMsg = SaftyCrit; } else if (Fault_1_Active Fault_2_Active) { ErrMsg = DriverWarn; } else { ErrMsg = NoFaults;</pre>	<p>The diagram shows a Simulink model for an if-elseif-else construct using an action subsystem. It features a switch block with three inputs. The first input, labeled '1', connects to a 'if (Fault_1_Active & Fault_2_Active)' block. The second input, labeled '2', connects to a 'if (Fault_1_Active Fault_2_Active)' block. The third input, labeled '3', connects to a 'else' block. The outputs from these blocks feed into a 'Merge' block, which then leads to the final output '2 Out2'. The 'if' blocks contain logic to handle the fault conditions, while the 'else' block provides a default value.</p>

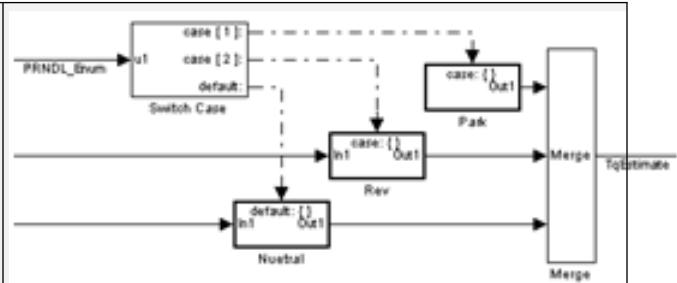
Simulink Patterns for case Constructs

These patterns shall be used for case constructs.

Function	Simulink Pattern
----------	------------------

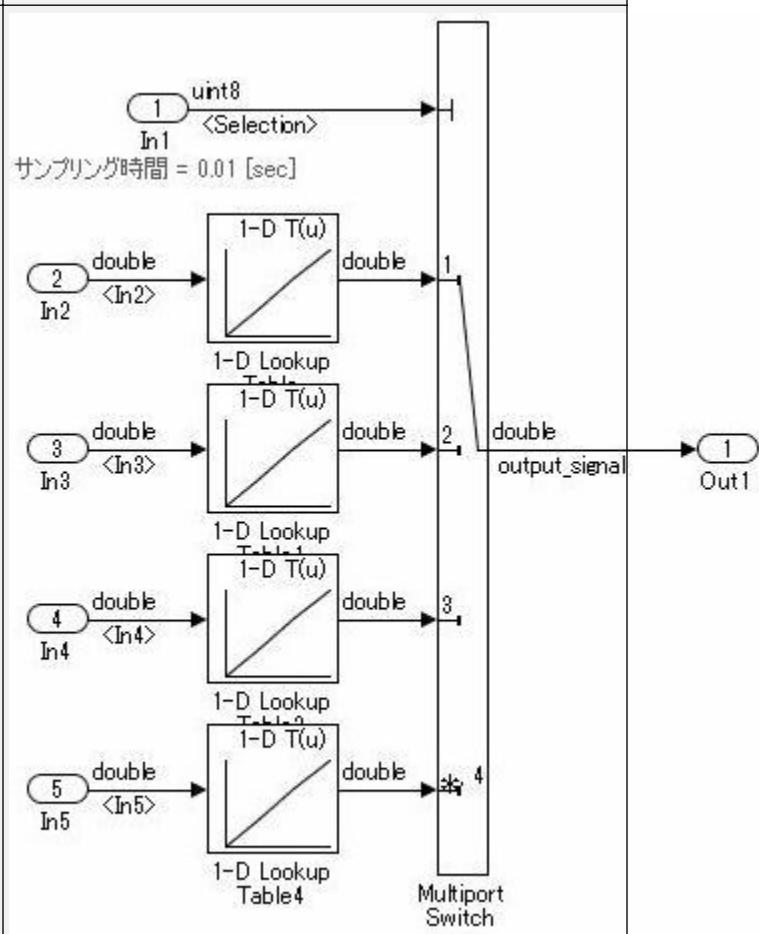
Case construct using an action subsystem.

```
switch (PRNDL_Enum)
{
    case 1
        TqEstimate = ParkV;
        break;
    case 2
        TqEstimate = RevV;
        break;
    default
        TqEstimate = NeutralV;
        break;
}
```



Case construct using a Multiport Switch block.

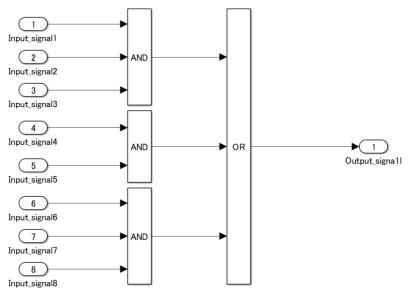
```
switch (Selection)
{
    case 1:
        output_signal =
            look1_binlwpw(In2,y1,x1,3U);
        break;
    case 2:
        output_signal =
            look1_binlwpw(In3,y2,x2,3U);
        break;
    case 3:
        output_signal =
            look1_binlwpw(In4,y3,x3,3U);
        break;
    default:
        output_signal =
            look1_binlwpw(In5,y4,x4,3U);
        break;
}
```



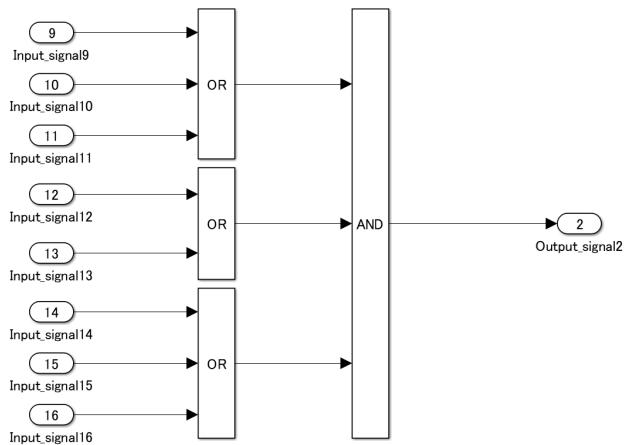
Simulink Patterns for Logical Constructs

These patterns shall be used for logical constructs.

Conjunctive normal form



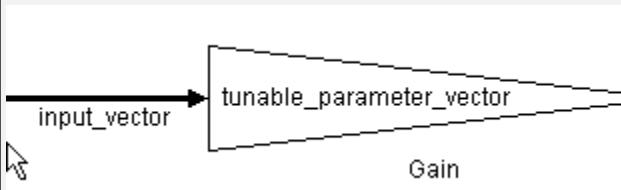
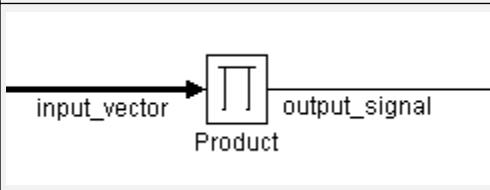
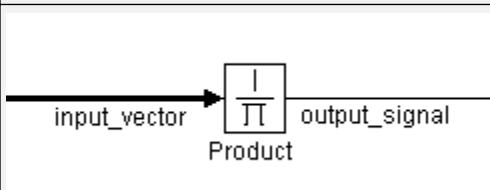
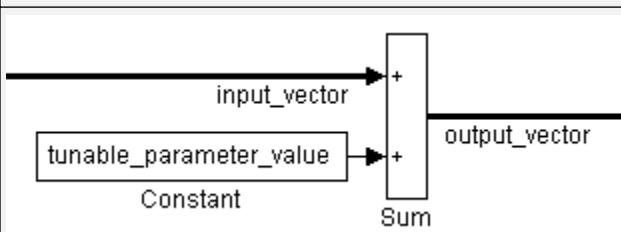
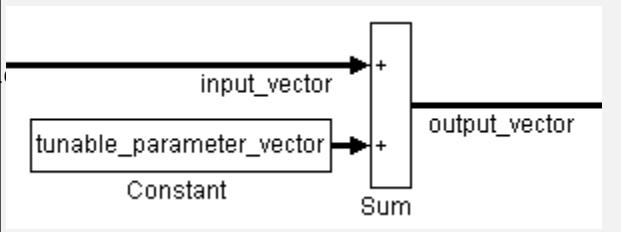
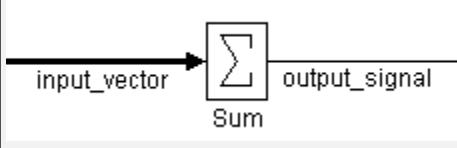
Disjunctive normal form



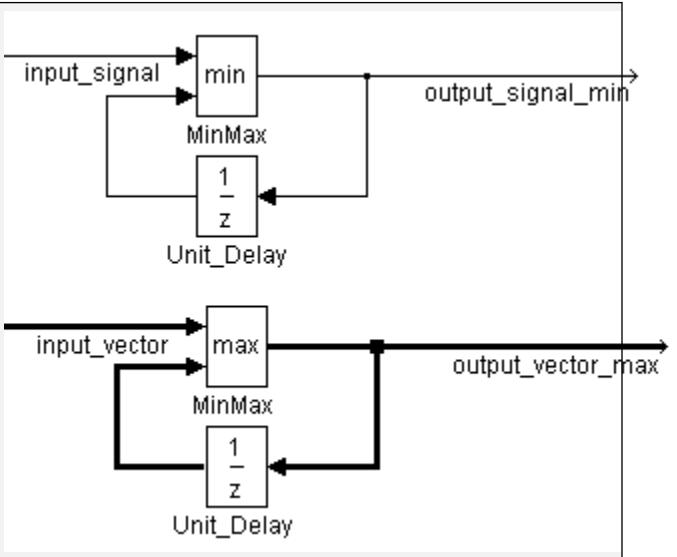
Simulink Patterns for Vector Signals

These patterns shall be used for vector signals.

Function	Simulink Pattern
<p>Vector signal and parameter (scalar) multiplication</p> <pre> for (i=0; i>input_vector_size; i++) { output_vector[i] = input_vector[i] * tunable_parameter_value; } </pre> <p>(Reference: generated code of R2013b)</p> <pre> for (i = 0; i < input_vectorDim; i++) { output_vector[i] = tunable_parameter_value * input_vector[i]; } </pre> <p>(As the code is generated using a variable number of dimensions, the upper limit of the normal loop is a direct value.)</p>	<p>The Simulink pattern shows a Gain block with two inputs: 'input_vector' and 'tunable_parameter_value'. The output of the Gain block is labeled 'output_v'.</p>

Multiplication of vector signals and parameters (vectors)	
<pre>for (i=0; i>input_vector_size; i++) { output_vector[i] = input_vector[i] * tunable_parameter_vector[i]; }</pre>	
Vector signal element multiplication	
<pre>output_signal = 1; for (i=0; i>input_vector_size; i++) { output_signal = output_signal * input_vector[i]; }</pre>	
Vector signal element division	
<pre>output_signal = 1; for (i=0; i>input_vector_size; i++) { output_signal = output_signal / input_vector[i]; }</pre>	
Vector signal and parameter (scalar) addition	
<pre>for (i=0; i>input_vector_size; i++) { output_vector[i] = input_vector[i] + tunable_parameter_value; }</pre>	
Vector signal and parameter (vector) addition	
<pre>for (i=0; i>input_vector_size; i++) { output_vector[i] = input_vector[i] + tunable_parameter_vector[i]; }</pre>	
Vector signal element subtraction	
<pre>output_signal = 0; for (i=0; i>input_vector_size; i++) { output_signal = output_signal - input_vector[i]; }</pre>	

Retention of minimum value/maximum value

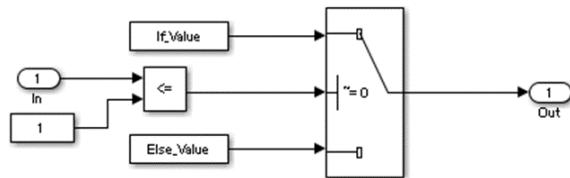


Using Switch and If, Elseif, Else Action Subsystems

The Switch block shall be used for modeling simple `if, elseif, else` structures when the associated `elseif` and `else` actions involve only the assignment of constant values.

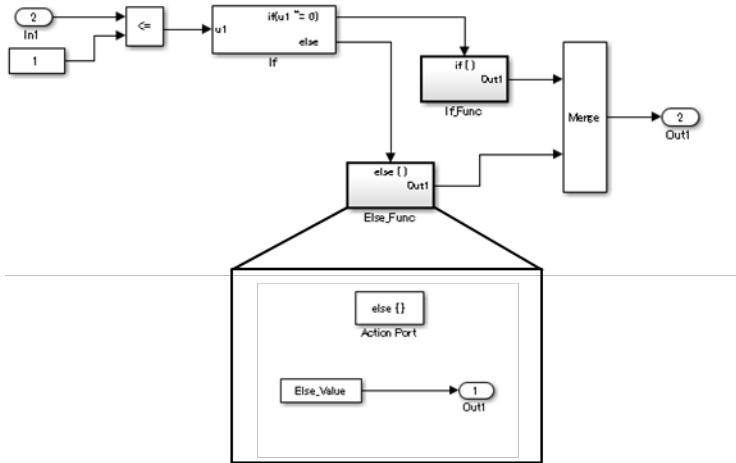
Example — Recommended

For a simple `if, elseif, else` structure, use the Switch block.



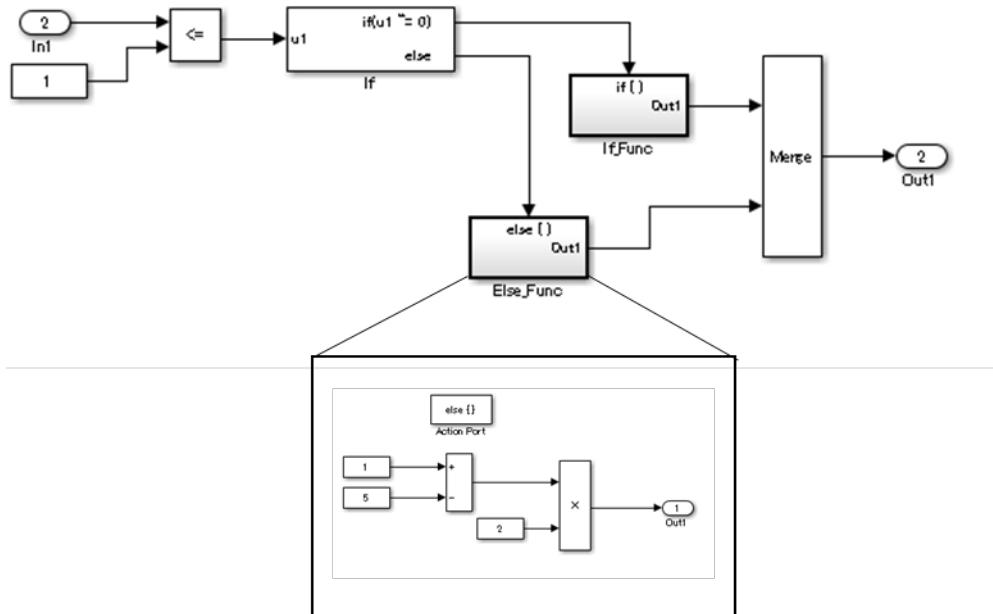
Example — Not recommended

Using If and If Action Subsystem blocks for a simple `if, elseif, else` structure.



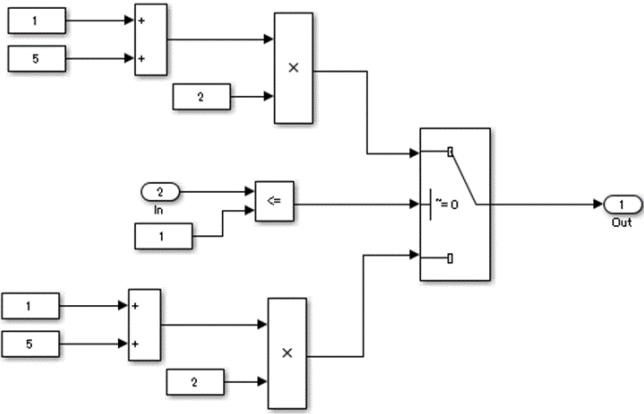
Example – Recommended

For a complex if, elseif, else structure, use If and If Action Subsystem blocks.



Example – Not recommended

Using Switch block for a complex if, elseif, else structure.

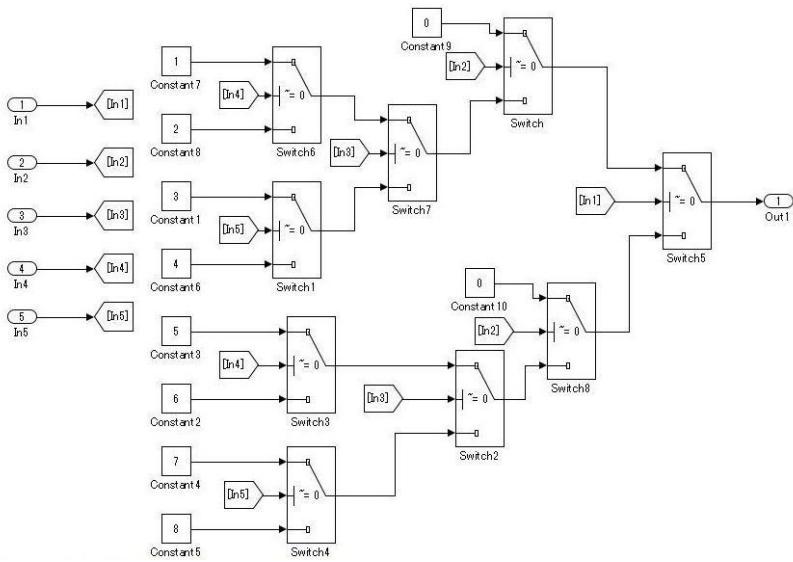


Use of If, Elself, Else Action Subsystem to Replace Multiple Switches

Frequent use of the Switch block for condition bifurcation shall be avoided. Instead, the **upper limit target** shall be used (such as up to three levels). When the target value is exceeded, a conditional control flow using the **if, elseif, else** action subsystem shall be used.

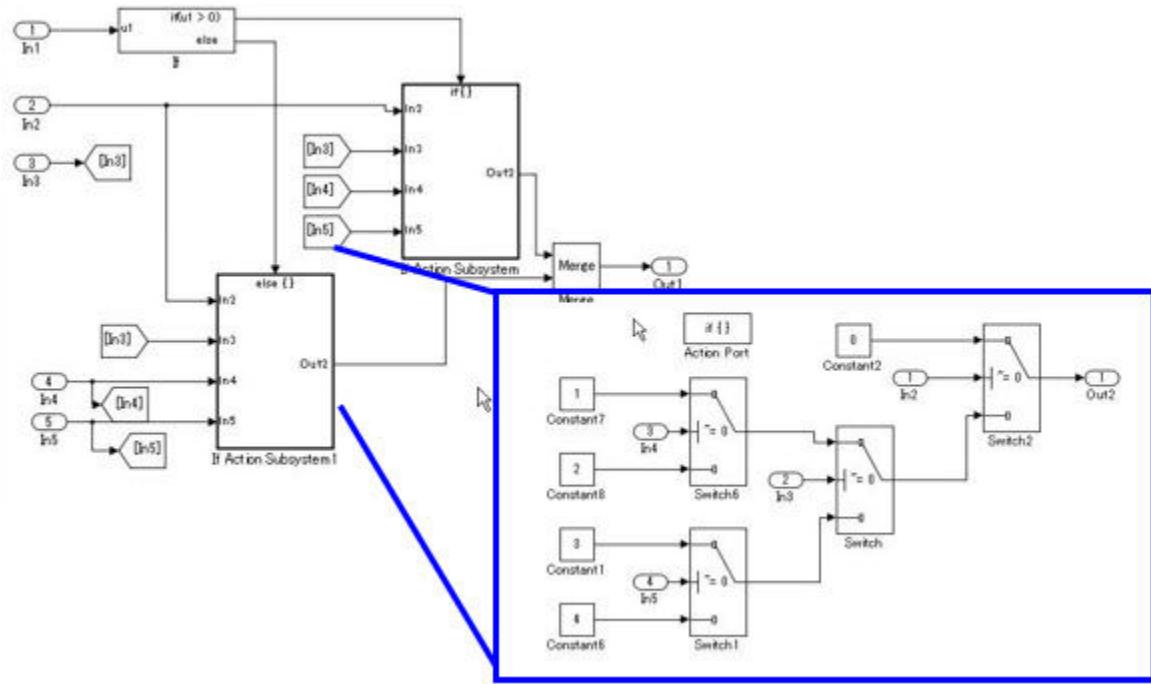
Example — Not recommended

Four levels of nesting.



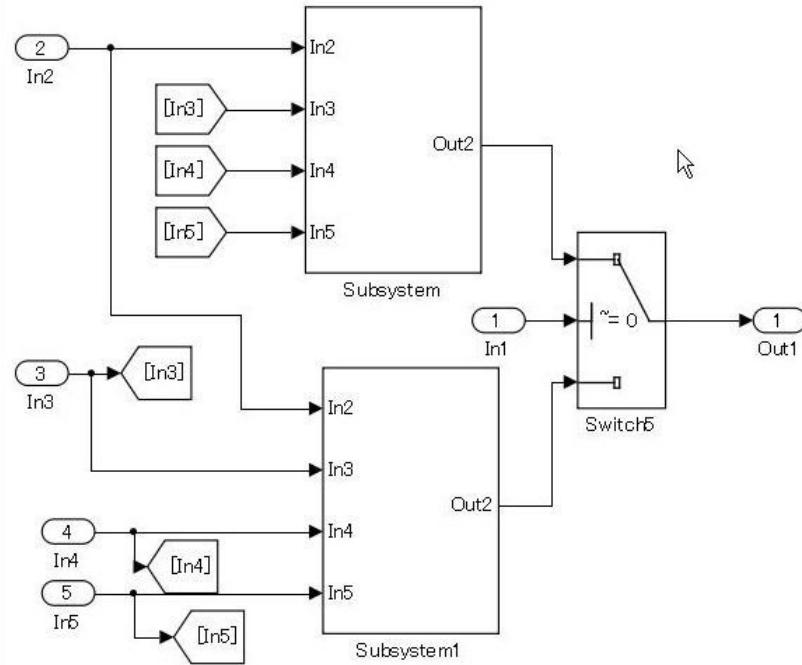
Example — Recommended

By setting the fourth level as an **if** action subsystem, nesting is limited to a single level.



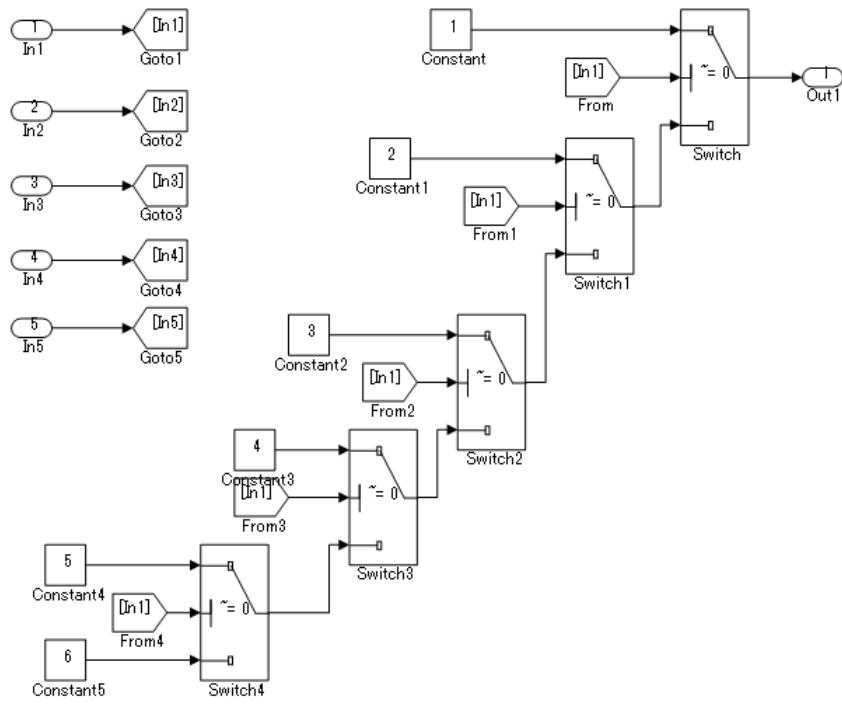
Example — Not recommended

Not dividing by using an if action subsystem.



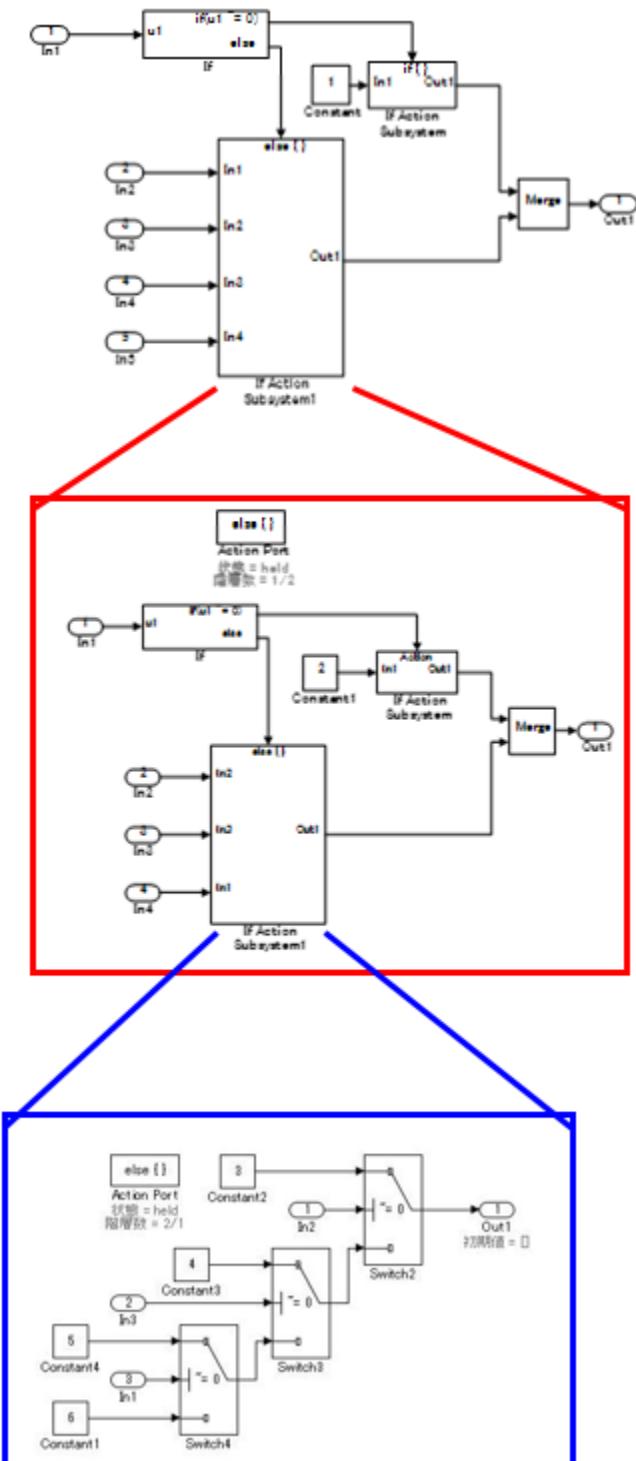
Use atomic subsystem + function setting when the C code limit is applied. In this case, there is no need to use the if, elseif, else action subsystem, but the configuration of the Switch block can be split and encapsulated in the subsystem.

Example of model with five levels of nesting — Not recommended



Example of model with five levels of nesting — Recommended

Use a description method that avoids layering of nesting in the Switch block.

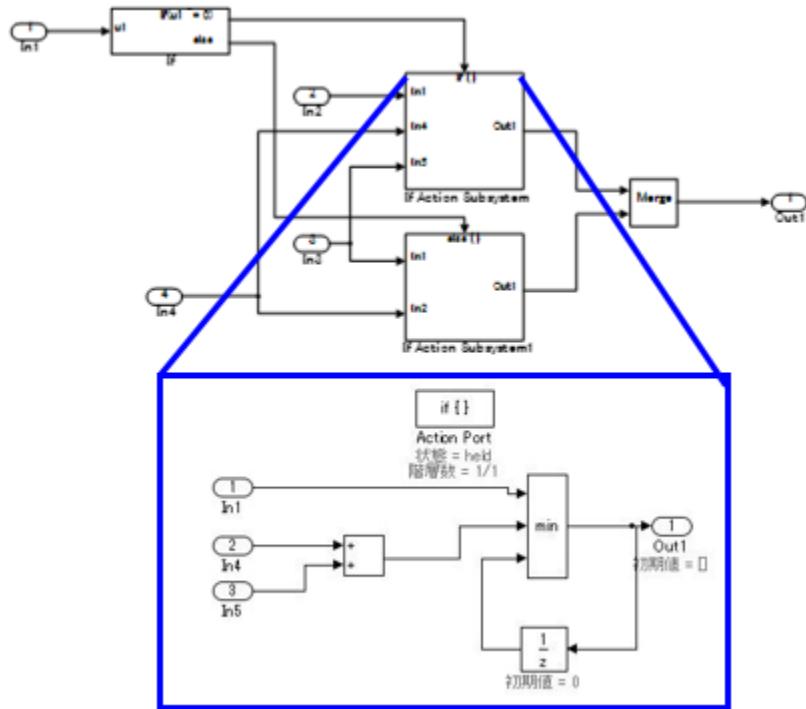


While provided as an example, an if action subsystem is not typically used for switching the fixed value. In these Recommended and Not Recommended examples, the generated C code will be the same if the user does not add a function conversion setting. (Confirmed in R2010b to R2013a) The C code is unconstrained.

Usage Rules for Action Subsystems Using Conditional Control Flow

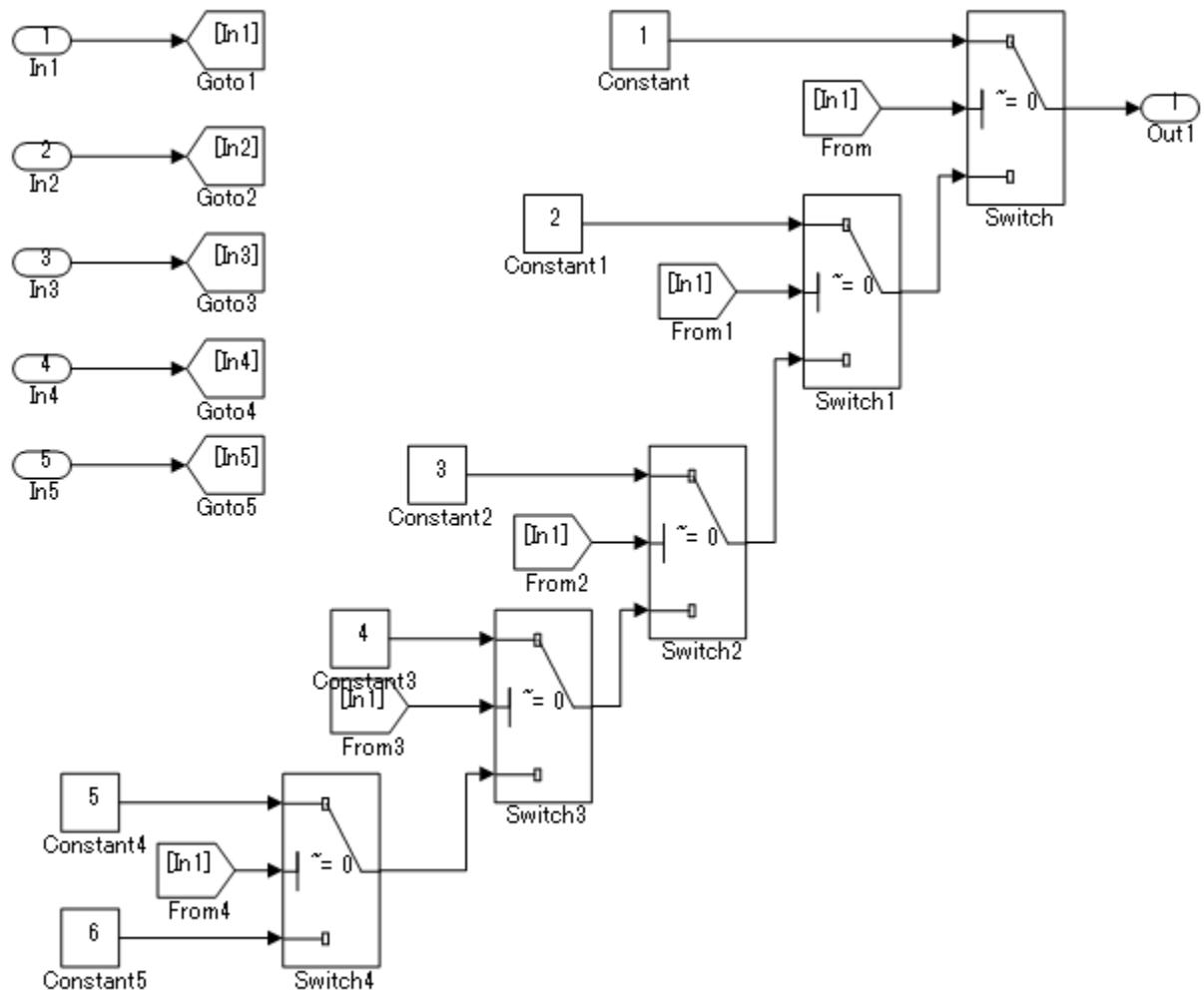
Example — Recommended

An if action subsystem shall not be used when the associated actions do not have a status variable.



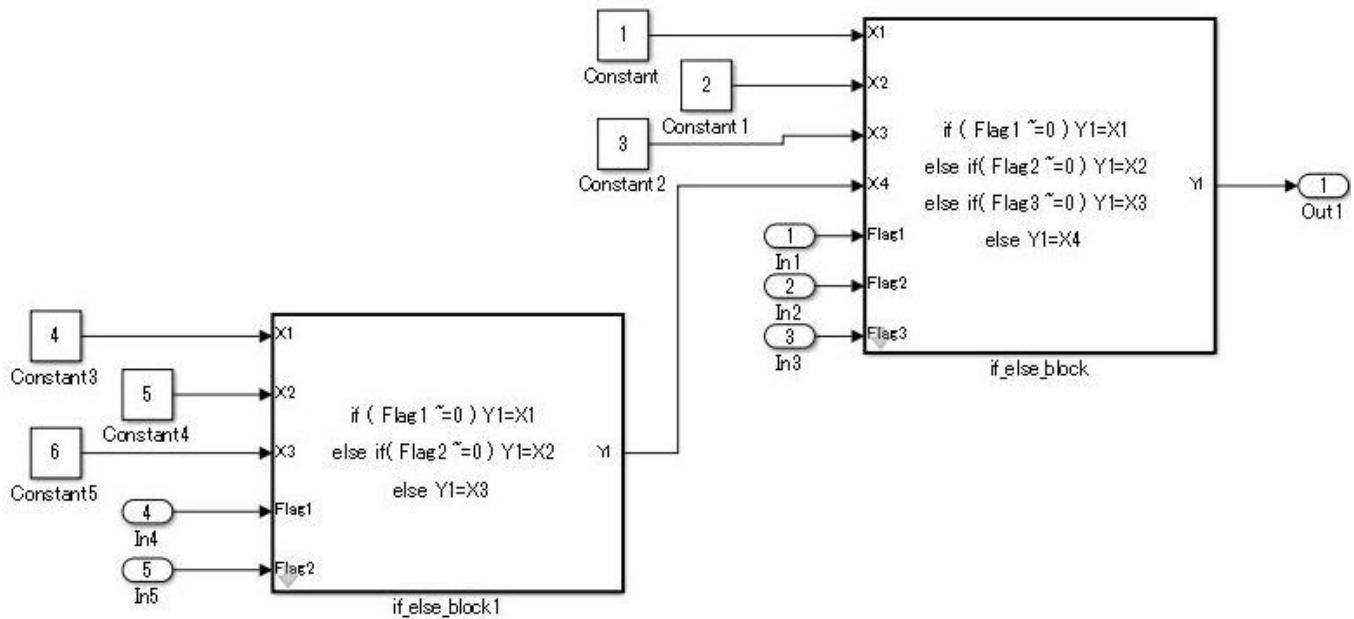
Example — Recommended

Example of a model using five levels of nesting. Layering by using a subsystem does not occur because there is no internal state.



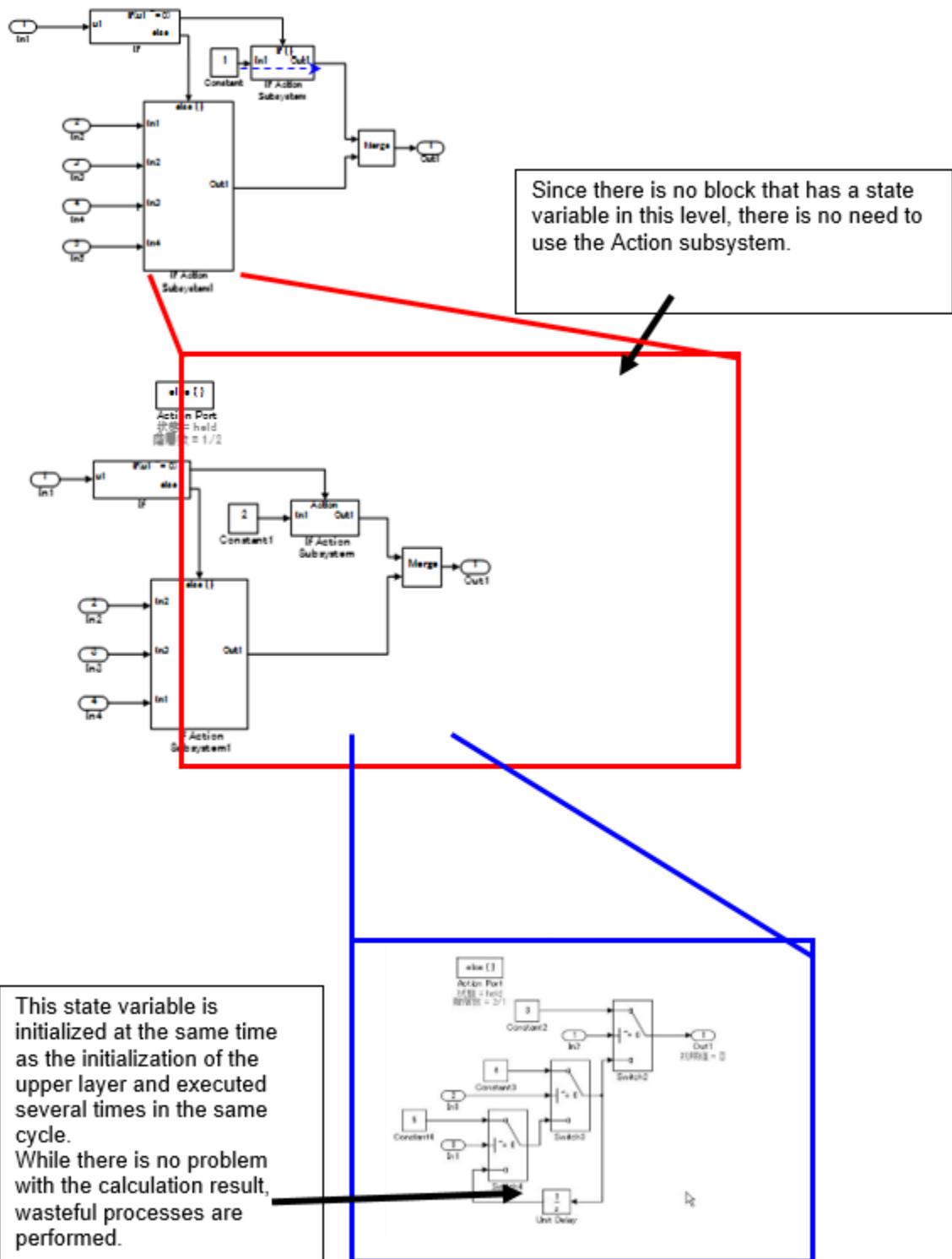
Example — Recommended

An atomic subsystem is used to split either side of the Switch block without using an action subsystem.



Example — Not Recommended

Layering through the use of an unnecessary action subsystem.



If a function can be achieved by using the action subsystem, then layering using the action subsystem is not performed.

In the Not Recommended example, when the lowest level Unit Delay block on the third level is initialized, the conditional subsystem initialization is first executed one time on the upper first level, and then again on the second level for a total of two times of initial value settings. To prevent the generation of unnecessary code, it is recommended that listing not be made in conditional subsystems that reside in levels where the state variable does not exist.

This is based on the concept that the model complexity is reduced by dropping to a level. The purpose of the rule is to avoid the execution of unnecessary initializations.

For bifurcation of systems where the bifurcation condition nest has a deep structure, split by using function conversions to decrease the code bifurcation nesting. Functions before and after the Switch block are divided into respective subsystems, and function settings are applied to the atomic subsystem + function. Be aware, it is possible that this may result in unintentional implementation and unnecessary RAM requirements.

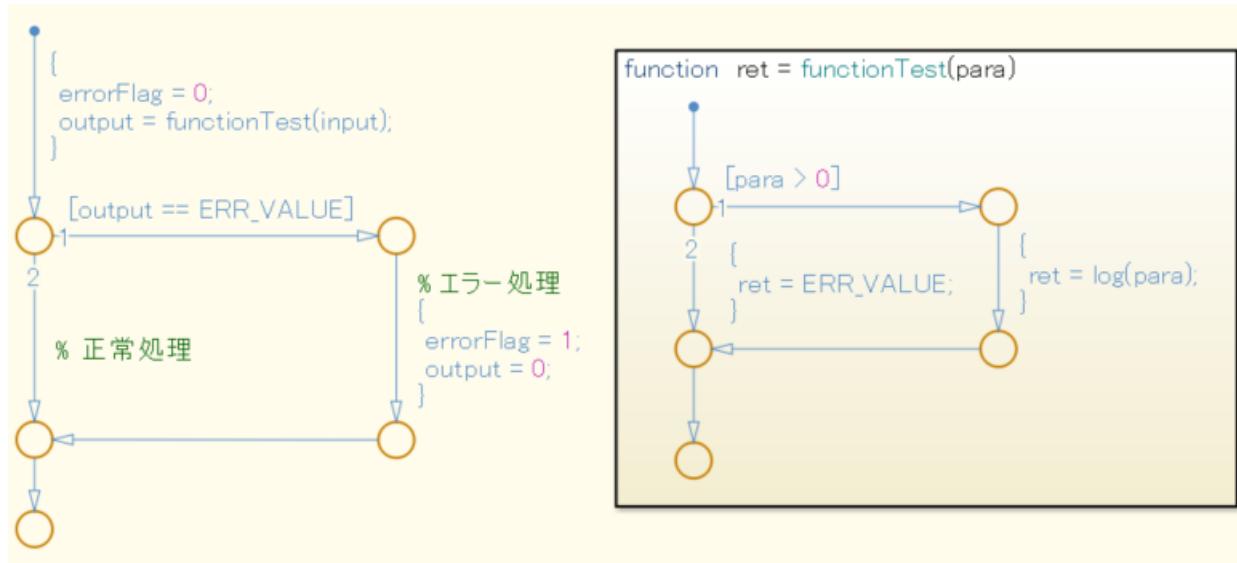
Test for Information From Errors

When functions that are used in Stateflow (graphical functions, MATLAB functions, etc.) results in an error, the error information shall be transformed into a model structure that will facilitate testing.

Not reviewing the error information returned by the functions can result in unintended behavior.

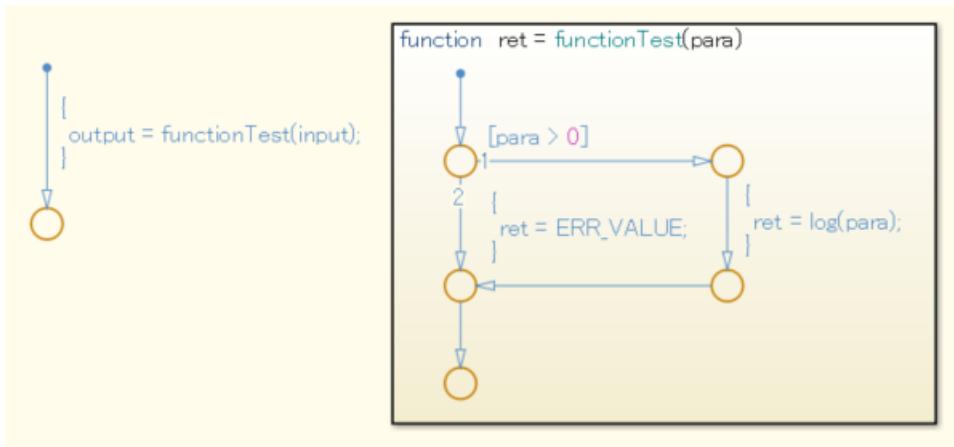
Example — Recommended

Error information is incorporated into the model structure, allowing the user to review and respond to the errors.



Example — Not Recommended

Error information is not incorporated into the model structure.



Flow Chart Patterns for Conditions

These patterns shall be used for conditions within Stateflow flow charts.

Function	Flow Chart Pattern
One condition. [condition]	 <i>* comment */</i>
Up to three conditions, short form. (The use of different logical operators in this form is not allowed. Use subconditions instead.) [condition1 && condition2 && condition3] [condition1 condition2 condition3]	
Two or more conditions, multiline form. (The use of different logical operators in this form is not allowed. Use subconditions instead.) [condition1 ... && condition2 ... && condition3] [condition1 ... condition2 ... condition3]	

<p>Conditions with sub conditions.</p> <p>(The use of different logical operators to connect subconditions is not allowed. The use of brackets is mandatory.)</p> <pre>[(condition1a condition1b) ... && (condition2a condition2b) ... && (condition3)] [(condition1a && condition1b) ... (condition2a && condition2b) ... (condition3)]</pre>	
<p>Conditions that are visually separated.</p> <p>(This form can be combined with the preceding patterns.)</p> <pre>[condition1 && condition2] [condition1 condition2]</pre>	

Flow Chart Patterns for Condition Actions

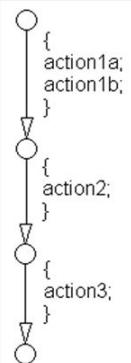
These patterns shall be used for condition actions within Stateflow flow charts

Function	Flow Chart Pattern
<p>One condition action.</p> <pre>action;</pre>	
<p>Two or more condition actions, multiline form.</p> <p>(Two or more condition actions in one line are not allowed.)</p> <pre>action1; ... action2; ... action3; ...</pre>	

Condition actions that are visually separated.

(This form can be combined with the preceding patterns.)

```
action1a;
action1b;
action2;
action3;
```



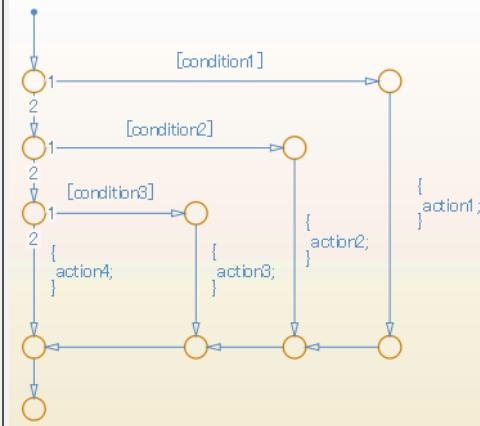
Flow Chart Patterns for If, Elseif, Else Constructs

These patterns shall be used for If constructs within Stateflow flow charts.

Function	Flow Chart Pattern
If construct <pre>if (condition){ action; }</pre>	<pre> graph TD S1(()) -- "[condition]" --> S2(()) S1 --> SF1((())) S2 -- "{action;}" --> SF1 </pre>
If, else construct <pre>if (condition) { action1; } else { action2; }</pre>	<pre> graph TD S1(()) -- "[condition]" --> S2(()) S1 --> SF2((())) S2 -- "{action2;}" --> SF2 S2 --> SF1((())) </pre>

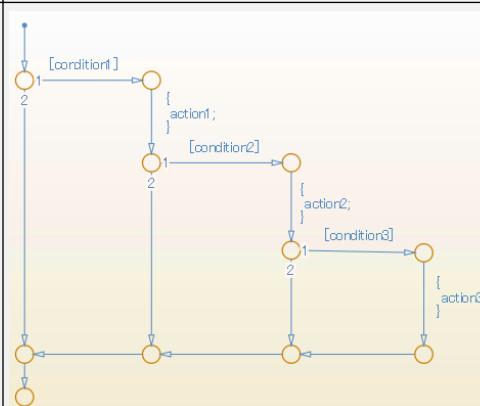
```
If, elseif, else construct

if (condition1) {
    action1;
}
else if (condition2) {
    action2;
}
else if (condition3) {
    action3;
}
else {
    action4;
}
```



Cascade of if construct.

```
if (condition1) {
    action1;
    if (condition2) {
        action2;
        if (condition3) {
            action3;
        }
    }
}
```



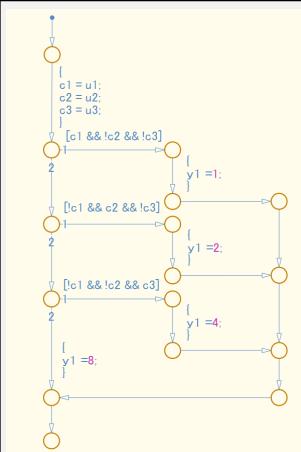
Flow Chart Patterns for Case Constructs

These patterns shall be used for case constructs in Stateflow flow charts.

Function	Flow Chart Pattern
Case construct with exclusive selection. <pre>selection = ul; switch (selection) { case 1: y1 = 1; break; case 2: y1 = 2; break; case 3: y1 = 4; break; default: y1 = 8; }</pre>	<pre> graph TD Start(()) --> S1(()) S1 -- "1" --> C1[selection == 1] C1 --> Y1_1{y1 = 1} S1 -- "2" --> C2[selection == 2] C2 --> Y1_2{y1 = 2} S1 -- "3" --> C3[selection == 3] C3 --> Y1_3{y1 = 4} S1 -- "default" --> D1{y1 = 8} Y1_1 --- M1(()) Y1_2 --- M1 Y1_3 --- M1 D1 --- M1 M1 --- Exit(()) </pre>

Case construct with exclusive conditions.

```
c1 = u1;
c2 = u2;
c3 = u3;
if (c1 && ! c2 && ! c3) {
    y1 = 1;
}
elseif (! c1 && c2 && ! c3) {
    y1 = 2;
}
elseif (! c1 && ! c2 && c3) {
    y1 = 4;
}
else {
    y1 = 8;
}
```



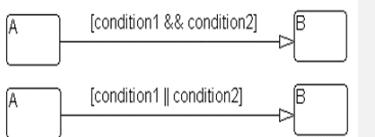
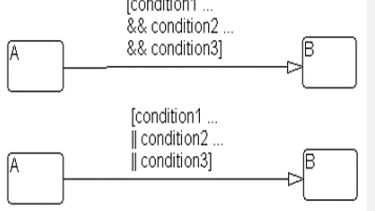
Flow Chart Patterns for Loop Constructs

These patterns shall be used to create loop constructs in Stateflow flow charts.

Function	Flow Chart Pattern
For loop construct	<pre> graph TD Start(()) --> S1((index=0)) S1 --> S2((index++)) S2 -- "[index < number_of_loops]" --> Parallel1[] S2 -- "[action]" --> Parallel1 </pre>
While loop construct	<pre> graph TD Start(()) --> S1(()) S1 --> S2(()) S2 -- "[condition]" --> Parallel2[] S2 -- "[action]" --> Parallel2 </pre>
Do While loop construct.	<pre> graph TD Start(()) --> S1(()) S1 --> S2(()) S2 -- "[condition]" --> Parallel3[] S2 -- "[action]" --> Parallel3 </pre>

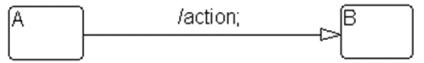
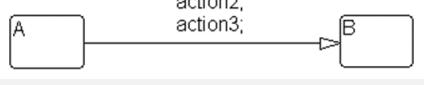
State Machine Patterns for Conditions

These patterns shall be used for conditions within Stateflow state machines

Function	State Machine Pattern
One condition (condition)	
Up to three conditions, short form (The use of different logical operators in this form is not allowed, use sub conditions instead) (condition1 && condition2) (condition1 condition2)	
Two or more conditions, multiline form A subcondition is a set of logical operations, all of the same type, enclosed in parentheses. (The use of different operators in this form is not allowed, use sub conditions instead.) (condition1 ... && condition2 ... && condition3) (condition1 ... condition2 ... condition3)	

State Machine Patterns for Transition Actions

These patterns shall be used for transition actions within Stateflow state machines.

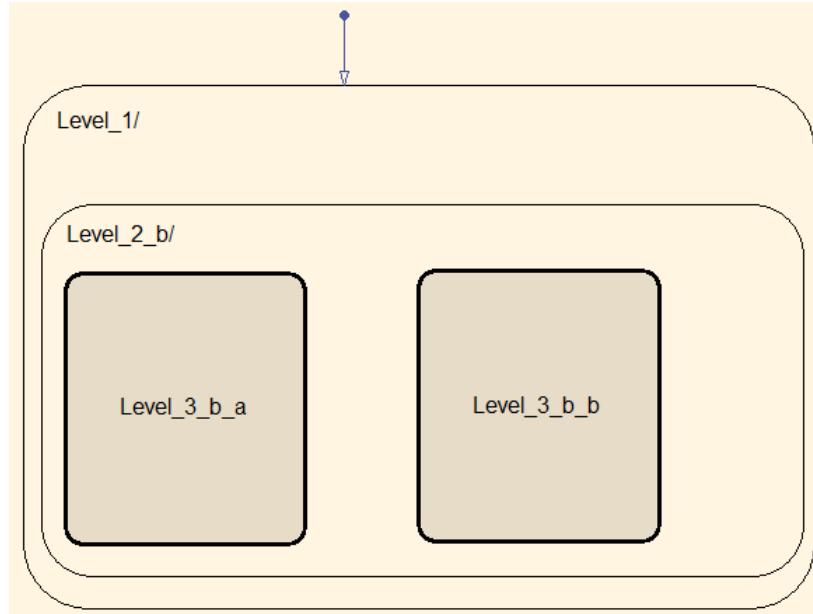
Function	State Machine Pattern
One transition action. action;	
Two or more transition actions, multiline form (Two or more transition actions in one line are not allowed.) action1; action2; action3;	

jc_0321 Limiting State Layering

Within a single viewer (subviewer), multiple layering shall be limited by defining constraints for a single view (subview). Subcharts shall be used to switch the screen when defined constraint goals are exceeded.

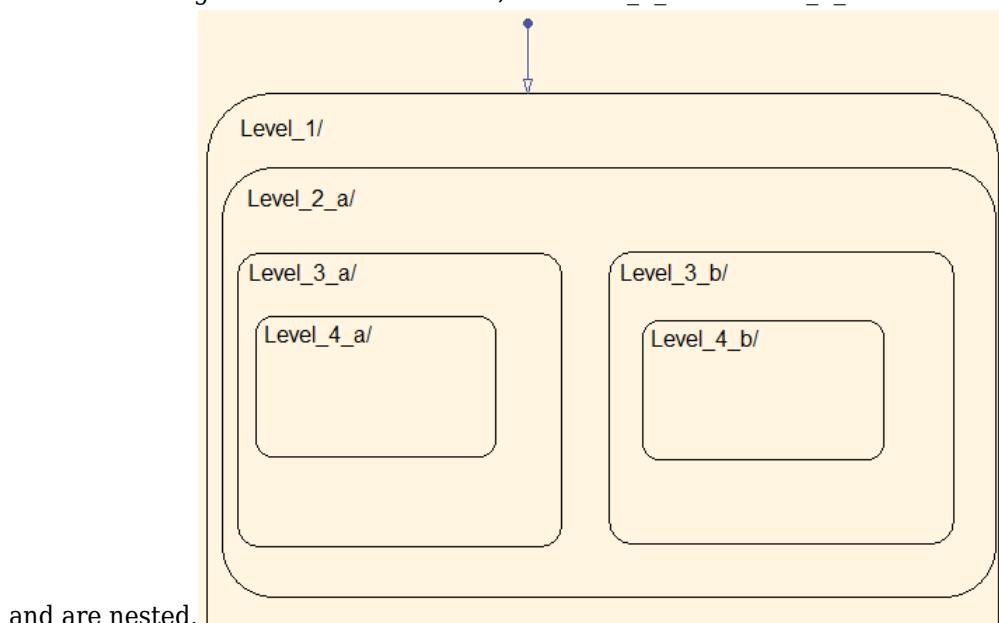
Example – Recommended

The fourth level is encapsulated in a subchart.



Example – Not Recommended

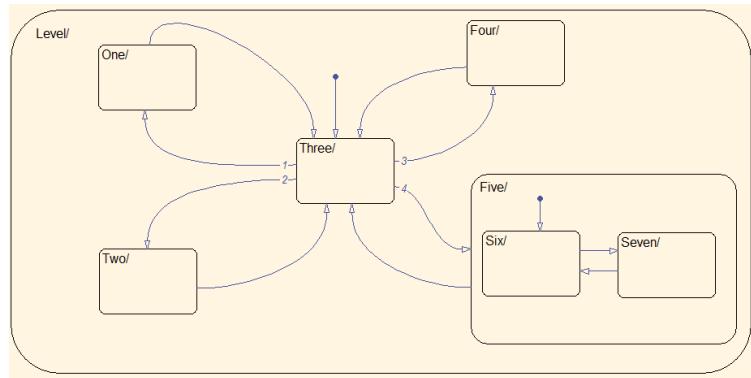
The constraint goal is set to three levels, but Level_4_a and Level_4_b have more than three levels



and are nested.

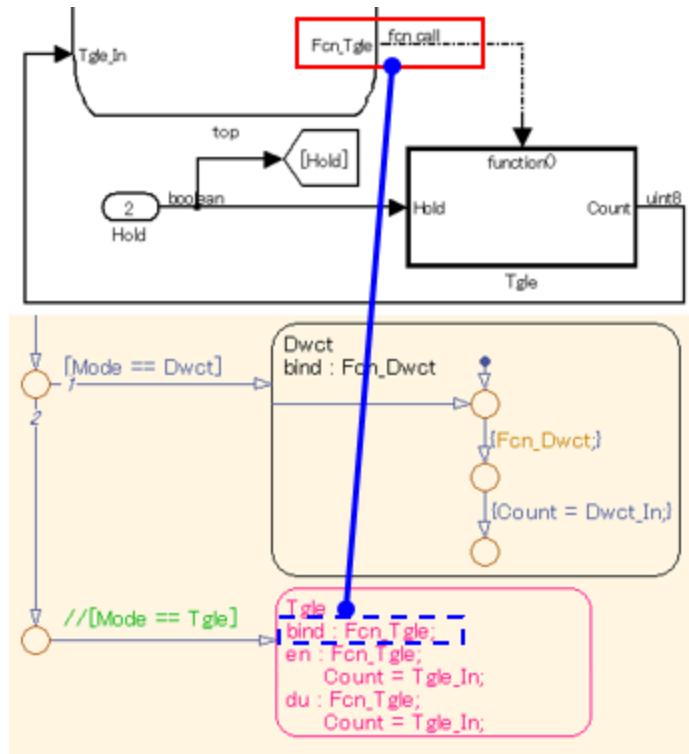
Number of States per Stateflow Container

The number of states per Stateflow container shall be determined by the number of states that can be viewed in the diagram. All states should be visible and readable.



Function Call from Stateflow

If a state exists in the Function-Call Subsystem of the call target, and a “reset” of the state is required when the state of the caller becomes inactive, the caller shall use a bind action.



Function Types Available in Stateflow

The functions types used in Stateflow shall be dependent on the required processing.

For graphical functions, use:

- `If, elseif, else` logic

For Simulink functions, use:

- Transfer functions
- Integrators
- Table look-ups

For MATLAB functions, use:

- Complex equations
- `If, elseif, else` logic

Glossary

Atomic Subsystem

A subsystem block that executes the structural subsystem as a single unit. Conditional subsystems, Stateflow Chart, and MATLAB Function blocks are considered atomic subsystems.

Basic Blocks

Built-in blocks in the standard Simulink library. Blocks with undefined internal processing, such as subsystems, are not considered basic blocks

Basic blocks can include:

- Inport
- Outport
- Ground
- Terminator
- Constant
- Scope
- Saturation
- Unit Delay
- Delay
- Discrete-Time Integrator
- Switch
- Gain
- Product
- Relational Operator
- Logical Operator

Block

All blocks (type=block), including:

- Subsystems
- Models
- Charts (unless otherwise stated).

Standard Simulink library blocks are divided into two categories:

- Basic blocks
- Structural subsystems

Built-in MATLAB functions

MATLAB functions and scripts.

Combined State Action Type

A combination of two or more of these basic state action types:

- entry (en), during (du)
- during (du), exit (ex)
- entry (en), exit (ex)
- entry (en), during (du), exit (ex)

Conditional Input Block

Includes Trigger, Enable, Function-Call Subsystem, Reset blocks.

Conditional Subsystem

A subsystem with conditional input ports.

Delay Block

Two meanings:

- 1 The previous value reference block that is placed in the loop route to specify the execution order in an algebraic loop (circular reference). Uses Unit Delay and Memory blocks.
(As of R2021b and later) The Delay block can also be used
- 2 A block that retains past values. Uses Unit Delay, Memory, Delay, and Tapped Delay blocks.

Calculation Block

Blocks whose block type is "sum" that carry out addition and subtraction operations. Includes Add, Subtract, Sum, Sum of Elements blocks.

Flow Chart

The part of a model that describes the action for the transition condition by using transition conditions and condition actions. The start point is the default transition line or internal transition line. The end point is the connective junction. Does not include states that are between the start and end points. Graphical functions and the inside of states can be modeled as flow charts.

Machine Level

The root subsystem of a Simulink model with Stateflow blocks.

Multiplication and Division Block

Blocks whose block type is "product" that carry out division and multiplication operations. Includes Product, Divide, and Product of Elements blocks.

Parameters

When modifications have not been made, this term refers to constants that are defined in the base workspace and/or model workspace.

Port Label Name

The input and output port labels of a structural subsystem.

The names of Import and Outport blocks are placed in a subsystem by default. Names of Stateflow input and output data are displayed by default.

The display option can be changed when masking a subsystem.

Reserved MATLAB words

MATLAB keywords and built-in MATLAB functions.

State

An atomic subchart is considered a state.

State Action Type

Basic state action types and combined state action types.

Stateflow Block

Includes Chart, State Transition Table, and Truth Table blocks.

Subsystem

A subsystem that can be internally modeled by using Simulink.

Even when the block type is "subsystem", blocks that describe the inside (other than Simulink) of a model, such as Chart, MATLAB Function blocks, are not included. The Model block also is not included.

