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

Version 5.0

MathWorks Advisory Board (MAB)

History

Date	Revision
February 2001	Initial document Release, Version 1.00
April 2007	Version 2.00, Update release
July 2011	Version 2.20, Update release
August 2012	Version 3.0, Update release
March 2020	Version 5.0, MAAB guidelines revised and reintroduced as the MathWorks Advisory Board (MAB) Modeling Guidelines

Table of Contents

1. Introduction

1.1. Purpose of the guidelines

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

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

The chapters of this document provide the following information:
Chapter 1 — Intent of these guidelines and an overview of the guideline template.
Chapters 2 through 5 — Guideline rules
Chapter 6 — Glossary
Chapter 7 — Process for evaluating and implementing guidelines for your project
Chapters 8 — Model architecture and operations that are required for advanced users.
Chapter 9 — Additional explanation and modelling information for Simulink/Stateflow functions, including modeling patterns.

1.2. Guideline template

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.

Rule ID: Title	**xx_nnnn: Title of the guideline (unique, short)**	
Sub ID Recommendations	NA-MAAB: x, y, z JMAAB: x, y, z	
MATLAB® Version	All RX, RY, RZ RX and earlier RX and later RX through RY	
Rule		
Sub ID	**Description**	**Custom Parameter**
xn	(Description of the guideline)	(Parameter Name)
	【Correct】 (Correct image and comment in description)	
	【Incorrect】 (Error image and comment in description)	
Rationale		

Sub ID	**Description**
xn	(Rationale)
See Also	
• XYZ	

Rule ID

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. *db***,** *jm*, *hd***,** *ar* are used for Ver 1.0 guidelines. *na* and *jc* are used for guidelines created from Ver 2.0 to present.

Sub ID Recommendations

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 OEMs and suppliers; NA-MAAB represents North America and Europe. JMAAB represents Japan.

MATLAB® Versions

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:

- 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

Sub ID

Specifies the condition(s) of the rule. There can be multiple sub IDs per rule ID, which are designated as either:

- 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.

Example

```
xy_0000  →  xy_0000a   Selectable (user's choice)
         →  xy_0000b1  Mutually Exclusive (if using, choose from xy_0000b1 or xy_0000b2)
         →  xy_0000b2  Mutually Exclusive (if using, choose from xy_0000b1 or xy_0000b2)
```

Title

The title is unique and provides a brief description of the guidelines.

Description

The description uses figures and tables to provide details for the guideline rules.

This table identifies characters that are used in the description

Description content	Explanation	Example
[] (square brackets)	Block name	[Outport]
{ } (curly brackets)	Block parameter name Stateflow parameter name Configuration parameter settings	{Display propagated signal}

“ ” (double quotation marks)	Parameter setting value	“0”

Custom Parameters

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.

Rational

The rationale provides reasoning for the use of the guideline with regard to readability, verification efficiency, efficiency of code after code generation, etc.

See Also

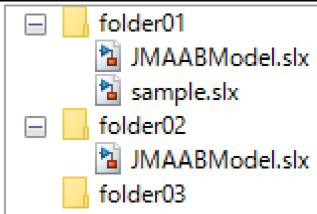
This optional section is only available in guidelines that have additional reference information that may be helpful to better understand the guideline.

2. Naming Conventions

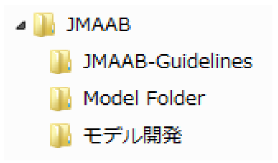
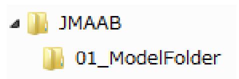
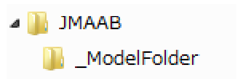
2.1. General Conventions

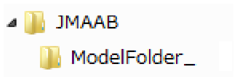
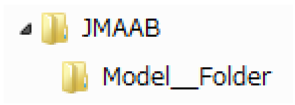
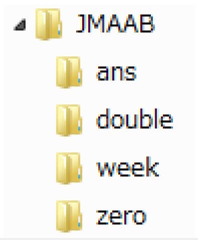
ar_0001: Usable characters for file names

Rule ID: Title	ar_0001: Usable characters for file names	
Sub ID Recommendations	NA-MAAB: a, b, c, d, e, f, g JMAAB: a, b, c, d, e, f, g	
MATLAB® Version	All	
Rule		
Sub ID	Description	Custom Parameter
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.	File (extension)
	【Incorrect】 MAB Model.slx — Single-byte spaces are used. JMAAB 設定.m — Double-byte characters are used. NA-MAABModel.p, JMAAB(Model).mdl — Symbol characters are used.	
b	The file name shall not use numbers at the beginning.	File (extension)
	【Incorrect】 001_JMAABModel.slx	
c	The file name shall not use underscores at the beginning.	File (extension)
	【Incorrect】 _JMAABModel.slx	
d	The file name shall not use an underscore at the end.	File (extension)
	【Incorrect】 MABModel_.slx	
e	The file name shall not use consecutive underscores.	File (extension)
	【Incorrect】 JMAAB__Model.slx	
f	The file name shall not consist solely of a single reserved MATLAB word	File (extension)
	【Incorrect】 ans.slx double.slx week.slx zero.slx, etc.	
g	File names on the MATLAB path shall not be identical.	File (extension)
	【Incorrect】 Files with the same name are saved to the folder that goes through the MATLAB path.	

	 folder01 / JMAABModel.slx / sample.slx / folder02 / JMAABModel.slx / folder03
Rationale	
Sub ID	**Description**
abcf	・Readability is impaired. ・Deviation from the rule can cause unexpected issues.
de	・Readability is impaired.
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.

ar_0002: Usable characters for folder names

Rule ID: Title	**ar_0002: Usable characters for folder names**	
Sub ID Recommendations	NA-MAAB: a, b, c, d, e, f JMAAB: a, b, c, d, e, f	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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.	-
	【Incorrect】  JMAAB / JMAAB-Guidelines / Model Folder / モデル開発 Symbol characters are used. Single-byte spaces are used. Double-byte characters are used.	
b	The folder name shall not use numbers at the beginning.	-
	【Incorrect】  JMAAB / 01_ModelFolder	
c	The folder name shall not use underscores at the beginning.	-
	【Incorrect】  JMAAB / _ModelFolder	
d	The folder name shall not use underscores at the end.	-

	【Incorrect】  JMAAB ModelFolder_	
e	The folder name shall not use consecutive underscores.	-
	【Incorrect】  JMAAB Model__Folder	
f	The folder name shall not consist solely of a single reserved MATLAB word.	-
	【Incorrect】  JMAAB ans double week zero	
Rationale		
Sub ID	**Description**	
abcdef	· Readability is impaired. · Deviation from the rule can cause unexpected issues.	

jc_0241: Length restriction for model file names

Rule ID: Title	**jc_0241: Length restriction for model file names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Model file name length shall be a maximum of 63 characters (not including dots and extension).	Maximum model file name length
Rationale		
Sub ID	**Description**	
a	· Possible that a long file name cannot be referred to in the model reference.	

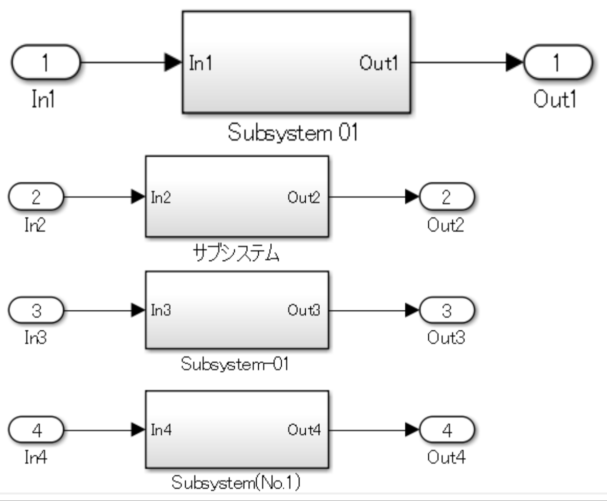
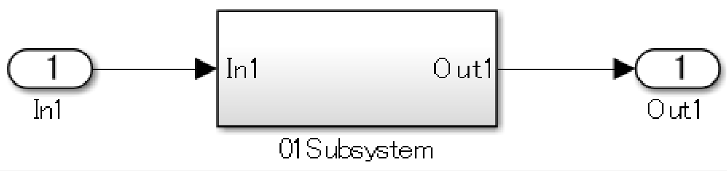
jc_0242: Length restriction for folder names

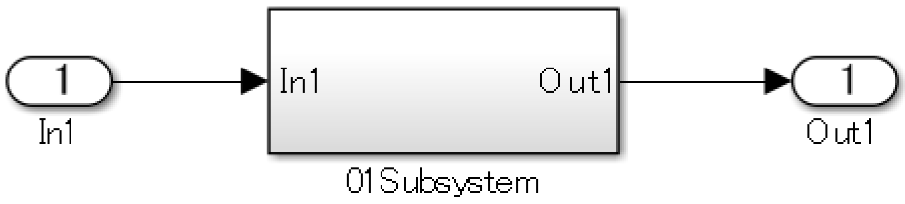
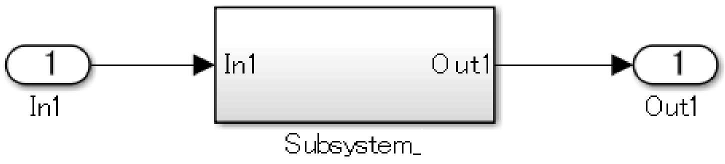
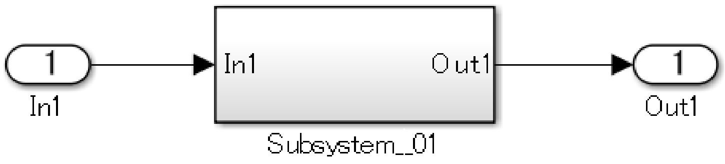
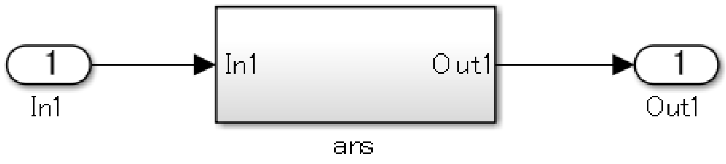
Rule ID: Title	**jc_0242: Length restriction for folder names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Folder name length shall be a maximum of 63	Maximum folder name

	characters.	length
Rationale		
Sub ID	**Description**	
a	・Possible that the full path name cannot be display in the user interface.	

2.2. Content Conventions

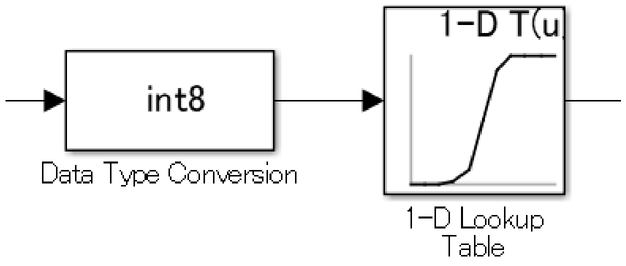
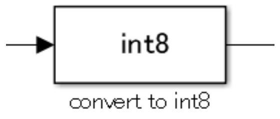
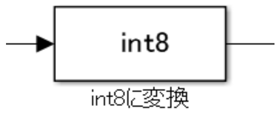
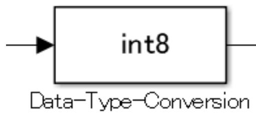
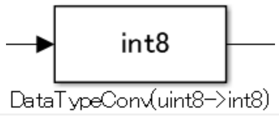
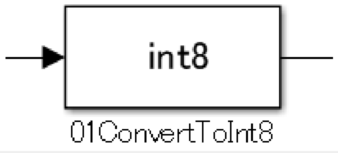
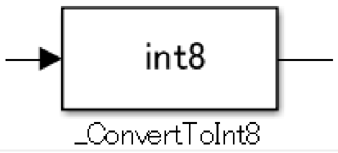
jc_0201: Usable characters for subsystem names

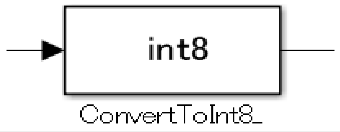
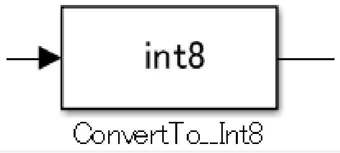
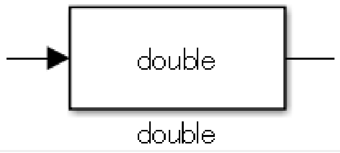
Rule ID: Title	**jc_0201: Usable characters for subsystem names**	
Sub ID Recommendations	NA-MAAB: a, b, c, d, e, f JMAAB: a, b, c, d, e, f	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Only these character types shall be used in structural subsystem names: ・Single-byte alphanumeric 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.	-
	【Incorrect】  1 In1 In1 Out1 1 Out1 Subsystem 01 2 In2 In2 Out2 2 Out2 サブシステム 3 In3 In3 Out3 3 Out3 Subsystem-01 4 In4 In4 Out4 4 Out4 Subsystem(No.1) Uses single-byte spaces. Uses double-byte characters. Uses symbol characters.	
b	A structural subsystem name shall not use numbers at the beginning.	-
	【Incorrect】 	
c	A structural subsystem name shall not use an underscore at the beginning.	-
	【Incorrect】	

		
d	A structural subsystem name shall not use an underscore at the end.	-
	【Incorrect】	
e	A structural subsystem name shall not use consecutive underscores.	-
	【Incorrect】	
f	A structural subsystem name shall not consist solely of a single reserved MATLAB word.	-
	【Incorrect】	
Rationale		
Sub ID	**Description**	
abf	・Cannot generate code using the configured structural subsystem name.	
cde	・May not be able to generate code using the configured structural subsystem name.	

jc_0231: Usable characters for block names

Rule ID: Title	**jc_0231: Usable characters for block names**	
Sub ID Recommendations	NA-MAAB: a, b, c, d, e, f JMAAB: a, b, c, d, e, f	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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 (_) Exception: [Inport] and [Outport]	-

	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.	
	【Correct】 Block names registered in the Simulink library.  int8 Data Type Conversion 1-D T(u) 1-D Lookup Table 【Incorrect】  Single-byte spaces are used.  Double-byte characters are used.  Symbol characters are used. 	
b	Basic block names shall not use numbers at the beginning. Exception: [Inport] and [Outport]	-
	【Incorrect】 	
c	Basic block names shall not use underscores at the beginning. Exception: [Inport] and [Outport]	-
	【Incorrect】 	
d	Basic block names shall not use underscores at the end. Exception: [Inport] and [Outport]	-

	【Incorrect】 	
e	Basic block names shall not use consecutive underscores. Exception: [Inport] and [Outport]	-
	【Incorrect】 	
f	Basic block names shall not consist solely of a single reserved MATLAB word. Exception: [Inport] and [Outport]	-
	【Incorrect】 	

Rationale

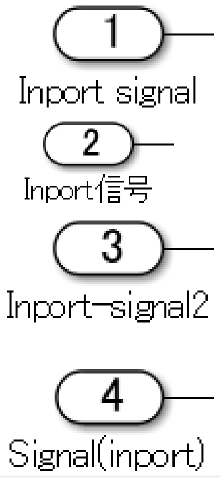
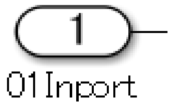
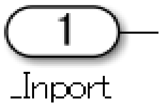
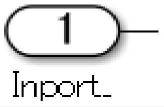
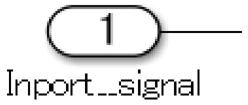
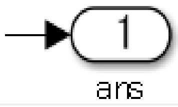
Sub ID	Description
ab	・Deviation from the rule can make it difficult to maintain the integrity of the model and code.
ce	・Readability is impaired.
d	・Readability is impaired. Underscores can be used to separate words. However, they are typically used as word breaks and can cause misunderstanding in the description.
f	・Readability is impaired. ・Deviation from the rule can cause unexpected issues.

jc_0211: Usable characters for Inport block and Outport block

Rule ID: Title	**jc_0211: Usable characters for Inport block and Outport block**
Sub ID Recommendations	NA-MAAB: a, b, c, d, e, f JMAAB: a, b, c, d, e, f
MATLAB® Version	All

Rule

Sub ID	Description	Custom Parameter
a	Only these character types shall be used in [Inport] 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.	-
	【Incorrect】	

	 1 Inport signal 2 Inport信号 3 Inport-signal2 4 Signal(inport) Single-byte spaces are used. Double-byte characters are used. Symbol characters are used.	
b	[Inport] and [Outport] block names shall not use numbers at the beginning.	-
	【Incorrect】 	
c	[Inport] and [Outport] block names shall not use underscores at the beginning.	-
	【Incorrect】 	
d	[Inport] and [Outport] block names shall not use underscores at the end.	-
	【Incorrect】 	
e	[Inport] and [Outport] block names shall not use consecutive underscores.	-
	【Incorrect】 	
f	[Inport] and [Outport] block names shall not consist solely of a single reserved MATLAB word.	-
	【Incorrect】 	
Rationale		
Sub ID	**Description**	
ab	・Deviation from the rule can make it difficult to maintain the integrity of the model and code.	

ce	Readability is impaired.
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.
f	· Readability is impaired. · Deviation from the rule can cause unexpected issues.

jc_0243: Length restriction for subsystem names

Rule ID: Title	**jc_0243: Length restriction for subsystem names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Structural subsystem name length shall be a maximum of 63 characters.	Maximum subsystem name length
Rationale		
Sub ID	**Description**	
a	· Code generation may not be possible.	

jc_0247: Length restriction for block names

Rule ID: Title	**jc_0247: Length restriction for block names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Basic block name length shall be a maximum of 63 characters. Exception: [Inport] and [Outport]	Maximum block name length
Rationale		
Sub ID	**Description**	
a	· Code generation may not be possible.	

jc_0244: Length restriction for Inport and Outport names

Rule ID: Title	**jc_0244: Length restriction for Inport and Outport names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Inport] and [Outport] name length shall be a maximum of 63 characters.	Maximum Inport block name length Maximum Outport block

		name length

Rationale	
Sub ID	**Description**
a	・Code generation may not be possible.

jc_0222: Usable characters for signal/bus names

Rule ID: Title	**jc_0222: Usable characters for signal/bus names**
Sub ID Recommendations	NA-MAAB: a, b, c, d, e, f JMAAB: a, b, c, d, e, f
MATLAB® Version	All

Rule		
Sub ID	**Description**	**Custom Parameter**
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.	-
b	Signal and bus names shall not use numbers at the beginning.	-
c	The signal or bus name shall not use underscores at the beginning.	-
d	Signal and bus names shall not use underscores at the end.	-
e	Signal and bus names shall not use consecutive underscores.	-
f	Signal and bus names shall not consist solely of a single reserved MATLAB word.	-

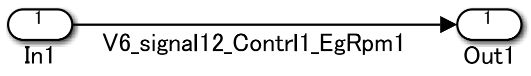
Rationale	
Sub ID	**Description**
ab	・Deviation from the rule can make it difficult to maintain the integrity of the model and code.
ce	・Readability is impaired.
d	・Readability is impaired. Underscores can be used to separate words. However, they are typically used as word breaks and can cause misunderstanding in the description..
f	・Readability is impaired. ・Deviation from the rule can cause unexpected issues.

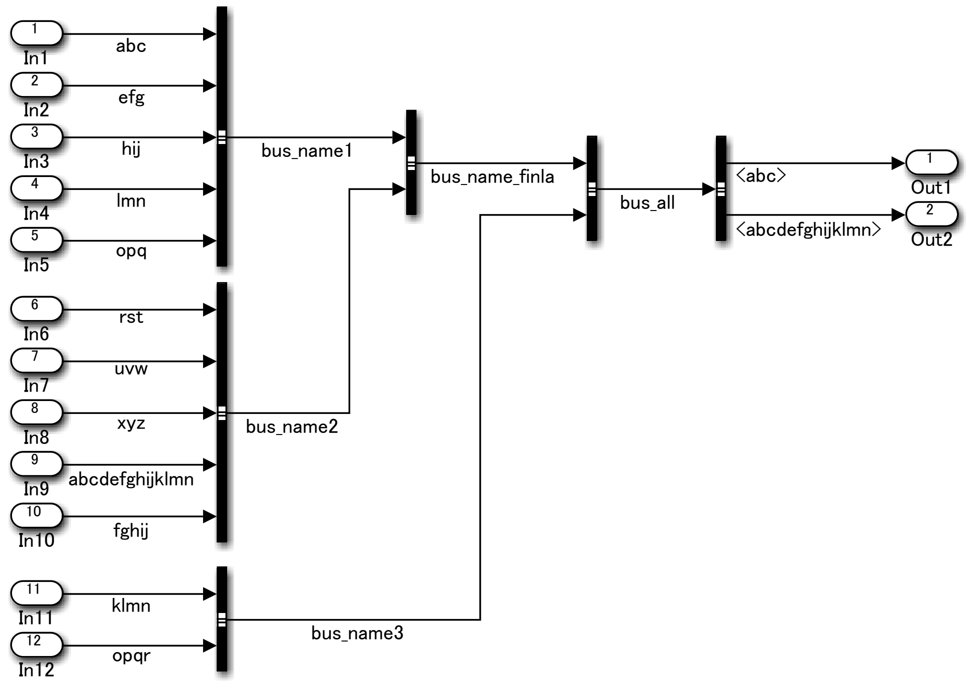
jc_0232: Usable characters for parameter names

Rule ID: Title	**jc_0232: Usable characters for parameter names**
Sub ID Recommendations	NA-MAAB: d, e JMAAB: a, b, c, d, e, f
MATLAB® Version	All
Rule	

Sub ID	Description	Custom Parameter
a	Only these character types shall be used in parameter names: ・Single-byte alphanumeric characters (a-z, A-Z, 0-9) ・Single-byte underscore (_) Line break, single-byte space, double-byte characters, and control characters shall not be used.	-
b	The parameter name shall not use numbers at the beginning.	-
c	The parameter name shall not use underscores at the beginning.	-
d	The parameter name shall not use underscores at the end.	-
e	The parameter name shall not use consecutive underscores.	-
f	The parameter name shall not consist solely of a single reserved MATLAB word.	-
Rationale		
Sub ID	**Description**	
ab	・Deviation from the rule can make it difficult to maintain the integrity of the model and code.	
ce	・Readability is impaired.	
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.	
f	・Readability is impaired. Deviation from the rule can cause unexpected issues.	

jc_0245: Length restriction for signal and bus names

Rule ID: Title	**jc_0245: Length restriction for signal and bus names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Signal and bus name length shall be a maximum of 63 characters.	Maximum signal name length Maximum bus name length
	【Correct】 	

	【Correct】 The hierarchical signal name length (full path bus_all.bus_name_finla.bus_name2.abcdefghijklmn) is less than or equal to 63 characters.  1 In1 abc 2 In2 efg 3 In3 hij 4 In4 lmn 5 In5 opq bus_name1 bus_name_finla bus_all <abc> 1 Out1 <abcdefghijklmn> 2 Out2 6 In6 rst 7 In7 uvw 8 In8 xyz bus_name2 9 In9 abcdefghijklmn 10 In10 fghij 11 In11 klmn bus_name3 12 In12 opqr
Rationale	
Sub ID	**Description**
a	· Code generation may not be possible.

jc_0246: Length restriction for parameter names

Rule ID: Title	**jc_0246: Length restriction for parameter names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Parameter name length shall be a maximum of 63 characters.	Maximum parameter name length
Rationale		
Sub ID	**Description**	
a	· Code generation may not be possible.	

jc_0795: Usable characters for Stateflow data names

Rule ID: Title	**jc_0795: Usable characters for Stateflow data names**
Sub ID Recommendations	NA-MAAB: a, b, c, d JMAAB: a, b, c, d
MATLAB® Version	All

Rule		
Sub ID	**Description**	**Custom Parameter**
a	Stateflow data {name} shall not use underscores at the beginning.	-
b	Stateflow data {name} shall not use underscores at the end.	-
c	Stateflow data {name} shall not use consecutive underscores.	-
d	Stateflow data {name} shall not consist solely of a single reserved MATLAB word.	-
Rationale		
Sub ID	**Description**	
abcd	· Readability is impaired. · Deviation from the rule may result in unintended code behavior.	

jc_0796: Length restriction for Stateflow data names

Rule ID: Title	**jc_0796: Length restriction for Stateflow data names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Stateflow data {name} shall be a maximum of 63 characters.	Stateflow data name character limit
Rationale		
Sub ID	**Description**	
a	· Readability is impaired. · Deviation from the rule can result in unintended code behavior	

jc_0791: Duplicate data name definitions

Rule ID: Title	**jc_0791: Duplicate data name definitions**	
Sub ID Recommendations	NA-MAAB: a, b, c JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Data name definitions shall not be duplicated in the base workspace and model workspace.	-
b	Data names shall not be duplicated in the base workspace and data dictionary (sldd).	Types of data dictionary
c	Data name definitions shall not be duplicated in the model workspace and data dictionary (sldd).	Types of data dictionary
Rationale		
Sub ID	**Description**	
abc	· Duplicated data name can cause unintended model behavior.	

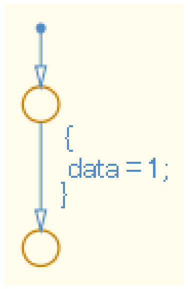
jc_0792: Unused data

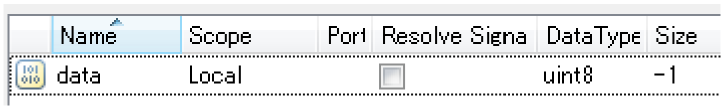
Rule ID: Title	**jc_0792: Unused data**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The data dictionary (sldd) shall define only the data that is used in the Simulink or Stateflow model.	Types of data dictionary
b	The model workspace shall define only the data that is used in the Simulink or Stateflow model.	-
Rationale		
Sub ID	**Description**	
ab	· Unused data can affect maintainability and operability.	

jc_0700: Unused data in Stateflow block

Rule ID: Title	**jc_0700: Unused data in Stateflow block**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Configuration parameter {Unused data, events, messages} shall be set to “Warning” or “Error” to prevent unused Stateflow data, events, and messages in the Stateflow block.	-

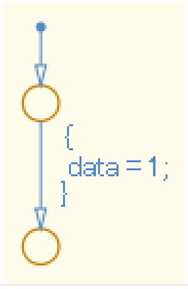
【Correct】

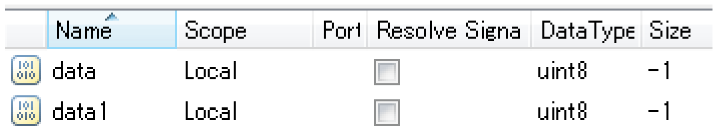




Name	Scope	Port	Resolve Signa	DataType	Size
data	Local		☐	uint8	-1

【Incorrect】
Unused data is defined.





Name	Scope	Port	Resolve Signa	DataType	Size
data	Local		☐	uint8	-1
data1	Local		☐	uint8	-1

Rationale	
Sub ID	**Description**

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.

na_0019: Restricted Variable Names

Rule ID: Title	**na_0019: Restricted Variable Names**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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.	-
b	Variable names that conflict with MATLAB Functions, such as `conv`, shall not be used.	-
Rationale		
Sub ID	**Description**	
ab	・Improves readability of the code. ・Code generation may not be possible.	

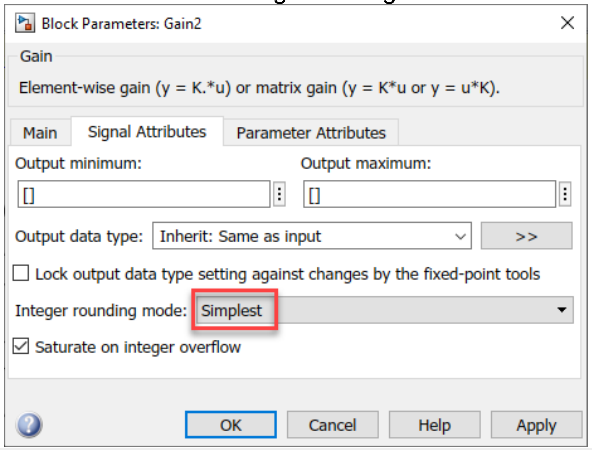
3. Simulink

3.1. Configuration Parameters

jc_0011: Optimization parameters for Boolean data types

Rule ID: Title	**jc_0011: Optimization parameters for Boolean data types**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Configuration parameter {Implement logic signals as Boolean data (vs. double)} shall be selected so that optimization parameters are activated for logic signals.	-
Rationale		
Sub ID	**Description**	
a	Using Boolean data can reduce RAM capacity when using C code.	

jc_0642: Integer rounding mode setting

Rule ID: Title	**jc_0642: Integer rounding mode setting**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When block signal attribute parameter {Integer rounding mode} is set to “Simplest”, configuration parameter {Production hardware signed integer division rounds to} shall be set to “Floor” or “Zero”.	-
	【Correct】 {Production hardware signed integer division rounds to} is set to “Zero”. 	

Block Parameters: Gain2

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Output minimum: []

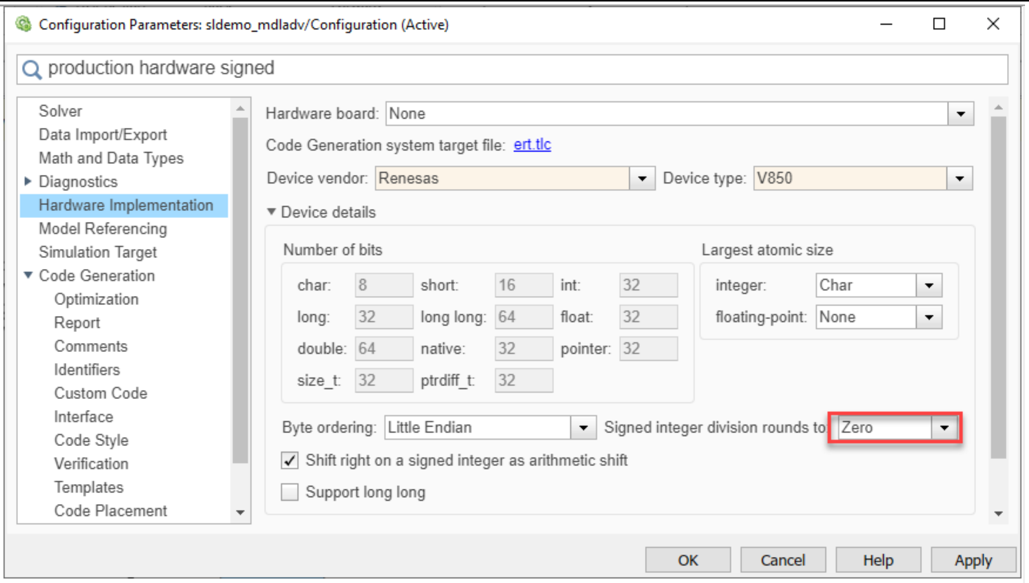
Output maximum: []

Output data type: Inherit: Same as input

☐ Lock output data type setting against changes by the fixed-point tools

Integer rounding mode: Simplest

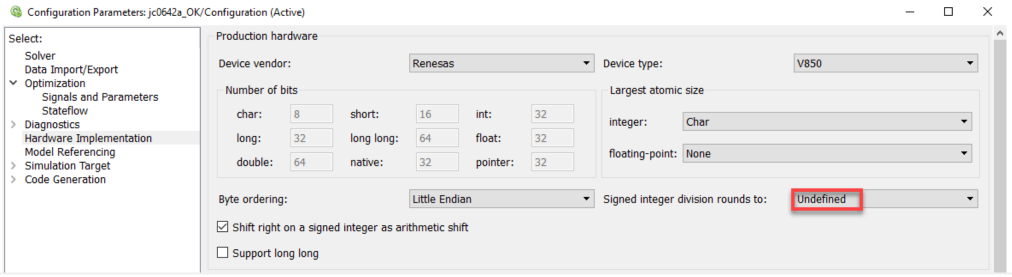
☑ Saturate on integer overflow


Configuration Parameters: sldemo_mdladv/Configuration (Active)

production hardware signed

Hardware board: None
Code Generation system target file: ert.tlc
Device vendor: Renesas   Device type: V850
Device details
Number of bits
char: 8   short: 16   int: 32
long: 32   long long: 64   float: 32
double: 64   native: 32   pointer: 32
size_t: 32   ptrdiff_t: 32
Largest atomic size
integer: Char
floating-point: None
Byte ordering: Little Endian   Signed integer division rounds to: Zero
☑ Shift right on a signed integer as arithmetic shift
☐ Support long long

【Incorrect】
Configuration parameter {Production hardware signed integer division rounds to} is set to “Undefined”.


Configuration Parameters: jc0642a_OK/Configuration (Active)

Production hardware
Device vendor: Renesas   Device type: V850
Number of bits
char: 8   short: 16   int: 32
long: 32   long long: 64   float: 32
double: 64   native: 32   pointer: 32
Largest atomic size
integer: Char
floating-point: None
Byte ordering: Little Endian   Signed integer division rounds to: Undefined
☑ Shift right on a signed integer as arithmetic shift
☐ Support long long

Rationale	
Sub ID	**Description**
a	・Prevents unintended rounding of divided signed integers.
See Also	
・Sub ID a, see MISRA AC SLSF 008B	

jc_0806: Detecting incorrect calculation results

Rule ID: Title	**jc_0806: Detecting incorrect calculation results**	
Sub ID Recommendations	NA-MAAB: a, b, c JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Configuration parameter {Division by singular matrix} shall be set to “Error”.	-
b	Configuration parameter {Inf or NaN block output} shall be set to “Error”.	-
c	For R2010b to R2014a, configuration parameter {Detect overflow} shall be set to “Error”.	-

	For R2014b and later, these configuration parameters shall be set to “Error”: • {Wrap on overflow} • {Saturate on overflow}	
Rationale		
Sub ID	**Description**	
abc	・Allows detection of operations with invalid values.	
See Also		
・Sub ID a, see hisl_0005c		

jc_0021: Model diagnostic settings

Rule ID: Title	**jc_0021: Model diagnostic settings**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	These configuration parameters shall be set to “warning” or “error”: • {Algebraic loop} • {Minimize algebraic loop} • {Multitask rate transition} • {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}	-
Rationale		
Sub ID	**Description**	
a	・Improves model workflow. ・Code generation may not be possible.	

3.2. Diagram appearance

na_0004: Simulink model appearance settings

Rule ID: Title	**na_0004: Simulink model appearance settings**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

a	Simulink model appearance settings shall conform with the project settings.	Display option

Example:

View Options	Setting
Model Browser	unchecked
Screen color	white
Status Bar	checked
Toolbar	checked
Zoom factor	Normal (100%)

Block Display Options	Setting
Background color	white
Foreground color	black
Execution Context	unchecked
Library Links	none
Linearization Indicators	checked
Ref. Model I/O Mismatch	unchecked
Ref. Model Version	unchecked
Sample Time **Colors**	unchecked
Execution Order	unchecked

Signal Display Options	Setting
Base Data Types	unchecked
Alias Data Types	unchecked
Signal Dimensions	unchecked
Storage Class	unchecked
Log & Testpoint	checked
Viewers	checked
Nonscalar Signals	checked

Rationale

Sub ID	**Description**
a	· Standard model appearance improves readability.

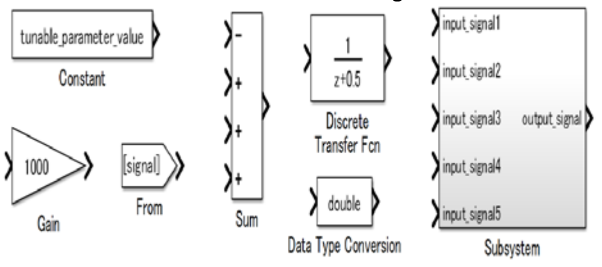
See Also

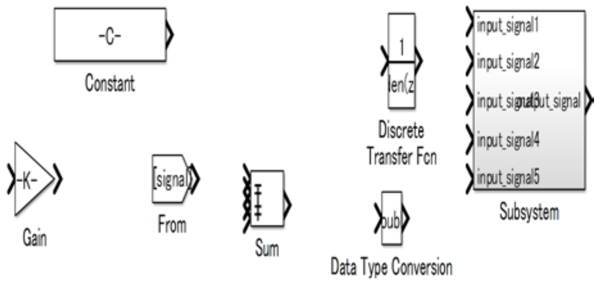
· Sub ID a, see MISRA AC SLSF 023A

db_0043: Model font and font size

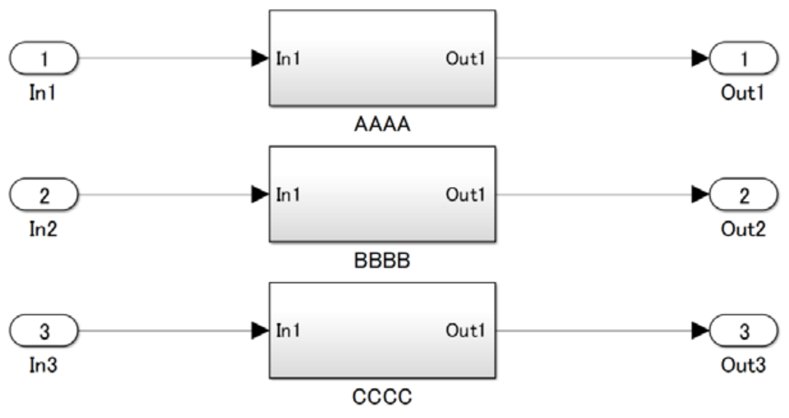
Rule ID: Title	db_0043: Model font and font size	
Sub ID Recommendations	NA-MAAB: a, b, c, d JMAAB: a, b, c, d	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	・Block name {font} and {font style} shall conform with the project settings. ・Signal name {font} and {font style} shall conform with the project settings.	Font Font style
b	・Block name font {size} shall conform with the project settings. ・Signal name font {size} shall conform with the project settings.	Font size
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.	Font Font style
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.	Font size
Rationale		
Sub ID	**Description**	
ac	・Standard fonts improve readability.	
bd	・Standard font size improves readability.	
See Also		
・Sub ID c and d, see MISRA AC SLSF 050B		

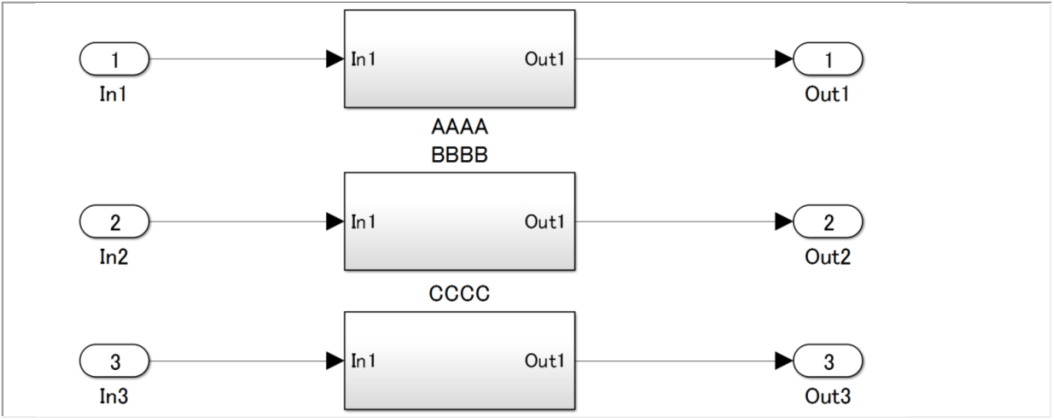
jm_0002: Block resizing

Rule ID: Title	jm_0002: Block resizing	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Blocks shall be sized so that the block icon is visible and recognizable.	-
	【Correct】 The block icon is visible and recognizable.  tunable_parameter_value Constant 1000 Gain [signal] From Sum 1 z+0.5 Discrete Transfer Fcn double Data Type Conversion input_signal1 input_signal2 input_signal3 output_signal input_signal4 input_signal5 Subsystem	

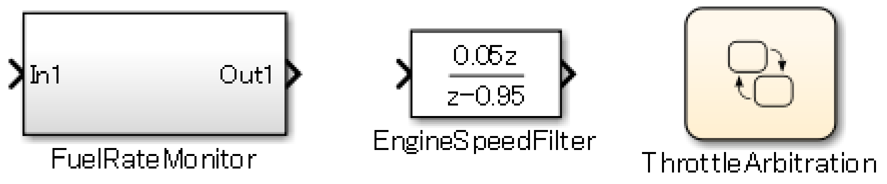
	【Incorrect】 The block is too small, so the icon is neither visible nor recognizable.  -C- Constant -K- Gain [signal] From Sum 1 len(z Discrete Transfer Fcn bub Data Type Conversion input_signal1 input_signal2 input_signal3 output_signal input_signal4 input_signal5 Subsystem
Rationale	
Sub ID	**Description**
a	・When a block is too small, the text and symbol displayed by the icon can be difficult to see, which impairs readability.

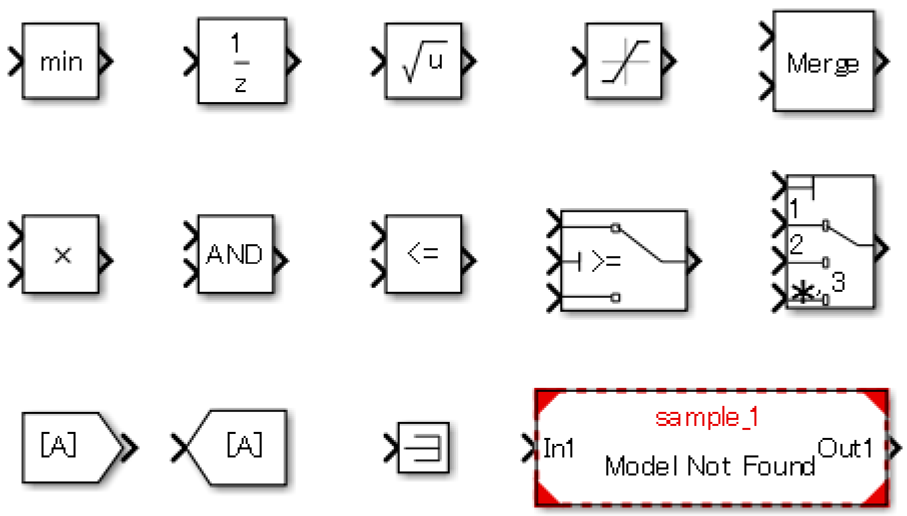
db_0142: Position of block names

Rule ID: Title	**db_0142: Position of block names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The block name shall be positioned below the block.	-
	【Correct】  1 In1 In1 Out1 AAAA 1 Out1 2 In2 In1 Out1 BBBB 2 Out2 3 In3 In1 Out1 CCCC 3 Out3 【Incorrect】	

	 1 In1 In1 Out1 AAAA BBBB 1 Out1 2 In2 In1 Out1 CCCC 2 Out2 3 In3 In1 Out1 3 Out3
Rationale	
Sub ID	**Description**
a	・Consistent placement of the block name improves model readability because it is easier to determine which name corresponds to the block.

jc_0061: Display of block names

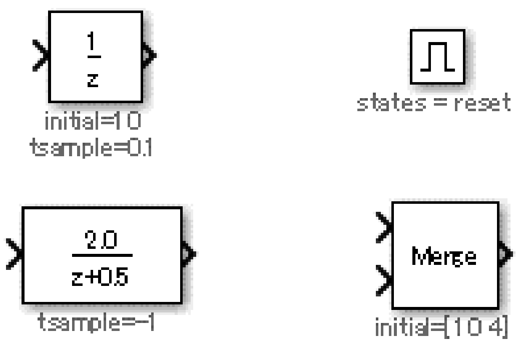
Rule ID: Title	**jc_0061: Display of block names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Block names shall be hidden for blocks that meet the following criteria: ・Block type is evident from its visual appearance ・Uses the 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.	Blocks with a clear type due their appearance
	Example of block names that are displayed  In1 Out1 FuelRateMonitor 0.05z z−0.95 EngineSpeedFilter ThrottleArbitration Example of hidden block names	

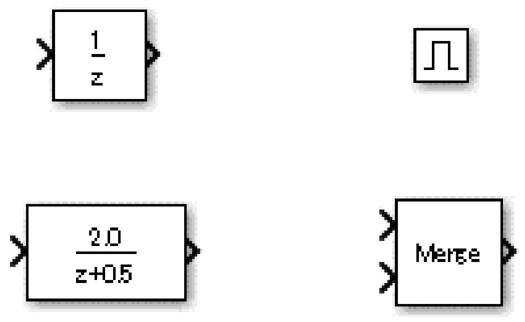
	 min 1 z u Merge × AND <= >= 1 2 3 [A] [A] sample_1 In1 Model Not Found Out1

Rationale	
Sub ID	Description
a	· Improves model readability.

See Also
· Sub ID a, see MISRA AC SLSF 026A

db_0140: Display of block parameters

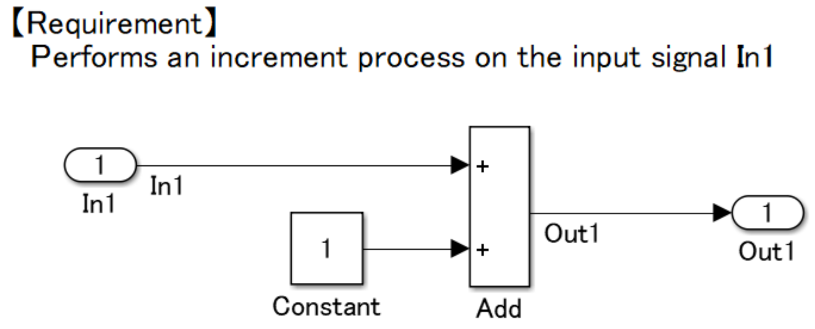
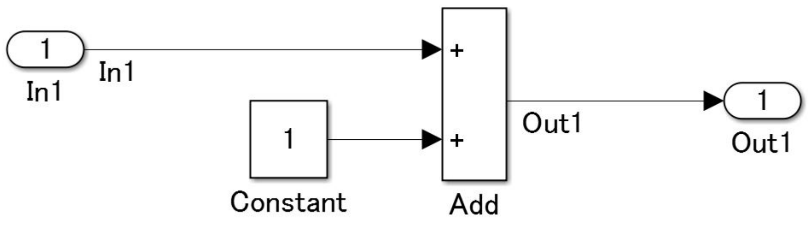
Rule ID: Title	db_0140: Display of block parameters	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	Description	Custom Parameter
a	Block annotation shall display the block parameters that are defined by the project.	Block Parameters
	【Correct】  1 z initial=10 tsample=0.1 states = reset 2.0 z+0.5 tsample=-1 Merge initial=[10 4] 【Incorrect】	

Rationale	
Sub ID	**Description**
a	· Readability improves when block parameters are displayed.

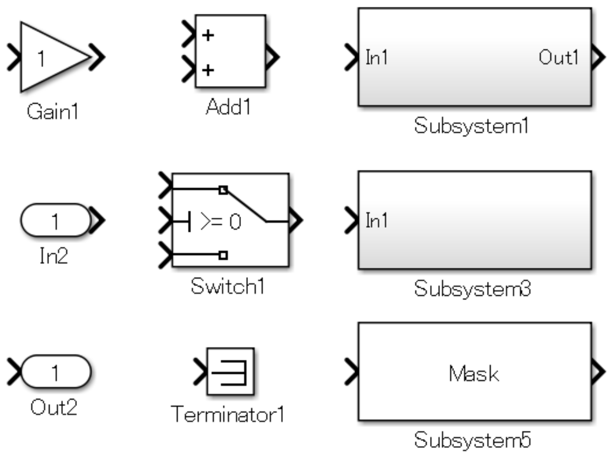
See Also
· Sub ID a, see MISRA AC SLSF 026E

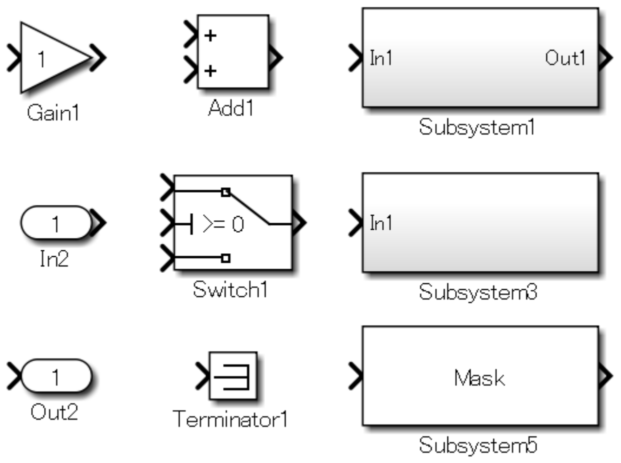
jc_0603: Model description

Rule ID: Title	**jc_0603: Model description**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The model layer shall include a description of the layer. The layers that require descriptions are defined (by function and layer type) in the project.	Description object (Block type, etc.) Layer being described
	【Correct】 Model layer includes a description.  【Requirement】 Performs an increment process on the input signal In1 1 In1 In1 1 Constant + + Add Out1 1 Out1 【Incorrect】 The layer does not include a description.  1 In1 In1 1 Constant + + Add Out1 1 Out1	

b	The format of the layer description shall be consistent in the model.	Model description format
Rationale		
Sub ID	**Description**	
a	・When a description is not included, the readability of the control specifications is reduced. Usability, maintainability, and portability also decreases.	
b	・Readability is impaired when the description format is not consistent.	
See Also		
・Sub ID a and b, see MISRA AC SLSF 022		

jc_0604: Using Block Shadow

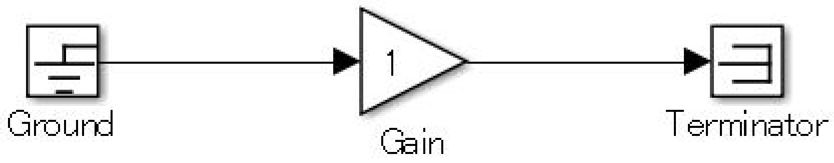
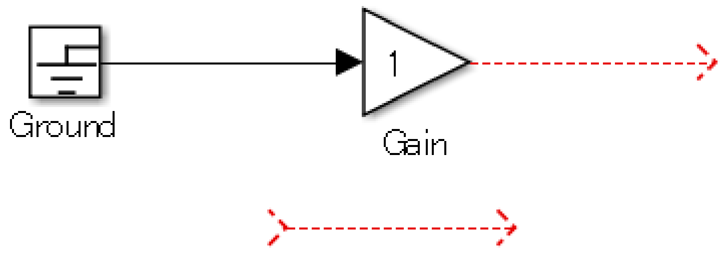
Rule ID: Title	**jc_0604: Using Block Shadow**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Block format property {Shadow} shall not be selected.	-
	【Correct】 A drop shadow is not applied to the blocks.  1 Gain1 + + Add1 In1 Out1 Subsystem1 1 In2 >= 0 Switch1 In1 Subsystem3 1 Out2 Terminator1 Mask Subsystem5 【Incorrect】 The block has a drop shadow.	

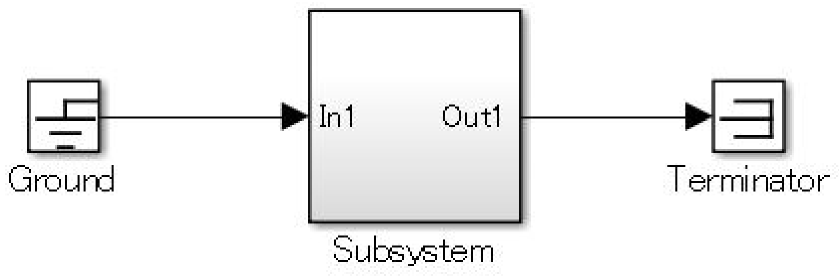
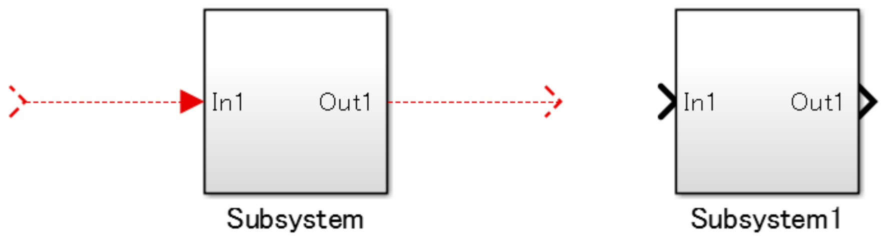
	 1 Gain1 + + Add1 In1 Out1 Subsystem1 1 In2 >= 0 Switch1 In1 Subsystem3 1 Out2 Terminator1 Mask Subsystem5

Rationale	
Sub ID	**Description**
a	・Difficult to determine if a port exists because it is hidden by the shading, which impairs readability.

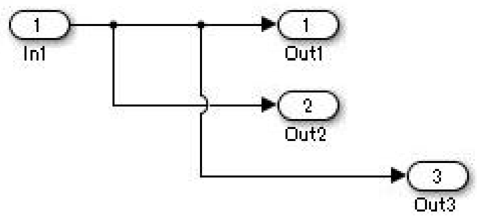
See Also
・Sub ID a, see MISRA AC SLSF 024A

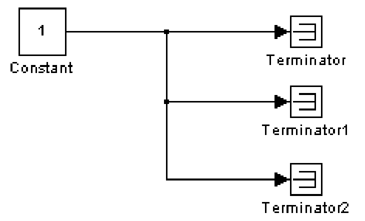
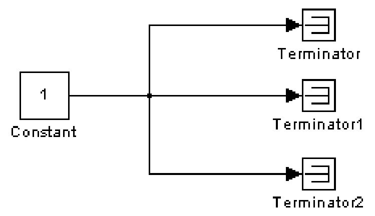
db_0081: Unconnected signals / blocks

Rule ID: Title	**db_0081: Unconnected signals / blocks**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The model shall not have signal lines that are not connected.	-
	【Correct】  【Incorrect】 	

b	The model shall not have subsystems or basic blocks that are not connected.	-
	【Correct】  【Incorrect】 	
Rationale		
Sub ID	**Description**	
ab	・Unconnected lines can have adverse effects, such as simulation errors or failure to generate code.	

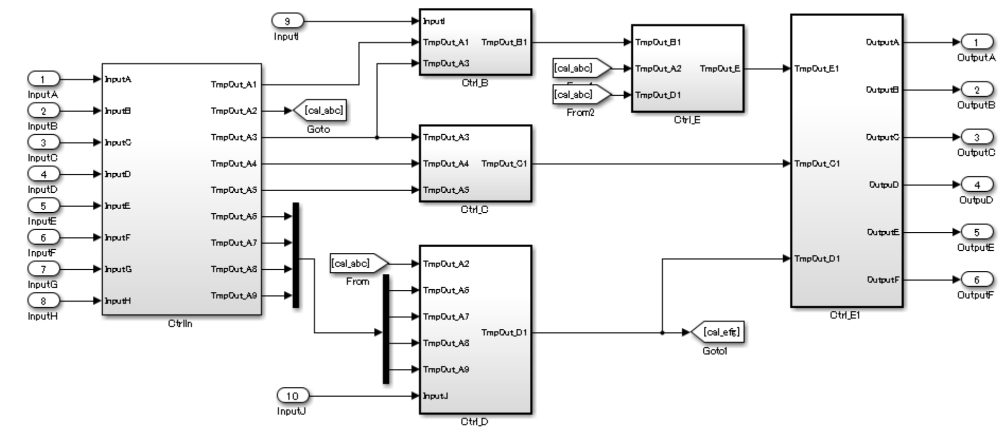
db_0032: Signal line connections

Rule ID: Title	**db_0032: Signal line connections**	
Sub ID Recommendations	NA-MAAB: a1/a2, b, c, e JMAAB: a1/a2, b, c, d, e	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	Vertical and horizontal signal lines shall not cross over one another.	-
a2	(R2014a and later) When vertical and horizontal signal lines must cross, Simulink editor preference {Line crossing style} shall be set to “Line hop”.	-
	【Correct】 The vertical line hops over the horizontal line. 	

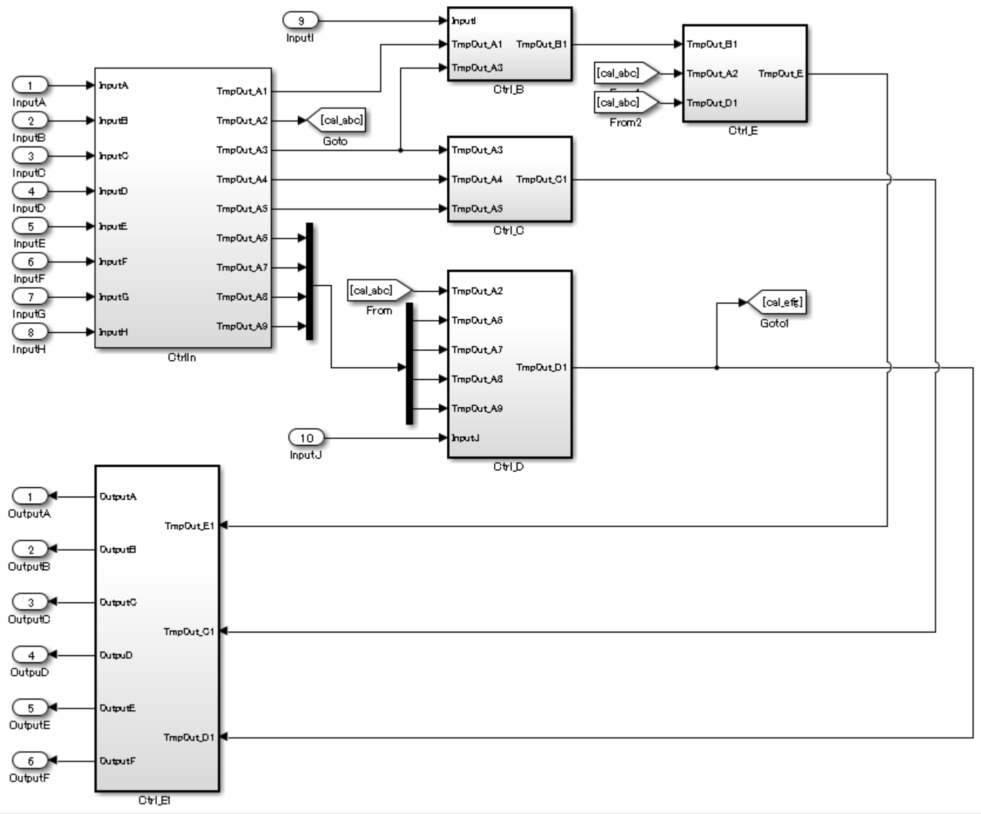
b	Signal lines shall not overlap with other signal lines.	-
c	Signal lines shall not cross over blocks.	-
d	Signal lines shall not split into more than two sub lines at a single branching point.	-
	【Correct】  【Incorrect】  	
e	Signal lines shall be resized vertically or horizontally as required for the model layout.	-
Rationale		
Sub ID	**Description**	
a1	· Difficult to understand the relationships between blocks when signal lines cross.	
A2	· In R2014a and later, the difference between crossing and branching is clarified.	
B	· Difficult to understand the relationships between blocks when signal lines overlap..	
C	· Difficult to understand the relationships between blocks when signal lines cross.	
D	· Difficult to understand the relationships between blocks.	
E	· Consistent application of signal lines improves readability.	

db_0141: Signal flow in Simulink models

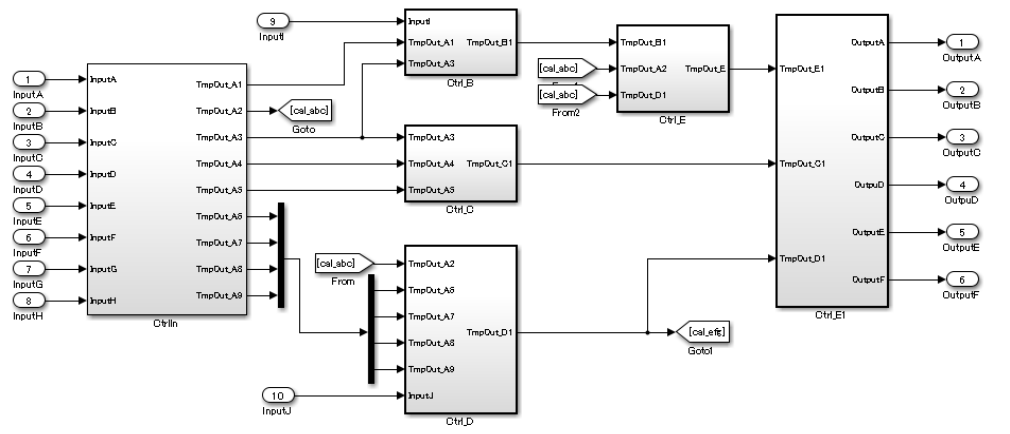
Rule ID: Title	**db_0141: Signal flow in Simulink models**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Signals shall flow from left to right. Exception: Feedback loops can flow from right to left.	-
	【Correct】	



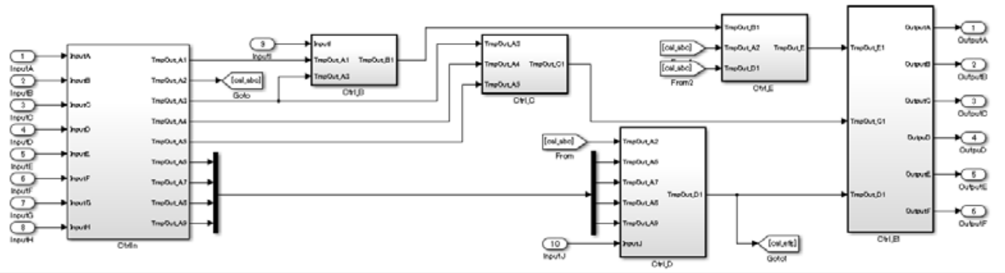
【Incorrect】



b	Parallel blocks or subsystems shall be arranged from top to bottom.	-
	【Correct】	

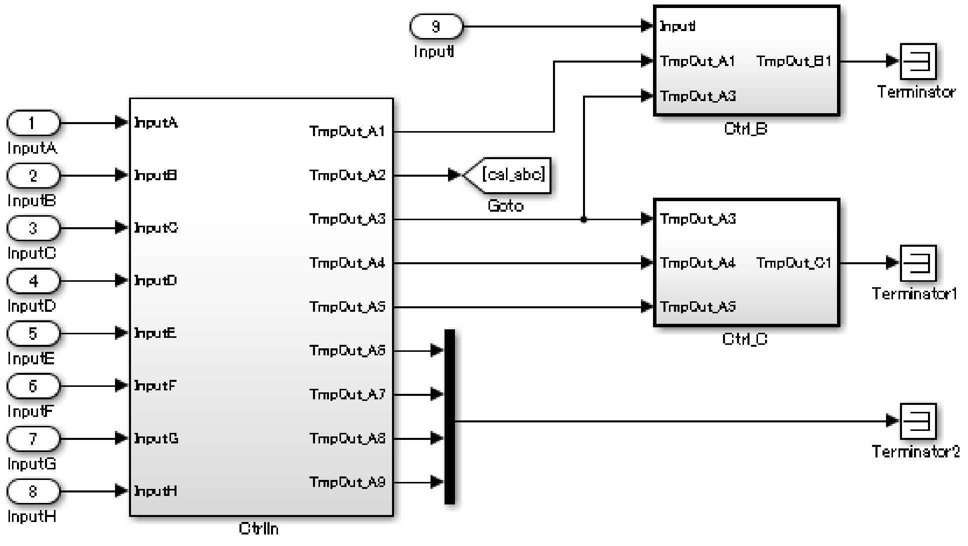


【Incorrect】

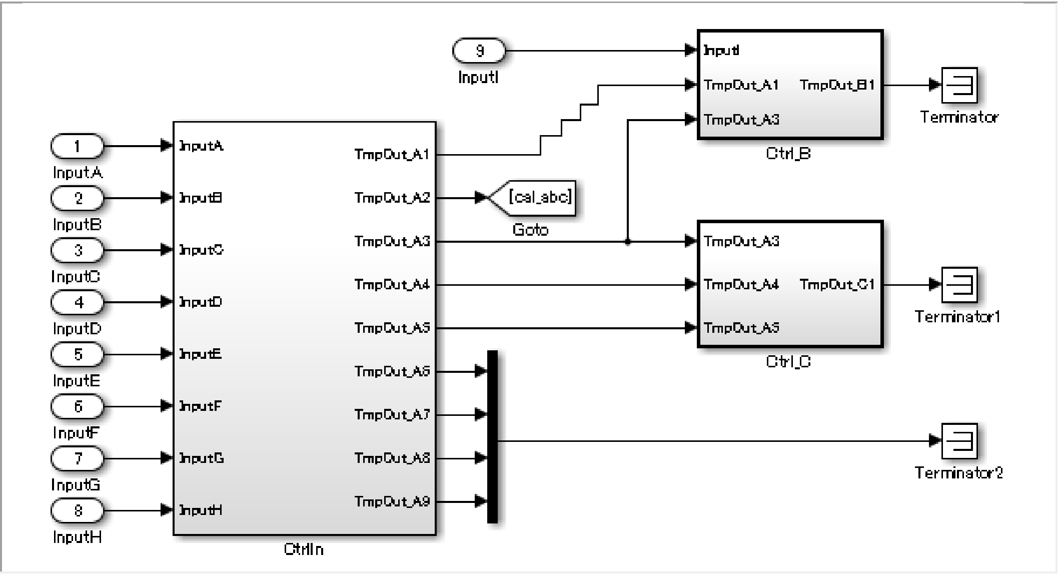


c	Signal lines shall not bend multiple times unnecessarily.	-

【Correct】

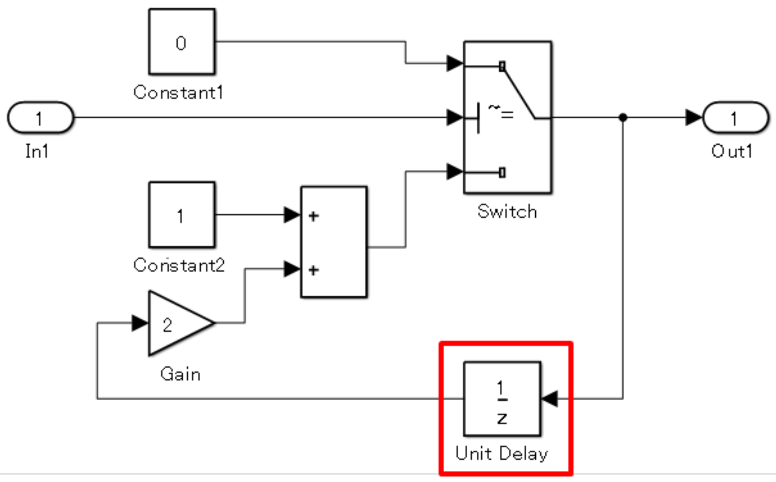


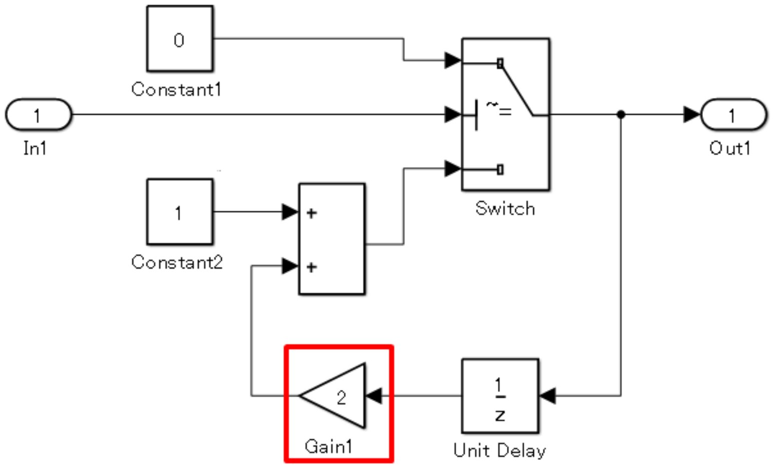
【Incorrect】



Rationale	
Sub ID	**Description**
abc	・ Deviation from the rules can impair readability.

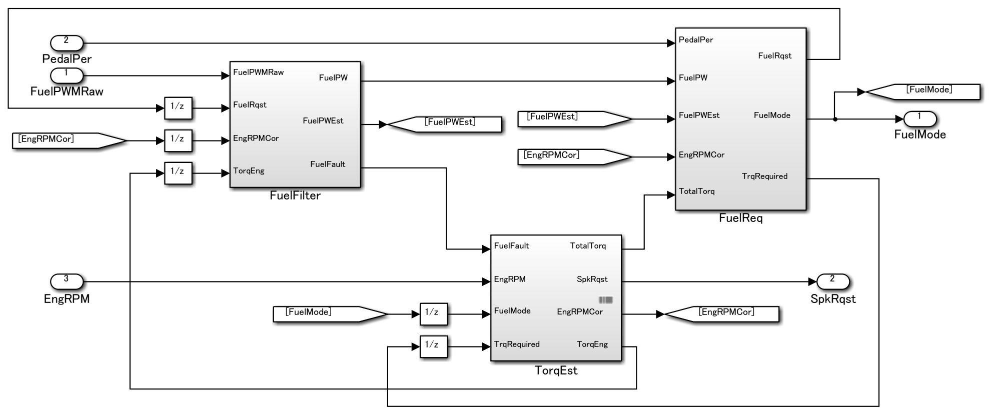
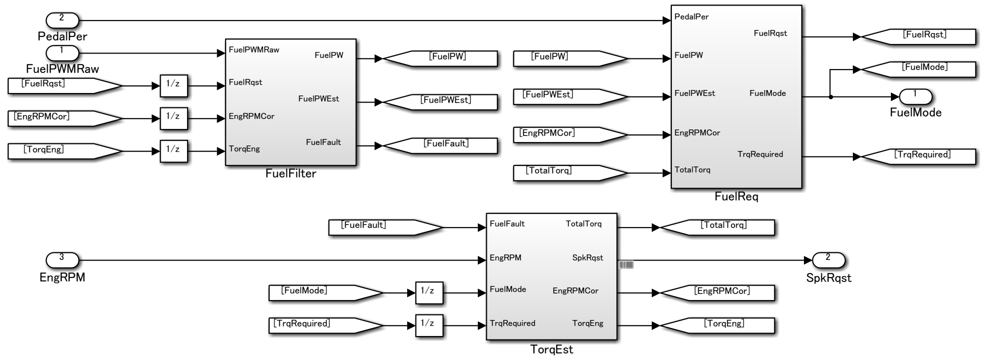
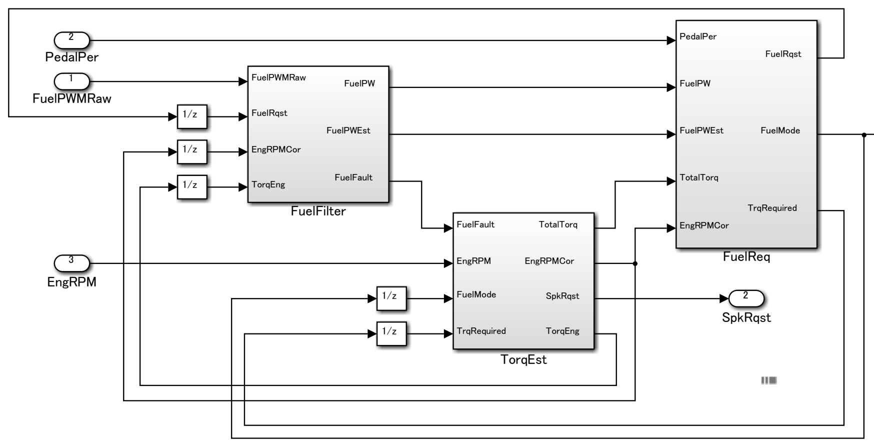
jc_0110: Direction of block

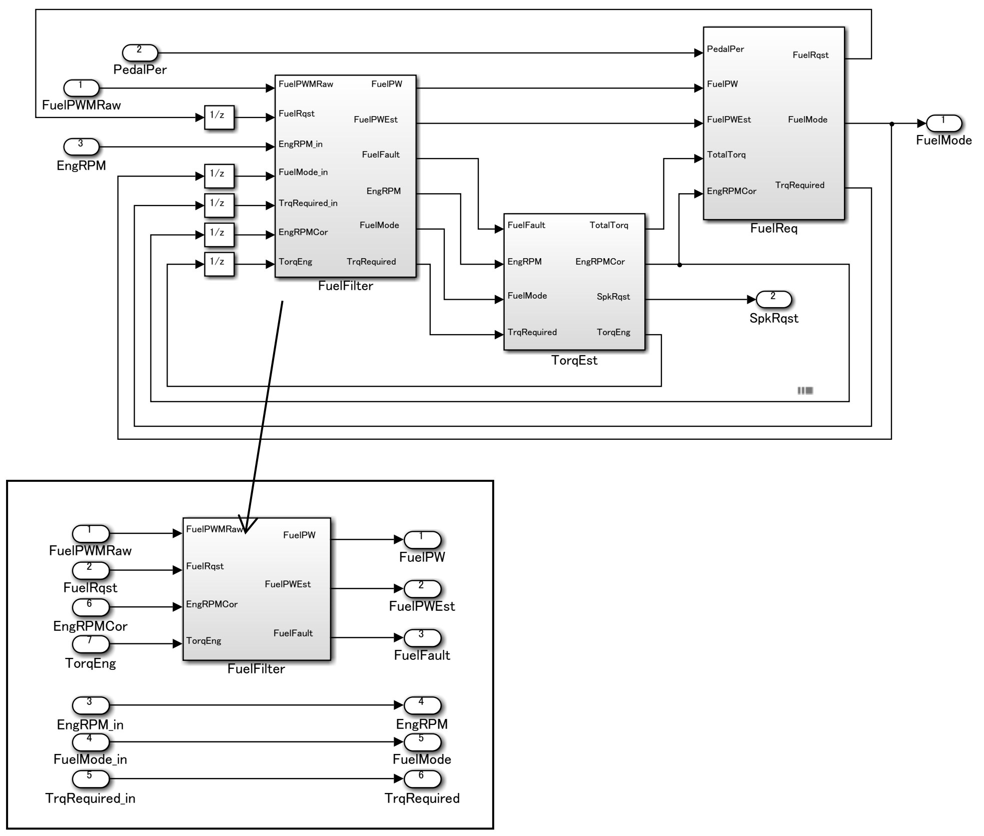
Rule ID: Title	**jc_0110: Direction of block**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Blocks shall be arranged so the output is to the right. Exception: When [Delay] is used in a feedback loop, the output can be to the left.	-
	【Correct】 The block is arranged so that the output is to the right. The output of [Delay] is to the left.  0 Constant1 1 In1 Switch 1 Out1 1 Constant2 + + 2 Gain 1/z Unit Delay	

	【Incorrect】 The block is arranged so the output is to the left.  0 Constant1 1 In1 ~= 1 Out1 Switch 1 Constant2 + + 2 Gain1 1 z Unit Delay
Rationale	
Sub ID	**Description**
a	Signal flow can be difficult to understand if the direction of the signals is not consistent.

jc_0171: Clarification of connections between structural subsystems

Rule ID: Title	**jc_0171: Clarification of connections between structural subsystems**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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] to create buses or connect signals to [Merge].	-
	【Correct】	

	 PedalPer FuelPWMRaw EngRPM FuelFilter FuelReq TorqEst FuelMode SpkRqst [FuelMode] [FuelPWEst] [EngRPMCor] 1/z 【Incorrect】  PedalPer FuelPWMRaw EngRPM FuelFilter FuelReq TorqEst FuelMode SpkRqst [FuelRqst] [EngRPMCor] [TorqEng] [FuelPW] [FuelPWEst] [FuelFault] [TotalTorq] [FuelMode] [TrqRequired] 1/z	
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.	-
	【Correct】  PedalPer FuelPWMRaw EngRPM FuelFilter FuelReq TorqEst FuelMode SpkRqst 1/z 【Incorrect】 Signals that are not used in the subsystem are connected to avoid crossing of signal lines.	

	 PedalPer FuelPWMRaw EngRPM 1/z FuelFilter TorqEst FuelReq FuelMode SpkRqst FuelPW FuelRqst FuelPWEst FuelFault EngRPMCor TorqEng EngRPM_in FuelMode_in TrqRequired_in TrqRequired TotalTorq
Rationale	
Sub ID	**Description**
a	· Clarifies structural subsystem connections and execution order.
B	· Eliminating unnecessary connections clarifies the relationship between connections. · Deviation from the rule can cause to confusion due to unused input/output signals.

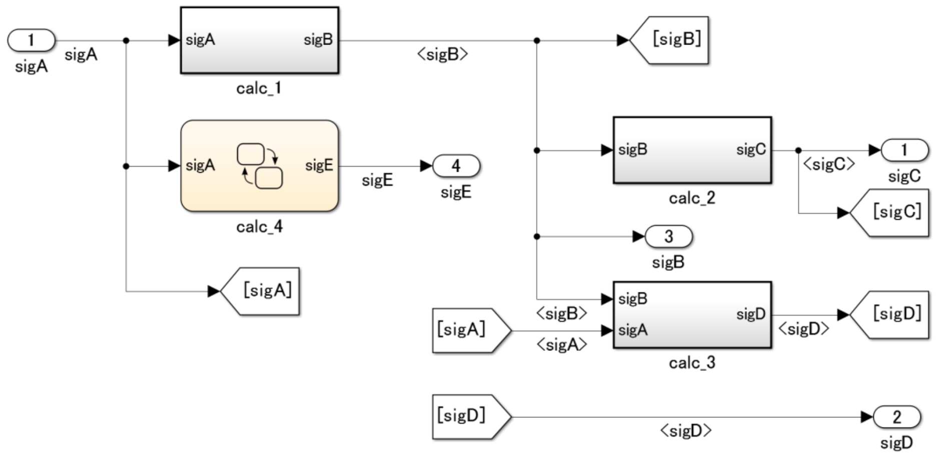
jc_0602: Consistency in model element names

Rule ID: Title	**jc_0602: Consistency in model element names**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	These names shall match when they are directly connected by using signal lines. · [Inport] block name · [Outport] block name · Structural subsystem input port label name · Structural subsystem output port label name · [From] tag name · [Goto] tag name · Signal line signal name Exception 1: A signal line that connects to one of the following	-

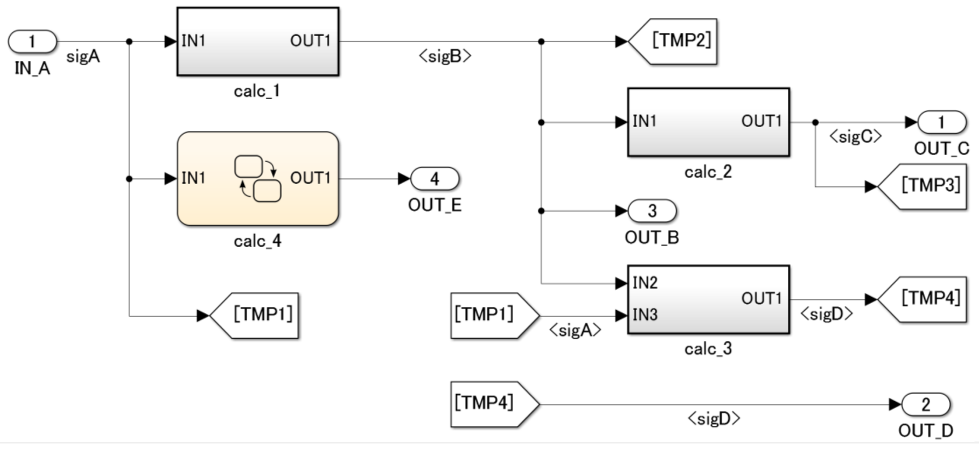
subsystem types can have a name that differs from that of the subsystem port label:
・Subsystems linked to a library
・Reusable subsystems

Exception 2:
When a combination of [Inport], [Outport], and other blocks have the same name, use a suffix or prefix for the [Inport] and [Outport] blocks. Any prefix or suffix can be used for ports, but they must be consistent. For example, [Inport] uses “in” and [Outport] uses “out”.
Note: [Inport] and [Outport] names and signal names must have different names.

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



【Incorrect】
Names of model elements that connect directly to signal lines are inconsistent.



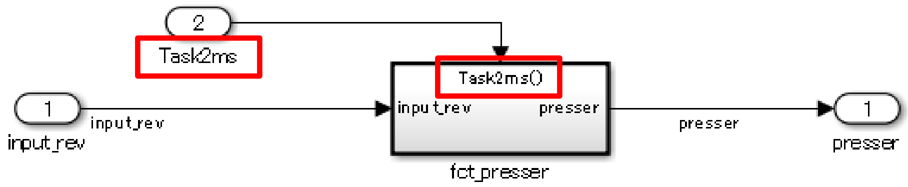
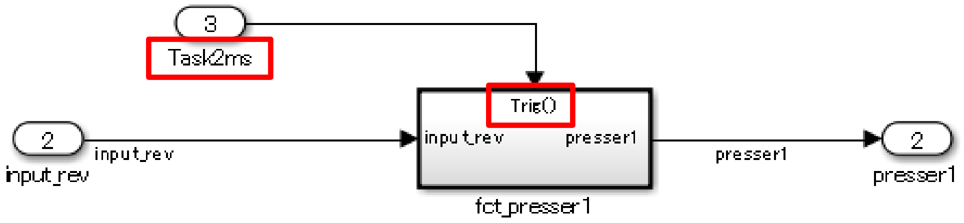
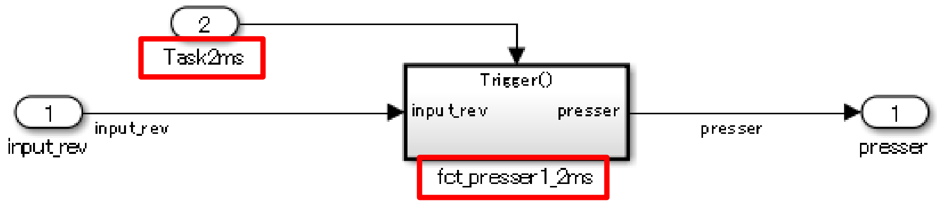
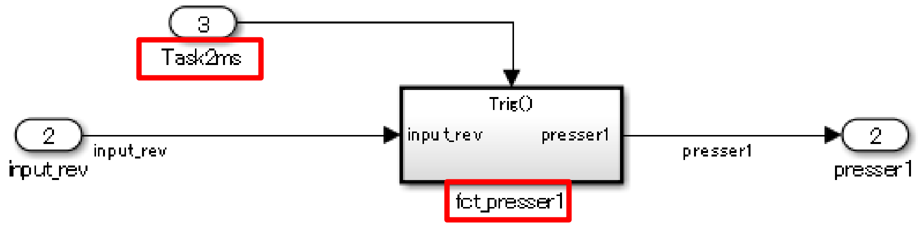
Rationale

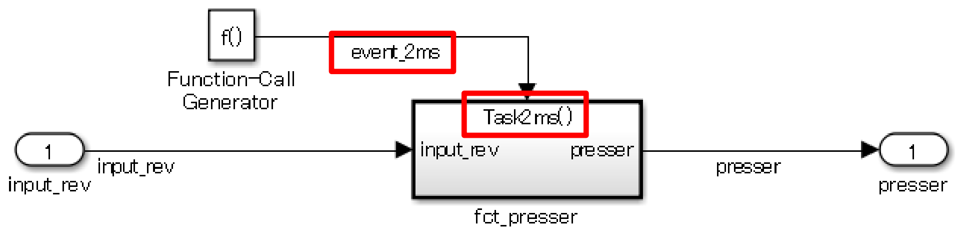
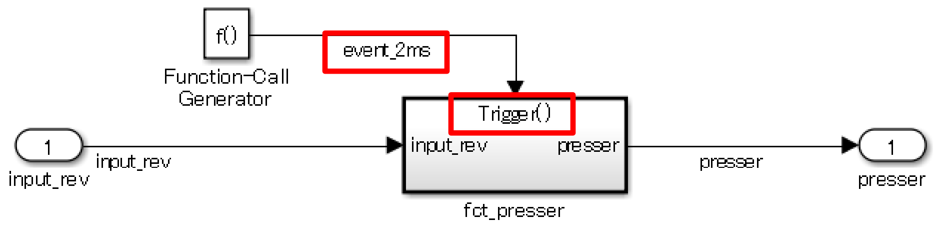
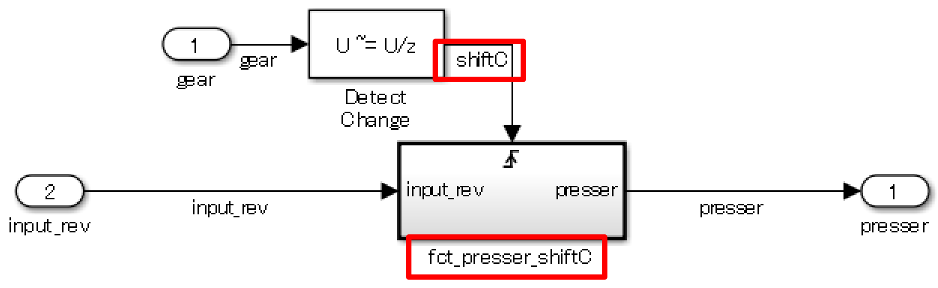
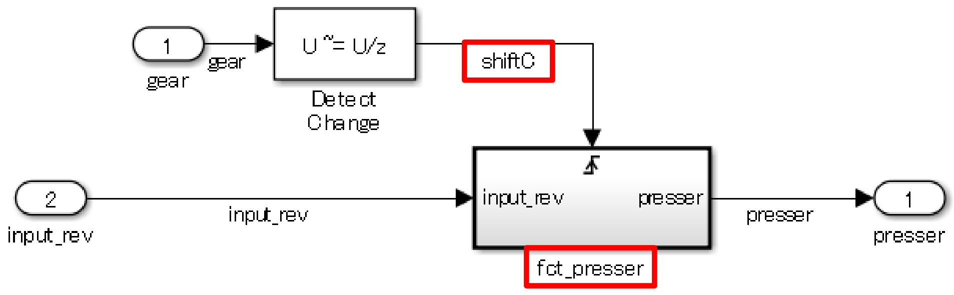
Sub ID	**Description**
a	・Prevent misconnected signal lines. ・Readability is impaired. ・Deviation from the rule can make it difficult to maintain the integrity of the model and code.

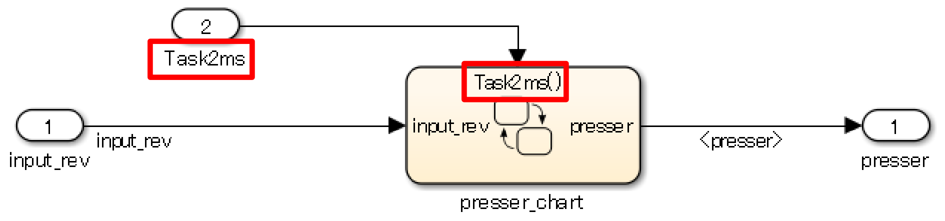
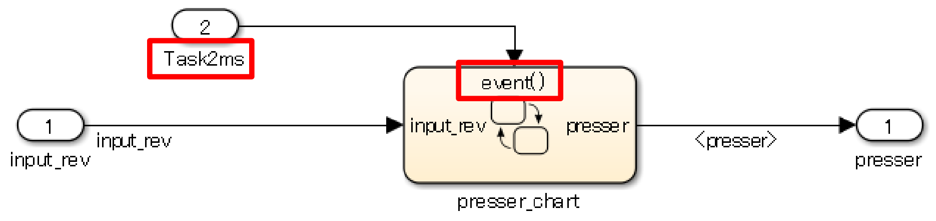
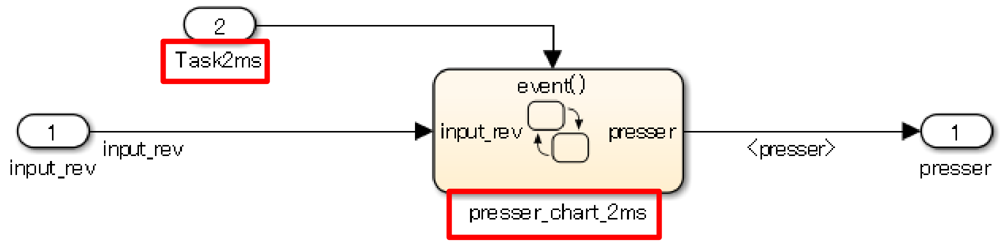
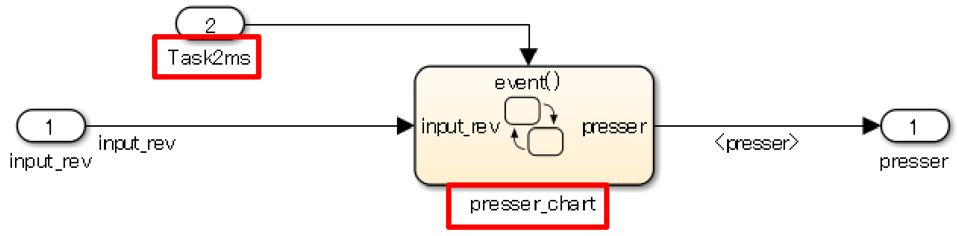
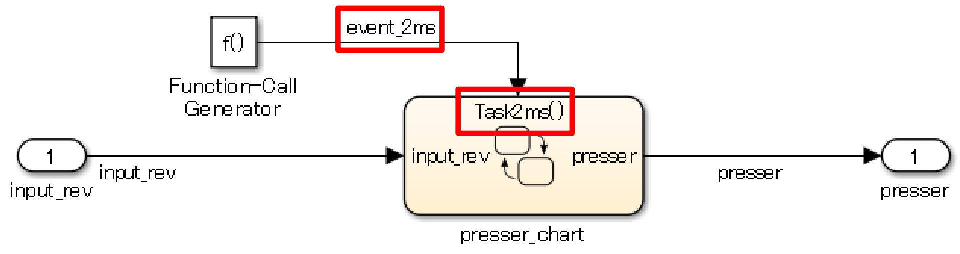
See Also

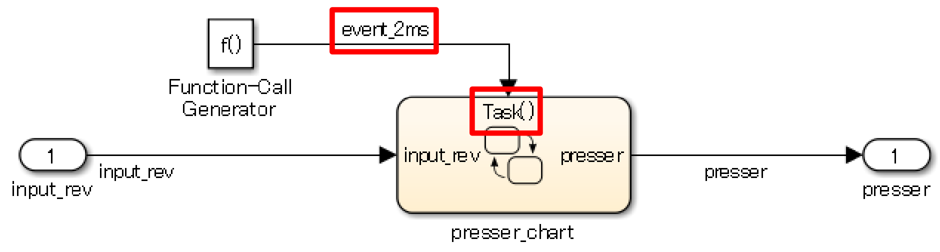
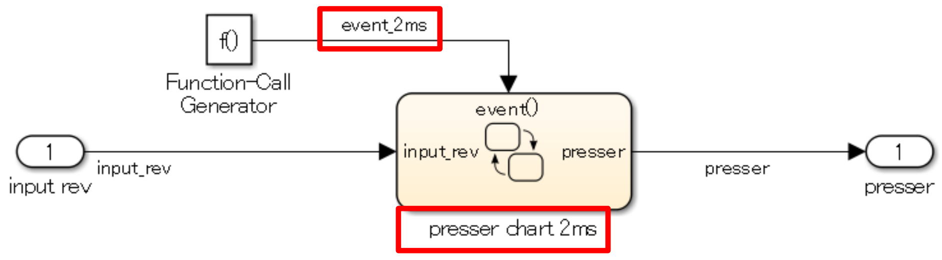
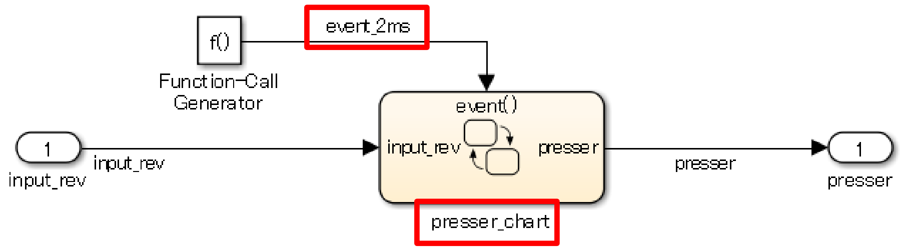
・Sub ID a, see MISRA AC SLSF 036C

jc_0281: Trigger signal names

Rule ID: Title	**jc_0281: Trigger signal names**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a1/a2/a3/a4, b1/b2/b3/b4	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	The name of the conditional input block at the destination shall include the name of the block at the origin of the trigger signal	-
	【Correct】  【Incorrect】 	
a2	The name of the conditional subsystem at the destination shall include the name of the block at the origin of the trigger signal.	-
	【Correct】  【Incorrect】 	

a3	The name of the conditional input block at the destination shall include the name of the trigger signal.	-
	【Correct】  f() Function-Call Generator event_2ms Task2ms() 1 input_rev input_rev input_rev presser fct_presser presser 1 presser 【Incorrect】  f() Function-Call Generator event_2ms Trigger() 1 input_rev input_rev input_rev presser fct_presser presser 1 presser	
a4	The name of the conditional subsystem at the destination shall include the name of the trigger signal.	-
	【Correct】  1 gear gear U ~= U/z Detect Change shiftC 2 input_rev input_rev input_rev presser fct_presser_shiftC presser 1 presser 【Incorrect】  1 gear gear U ~= U/z Detect Change shiftC 2 input_rev input_rev input_rev presser fct_presser presser 1 presser	
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.	-
	【Correct】	

	 2 Task2ms Task2ms() 1 input_rev input_rev input_rev presser presser_chart <presser> 1 presser 【Incorrect】  2 Task2ms event() 1 input_rev input_rev input_rev presser presser_chart <presser> 1 presser	
b2	The name of [Chart] at the destination shall include the name of the block at the origin of the trigger signal.	-
	【Correct】  2 Task2ms event() 1 input_rev input_rev input_rev presser presser_chart_2ms <presser> 1 presser 【Incorrect】  2 Task2ms event() 1 input_rev input_rev input_rev presser presser_chart <presser> 1 presser	
b3	The name of the Stateflow block event at the destination shall include the name of the trigger signal.	-
	【Correct】  f() Function-Call Generator event_2ms Task2ms() 1 input_rev input_rev input_rev presser presser_chart presser 1 presser	

	【Incorrect】  event_2ms f() Function-Call Generator Task() 1 input_rev input_rev input_rev presser presser 1 presser presser_chart	
b4	The name of the trigger signal and the [Chart] name at the destination must include the same name. The name of [Chart] at the destination shall include the name of the trigger signal.	-
	【Correct】  event_2ms f() Function-Call Generator event() 1 input_rev input rev input_rev presser presser 1 presser presser chart 2ms 【Incorrect】  event_2ms f() Function-Call Generator event() 1 input_rev input_rev input_rev presser presser 1 presser presser_chart	

Rationale	
Sub ID	**Description**
a1a2a3 a4b1b2 b3b4	・Reduces connection mistakes. ・Increases understanding of the relationship between the origin of the trigger signal and the destination.
・See Also	
・Sub ID a1, a2, a3, a4, see MISRA AC SLSF 026C	

db_0143: Usable block types in model hierarchy

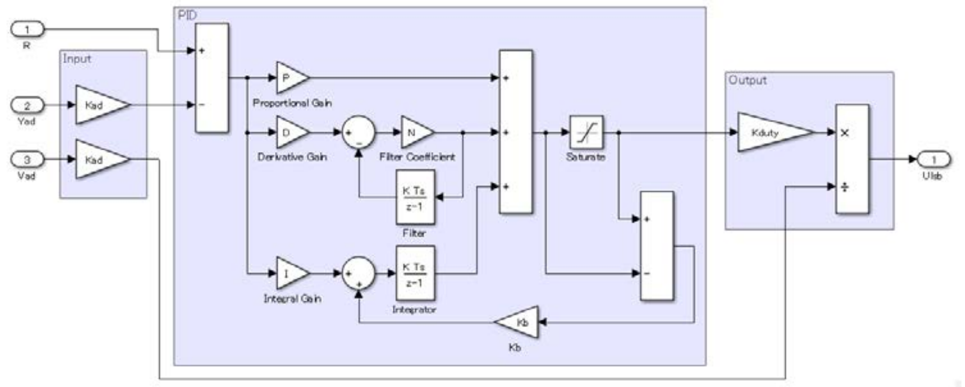
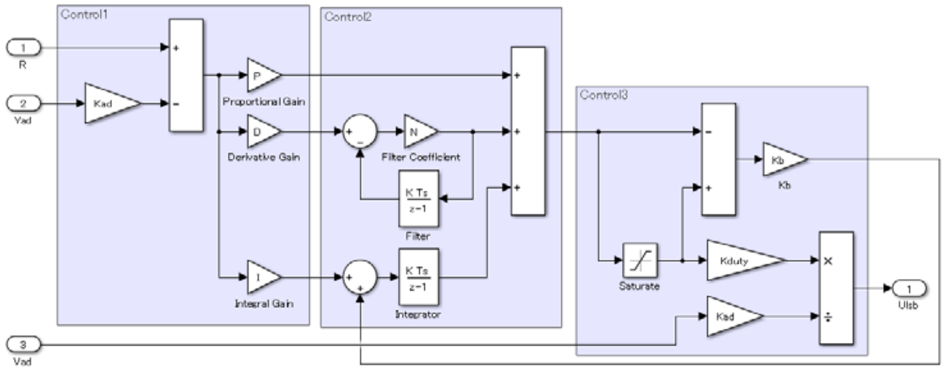
Rule ID: Title	**db_0143: Usable block types in model hierarchy**
Sub ID Recommendations	NA-MAAB: a JMAAB: a
MATLAB® Version	All

Rule		
Sub ID	**Description**	**Custom Parameter**
a	Model levels shall use only the block types that are defined for the layer type.	Layer type Block type

	For information on layer types, see Appendix 8.2 - Hierarchical Structure of a Controller Model. Clearly defined layer types restrict the number of blocks that can be used. Block restrictions: ・ (R2011a and earlier) [Enable] 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]	
Rationale		
Sub ID	**Description**	
a	Readability is impaired when subsystems and basic blocks are used in the same layer.	

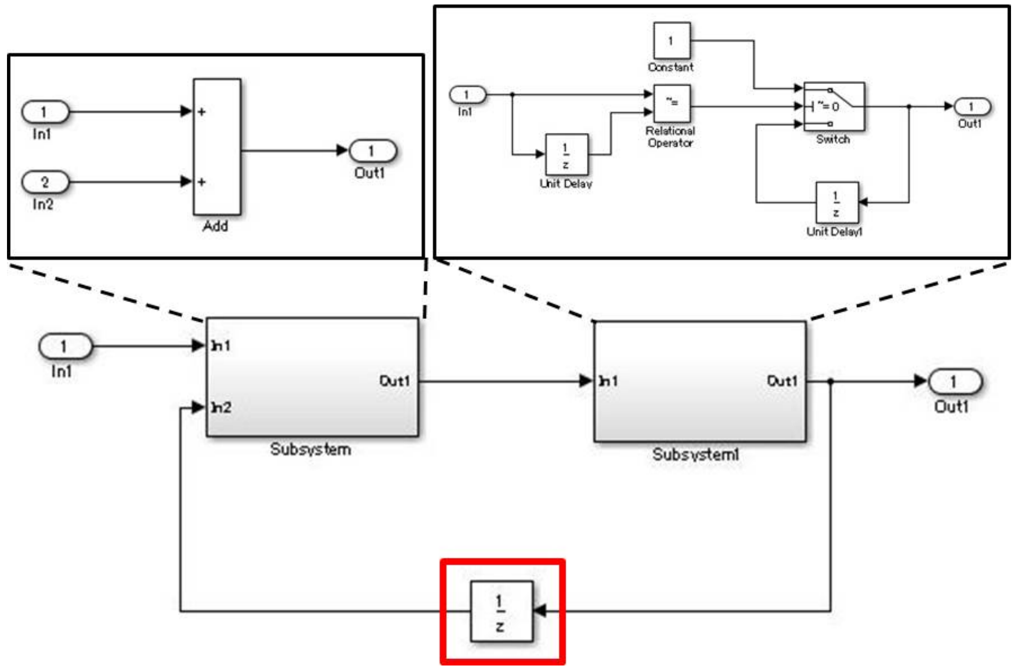
db_0144: Use of subsystems

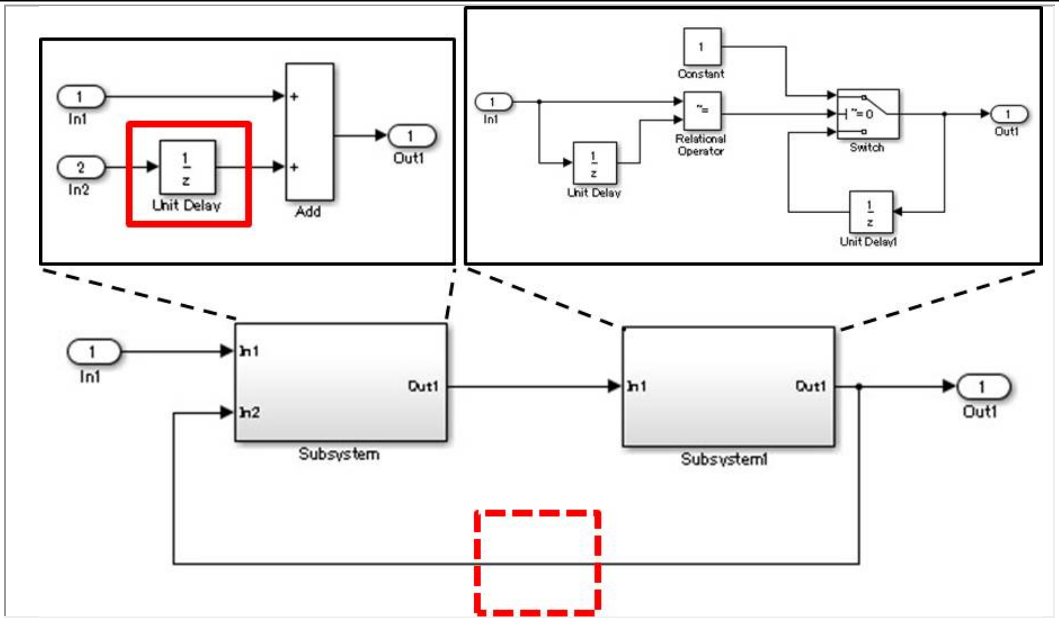
Rule ID: Title	**db_0144: Use of subsystems**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Blocks in a Simulink diagram shall be grouped together into subsystems based on functional decomposition of	-

	the algorithm, or portion thereof, represented in the diagram. 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. Blocks can also be grouped together based on behavioral variants or timing. When implementing a subsystem to alleviate readability issues, use a virtual subsystem.	
	【Correct】 Subsystems are divided by functional unit.  PID Input Output R Yad Vad Kad Kad P Proportional Gain D Derivative Gain N Filter Coefficient Filter I Integral Gain Integrator Saturate Kb Kduty Ulsb 【Incorrect】 Subsystems are not divided by functional unit.  Control1 Control2 Control3 R Yad Vad Kad P Proportional Gain D Derivative Gain I Integral Gain N Filter Coefficient Filter Integrator Kb Saturate Kduty Kad Ulsb	
b	A virtual subsystem shall be used when processing order and code generation does not need to be taken into consideration.	-

Rationale	
Sub ID	**Description**
a	・Avoid grouping blocks into subsystems primarily for the purpose of saving space in the diagram. ・It can be difficult to reuse the subsystem.
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.

jc_0653: Delay block layout in feedback loops

Rule ID: Title	**jc_0653: Delay block layout in feedback loops**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Delay] in feedback loops across subsystems shall reside in the hierarchy that describes the feedback loop.	-
	【Correct】 [Delay] resides in the hierarchy that describes the feedback loop.  【Incorrect】 [Delay] resides in a subsystem that is nested within the hierarchy which describes the feedback loop.	

Rationale	
Sub ID	**Description**
a	· Prevents double placement of [Delay]. · Clarifying the extent of diversion improves reusability. · Improves testability; it is difficult to test a subsystem that contains [Delay] on its own because past values cannot be entered directly.

hd_0001: Prohibited Simulink sinks

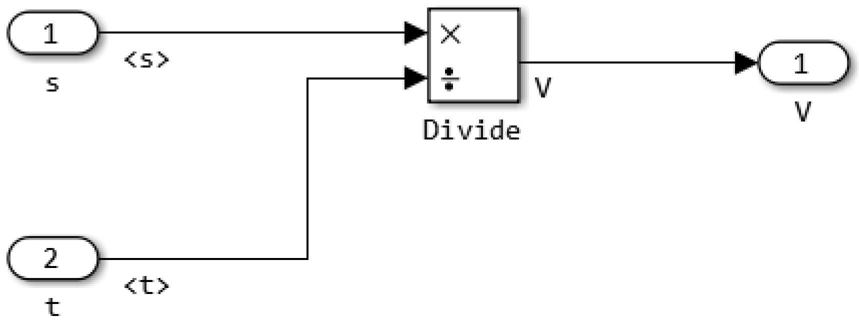
Rule ID: Title	**hd_0001: Prohibited Simulink sinks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Control algorithm models shall be designed from discrete blocks. [Scope] and [Display] can be used in the model diagram. These sink blocks shall not be used: • [To File] • [To Workspace] • [Stop Simulation] Consider using signal logging and the Signal and Scope 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.	-
Rationale		
Sub ID	**Description**	
a	· Improves readability and model simulation. · Code generation may not be possible.	

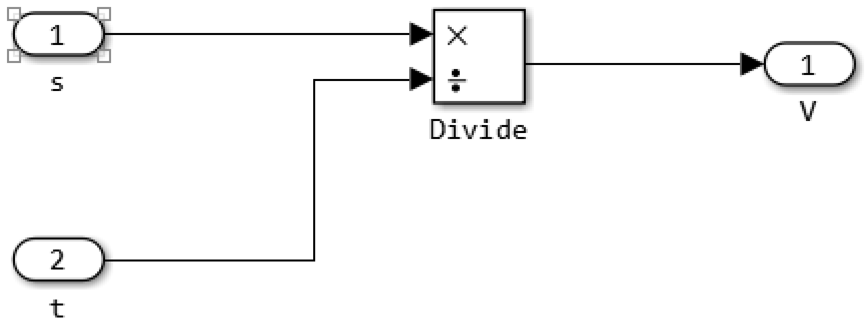
3.3. Signal

na_0010: Usage of vector and bus signals

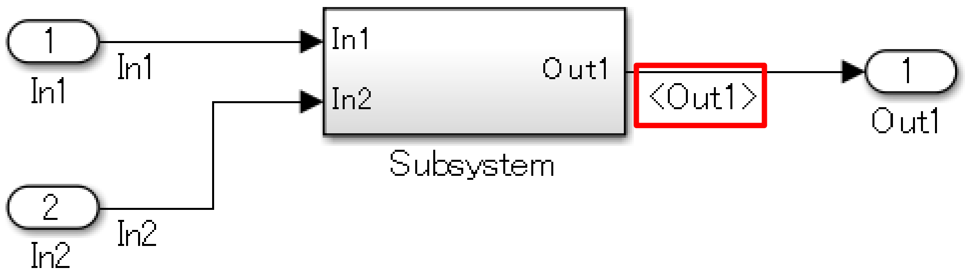
Rule ID: Title	na_0010: Usage of vector and bus signals	
Sub ID Recommendations	NA-MAAB: a, b, c, d JMAAB: a, b, c, d	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Mux] and [Demux] blocks shall be used when generating and decomposing vectors.	-
b	[Mux] inputs shall be scalars and vectors.	-
c	[BusCreator] and [BusSelector] shall be used when generating and decomposing busses.	-
d	Busses shall connect to blocks that support busses.	-
Rationale		
Sub ID	**Description**	
abcd	· Prevents issues that are caused by combining vector and bus signals. See "Preventing the mixing of busses and Mux" for additional information.	
See Also		
· Sub ID a, b, c, d, see MISRA AC SLSF 015A,B		

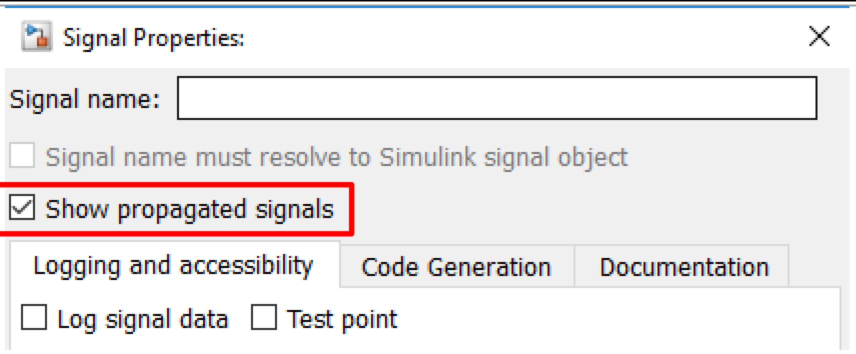
jc_0008: Definition of signal names

Rule ID: Title	jc_0008: Definition of signal names	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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. Important blocks: An important block is defined by the system input and output of meaningful results, not by its type.	Definition of an important block
	【Correct】 1 s <s> × ÷ Divide V 1 V 2 t <t>	

	【Incorrect】 
Rationale	
Sub ID	**Description**
a	・Defining the signal name and displaying the label for the output of meaningful results from important blocks improves the readability of the model.

jc_0009: Signal name propagation

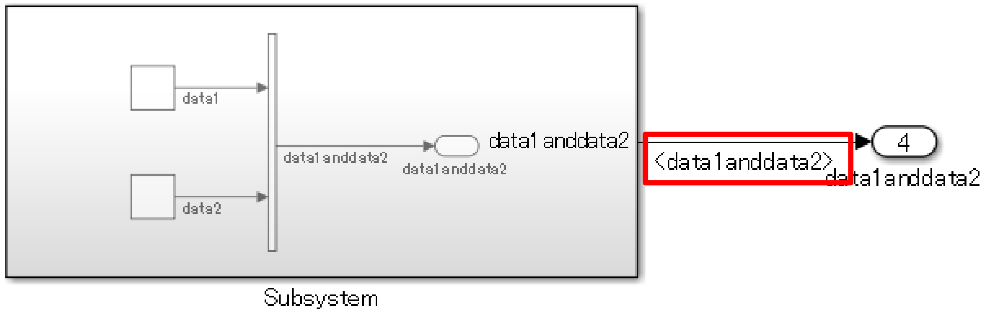
Rule ID: Title	**jc_0009: Signal name propagation**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When defining the signal name for a signal that extends across a hierarchy, signal property {Show propagated signals} 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 ・A signal name is not set at the [Bus Creator] outport signal.	-
	【Correct】 {Show propagated signals} is selected, displaying the propagated signal names.  1 In1 In1 2 In2 In2 In1 In2 Out1 Subsystem <Out1> 1 Out1	


Signal Properties:

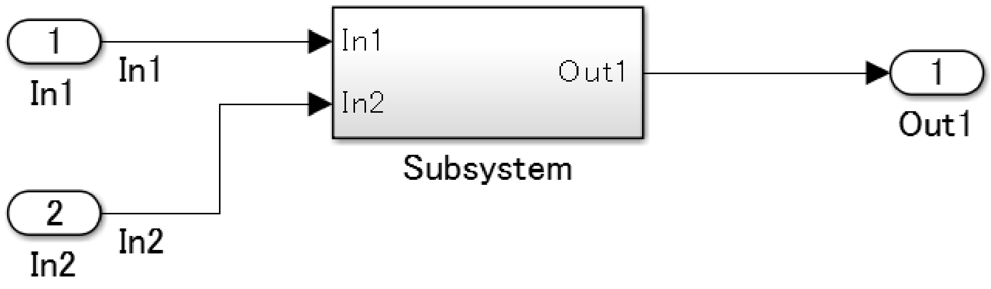
Signal name:

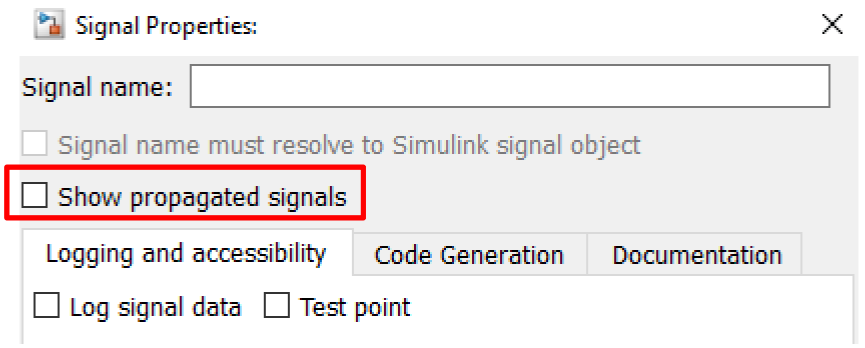
☐ Signal name must resolve to Simulink signal object

☑ Show propagated signals



【Incorrect】
{Show propagated signals} is not selected, therefore signal names are not displayed.



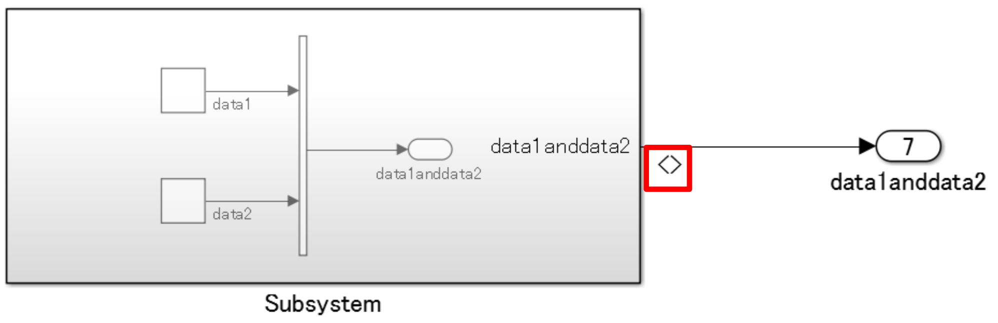
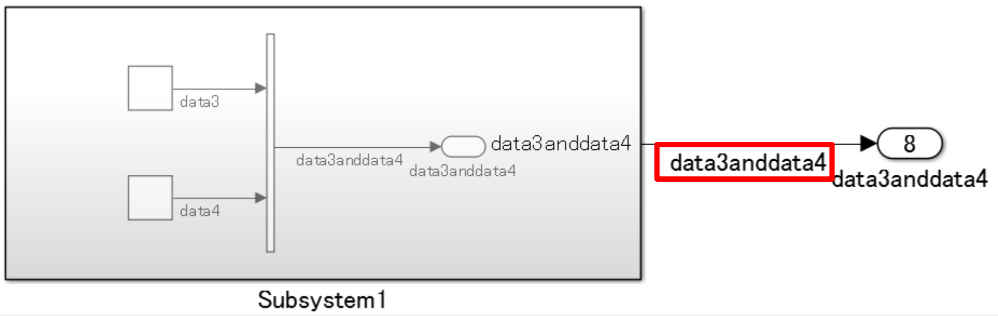
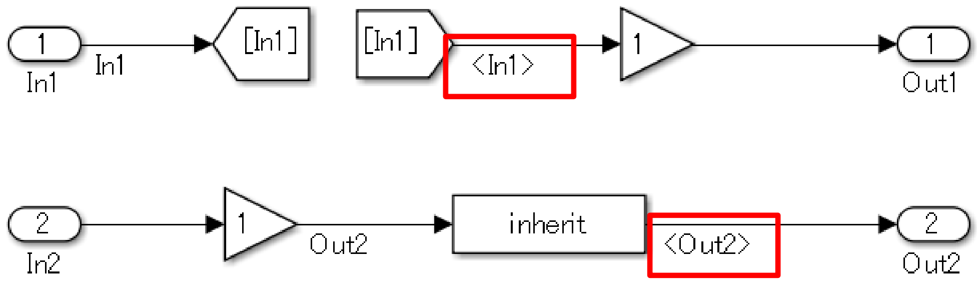

Signal Properties:

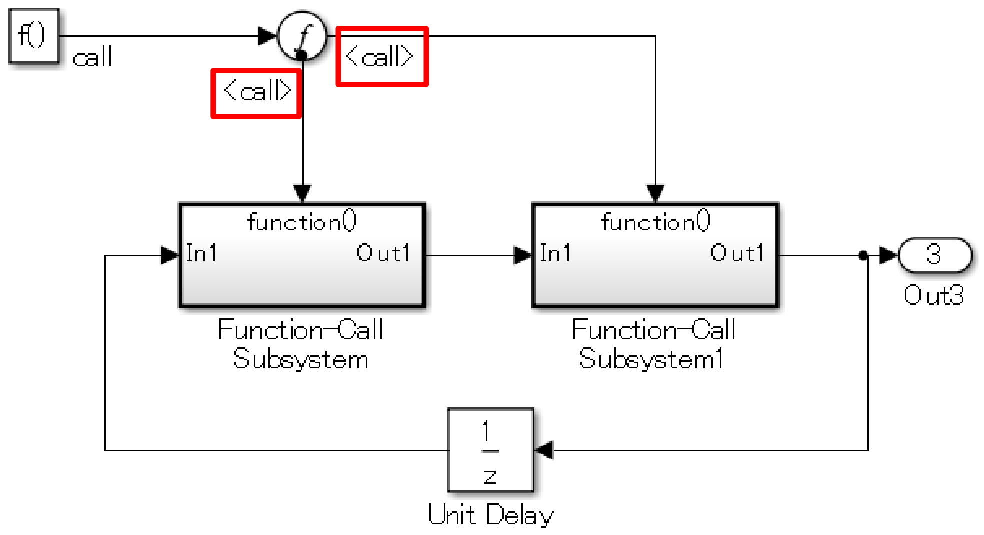
Signal name:

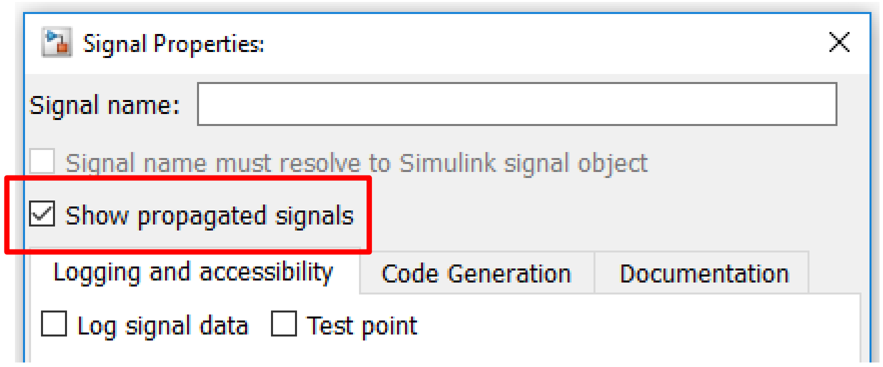
☐ Signal name must resolve to Simulink signal object

☐ Show propagated signals

Signals that connect to [Bus Creator] and [Outport] do not have names, but {Show propagated signals} is selected for signals that connect to [Subsystem] and [Outport].

	 data1 data2 data1anddata2 data1anddata2 7 data1anddata2 Subsystem Signals that connect to [Bus Creator] and [Outport] have names, but signals that connect to [Subsystem] and [Outport] also have names.  data3 data4 data3anddata4 data3anddata4 data3anddata4 data3anddata4 8 data3anddata4 Subsystem1	
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]	-
	【Correct】 {Show propagated signals} is selected, displaying the propagated signal names.  1 In1 In1 [In1] [In1] <In1> 1 1 Out1 2 In2 1 Out2 inherit <Out2> 2 Out2	




Signal Properties:

Signal name:

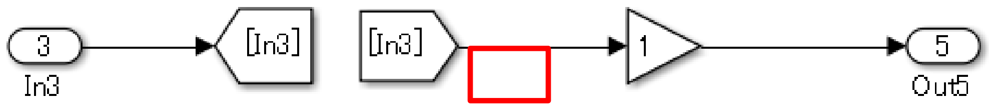
☐ Signal name must resolve to Simulink signal object

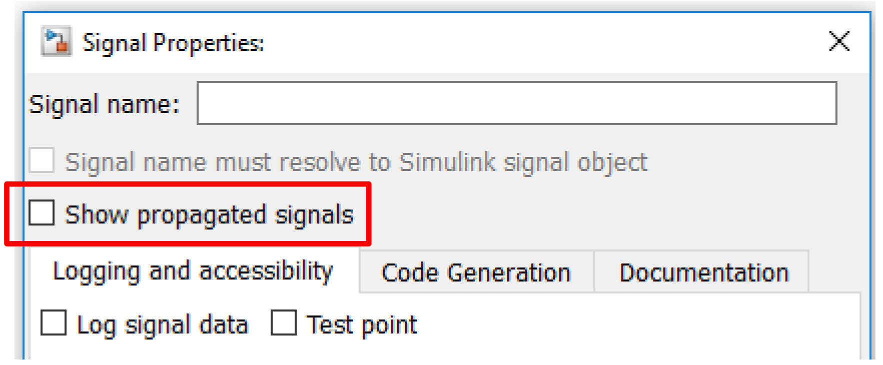
☑ Show propagated signals

Logging and accessibility | Code Generation | Documentation

☐ Log signal data ☐ Test point

Signals that connect to [Inport] and [Goto] do not have names, therefore {Show propagated signals} does not need to be selected.




Signal Properties:

Signal name:

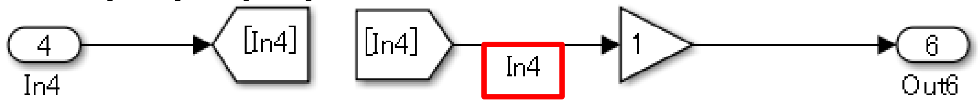
☐ Signal name must resolve to Simulink signal object

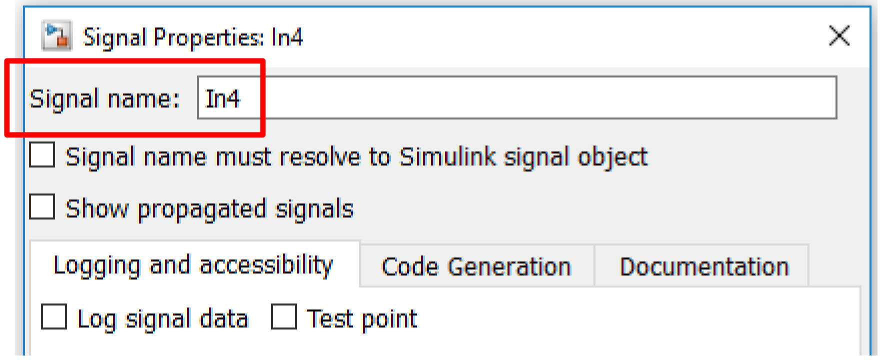
☐ Show propagated signals

Logging and accessibility | Code Generation | Documentation

☐ Log signal data ☐ Test point

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




Signal Properties: In4

Signal name: In4

☐ Signal name must resolve to Simulink signal object

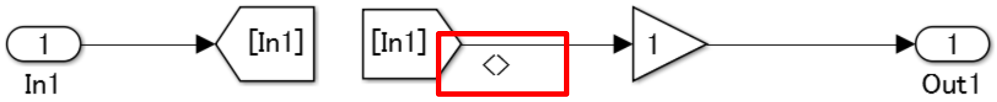
☐ Show propagated signals

Logging and accessibility | Code Generation | Documentation

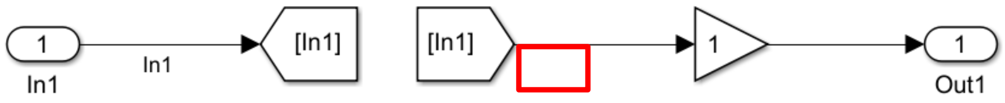
☐ Log signal data ☐ Test point

【Incorrect】

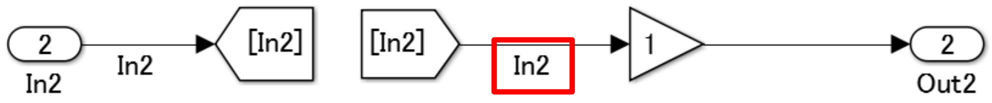
Signals that connect to [Inport] and [Goto] do not have names, but {Show propagated signals} is selected for signals that connect to [From] and [Gain].



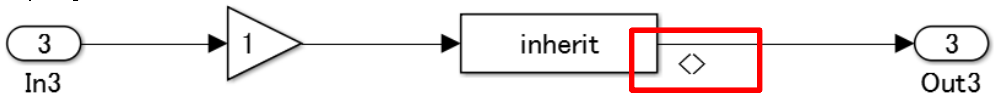
Regardless of whether signals are propagated, {Show propagated signals} is not selected.



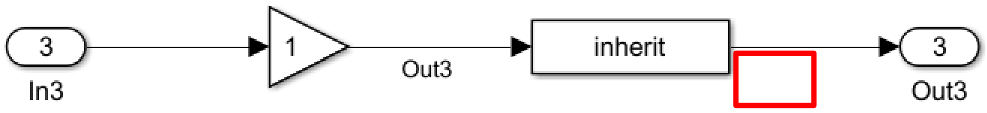
Signals that connect to [Inport] and [Goto] have names, but signals that connect to [From] and [Gain] are named.



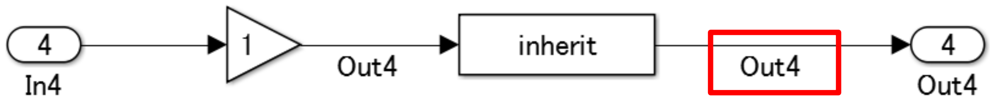
Signals that connect to [Gain] and [Signal Specification] do not have names, but {Show propagated signals} is selected for signals that connect to [Signal Specification] and [Outport].



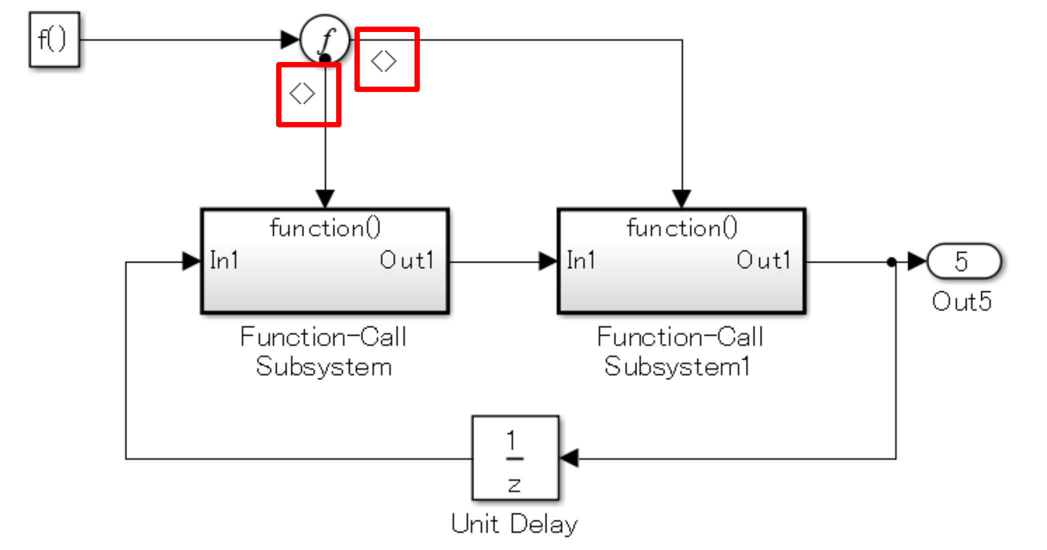
Regardless of whether signals are propagated, {Show propagated signals} is not selected.



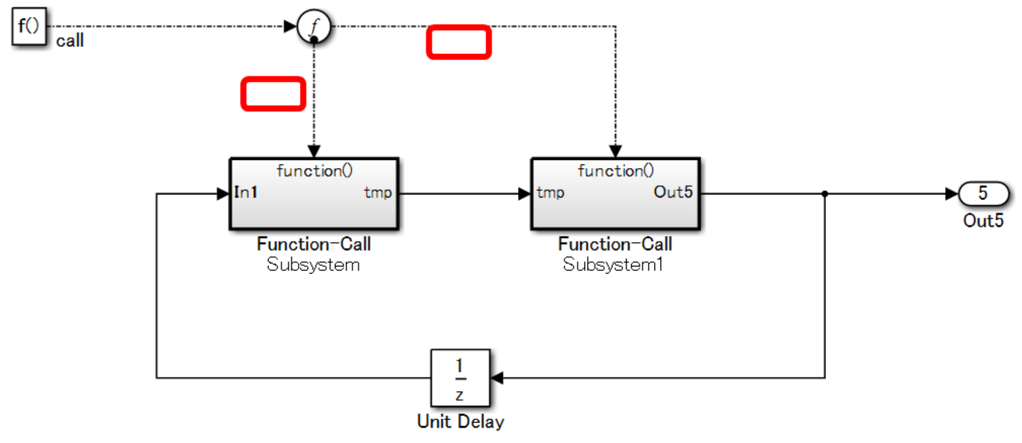
Signals that connect to [Gain] and [Signal Specification] have names, but signals that connect to [Signal Specification] and [Outport] have names.



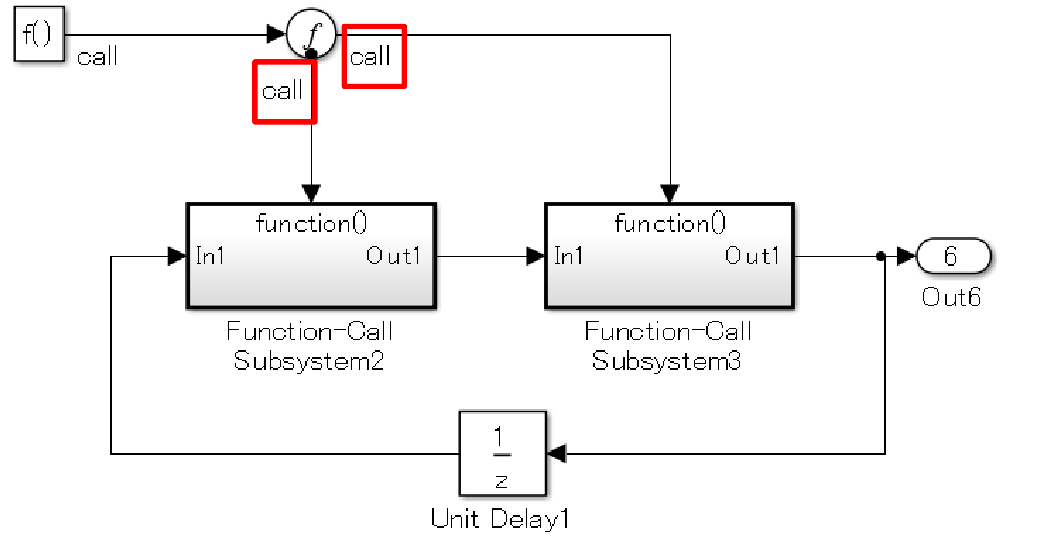
Signals that connect to [Function-Call Generator] and [Function-Call Split] do not have names, but {Show propagated signals} is selected for signals that connect to [Function-Call Split] and [Function-Call Subsystem].



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

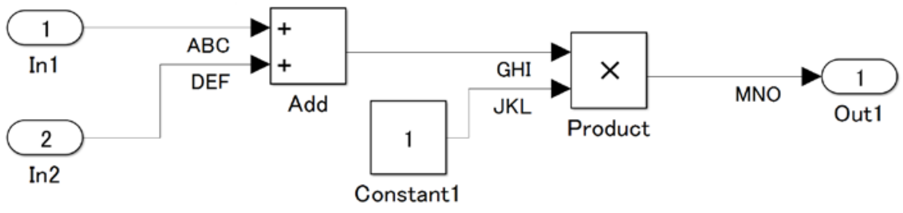
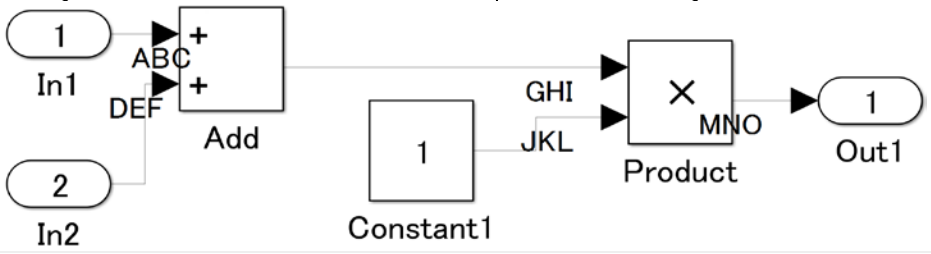
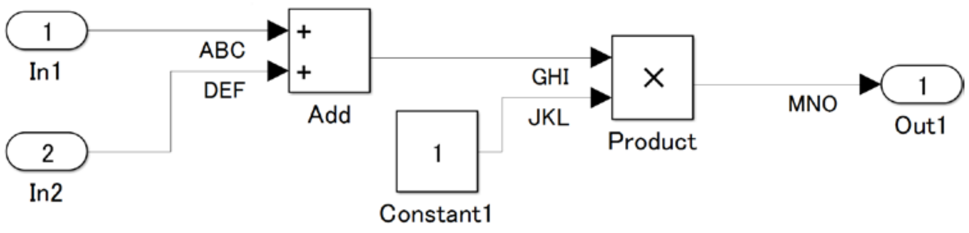


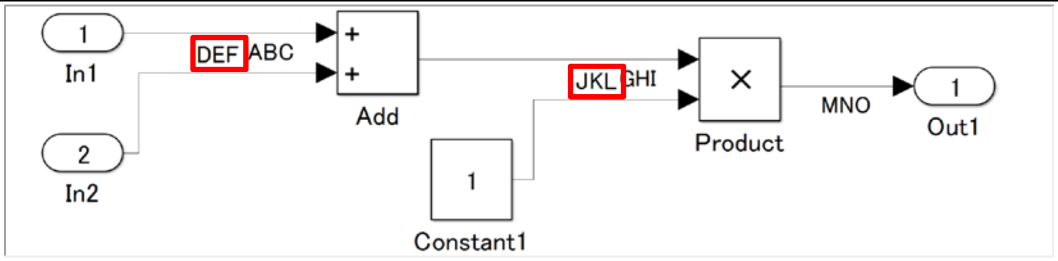
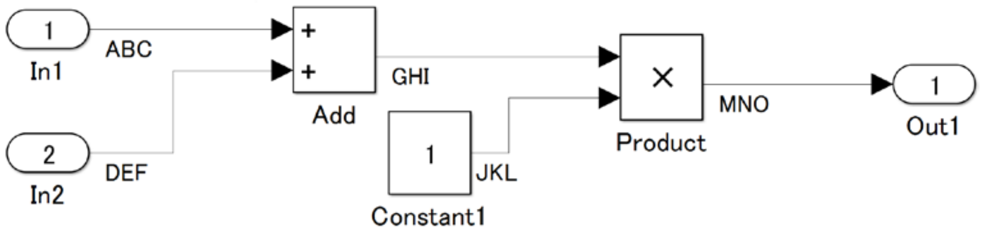
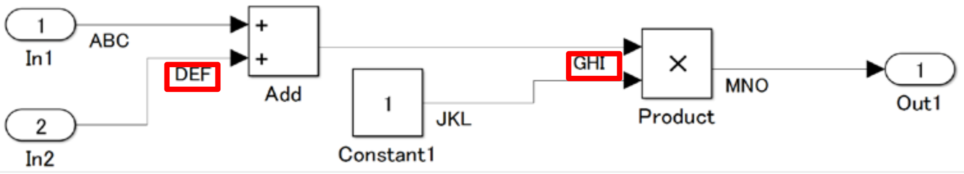
Signals that connect to [Function-Call Generator] and [Function-Call Split] have names and signals that connect to [Function-Call Split] and [Function-Call Subsystem] are also named.



Rationale	
Sub ID	**Description**
ab	･Prevents signal line connection mistakes. ･Prevents signal line name mistakes.

db_0097: Position of labels for signals and busses

Rule ID: Title	**db_0097: Position of labels for signals and busses**	
Sub ID Recommendations	NA-MAAB: a, b, c JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Signal line labels and bus labels shall not overlap other labels, signal lines, or blocks.	-
	【Correct】 The signal line labels and bus labels do not overlap other labels, signal lines, or blocks.  1 In1 2 In2 ABC DEF + + Add 1 Constant1 GHI JKL × Product MNO 1 Out1 【Incorrect】 The signal line labels and bus labels overlap other labels, signal lines, or blocks.  1 In1 2 In2 ABC DEF + + Add 1 Constant1 GHI JKL × MNO Product 1 Out1	
b	Signal line labels and bus labels shall be positioned below signal lines.	-
	【Correct】 Signal line labels and bus labels are below signal lines.  1 In1 2 In2 ABC DEF + + Add 1 Constant1 GHI JKL × Product MNO 1 Out1 【Incorrect】 Signal line labels and bus labels are above the signal line.	

	 1 In1 DEF ABC + + Add 2 In2 JKL GHI × Product MNO 1 Out1 1 Constant1	
c	Signal line labels and bus labels shall be positioned at the origin of the connection.	-
	【Correct】 Signal line labels and bus labels are positioned at the origin of the signal line connection.  1 In1 ABC + + Add GHI × Product MNO 1 Out1 2 In2 DEF 1 Constant1 JKL 【Incorrect】 Signal line labels and bus labels are positioned at the destination of the signal line connection.  1 In1 ABC DEF + + Add 2 In2 1 Constant1 JKL GHI × Product MNO 1 Out1	
Rationale		
Sub ID	**Description**	
a	· Adherence to this rule prevents confusion with corresponding names, signal lines, and busses, which improves readability of the model.	
bc	· Consistent label position prevents confusion with corresponding labels, signal lines, and busses, which improves the readability of the model.	

na_0008: Display of labels on signals

Rule ID: Title	**na_0008: Display of labels on signals**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	A label shall be displayed on the signal line originating from these blocks: • [Inport] • [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) • [Chart] Exception: When the signal label is visible in the originating block icon display, the signal does not need not to have the label displayed *unless* the signal label is needed elsewhere due to a destination-based rule.	
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] • [Chart]	-
Rationale		
Sub ID	**Description**	
a	· Improves readability, model simulation, and workflow. · Code generation may not be possible.	
b	· Improves readability, model simulation, and workflow.	

na_0009: Entry versus propagation of signal labels

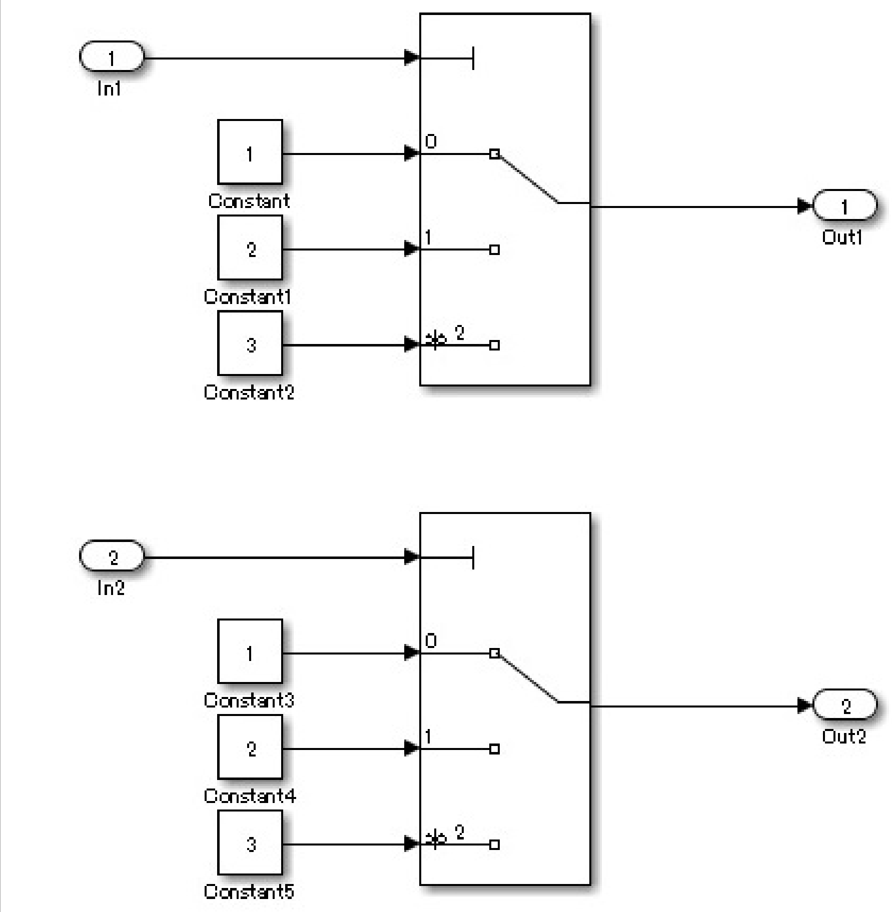
Rule ID: Title	**na_0009: Entry versus propagation of signal labels**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When a label is displayed for a signal, the following rules define whether that label is created there (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: • [Inport] 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] are included among the blocks that perform transformative operations.) Signal labels shall be *propagated* for signals that originate from: • [Inport] in a nested subsystem Exception**:** When the nested subsystem is a library subsystem, a label can be *entered* on the signal coming from [Inport] to accommodate reuse of the library block. • Basic blocks that perform a non-transformative operation · [Subsystem] or Stateflow [Chart] Exception**:** When the connection originates from the output of a library subsystem block, a new label can be *entered* on the signal to accommodate readability. · The result of executing a MATLAB command is reflected in the code, which makes consistency between the model and code difficult to maintain.	

db_0110: Block parameters

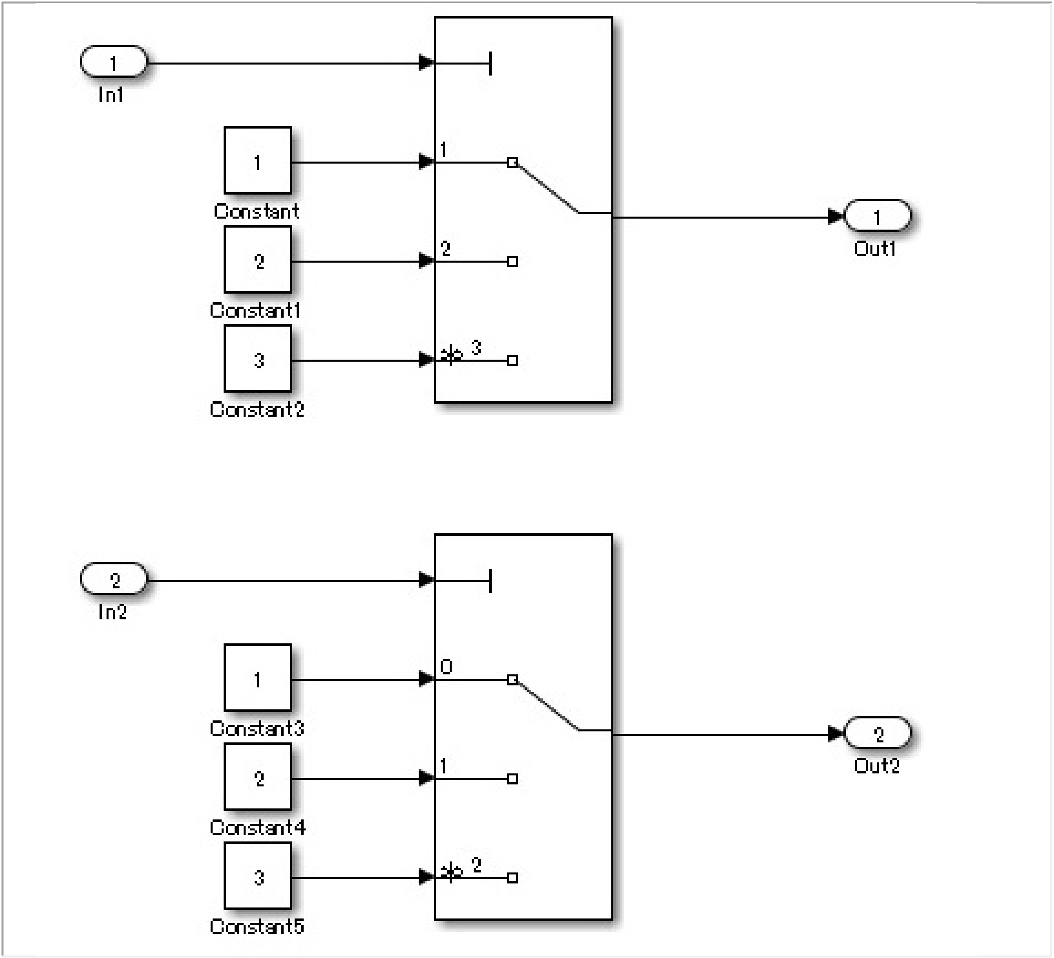
Rule ID: Title	**db_0110: Block parameters**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Block parameters shall not be used to describe: ・Operation expressions ・Data type conversion ・Selection of rows or columns ・MATLAB commands	-
Rationale		
Sub ID	**Description**	
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.	

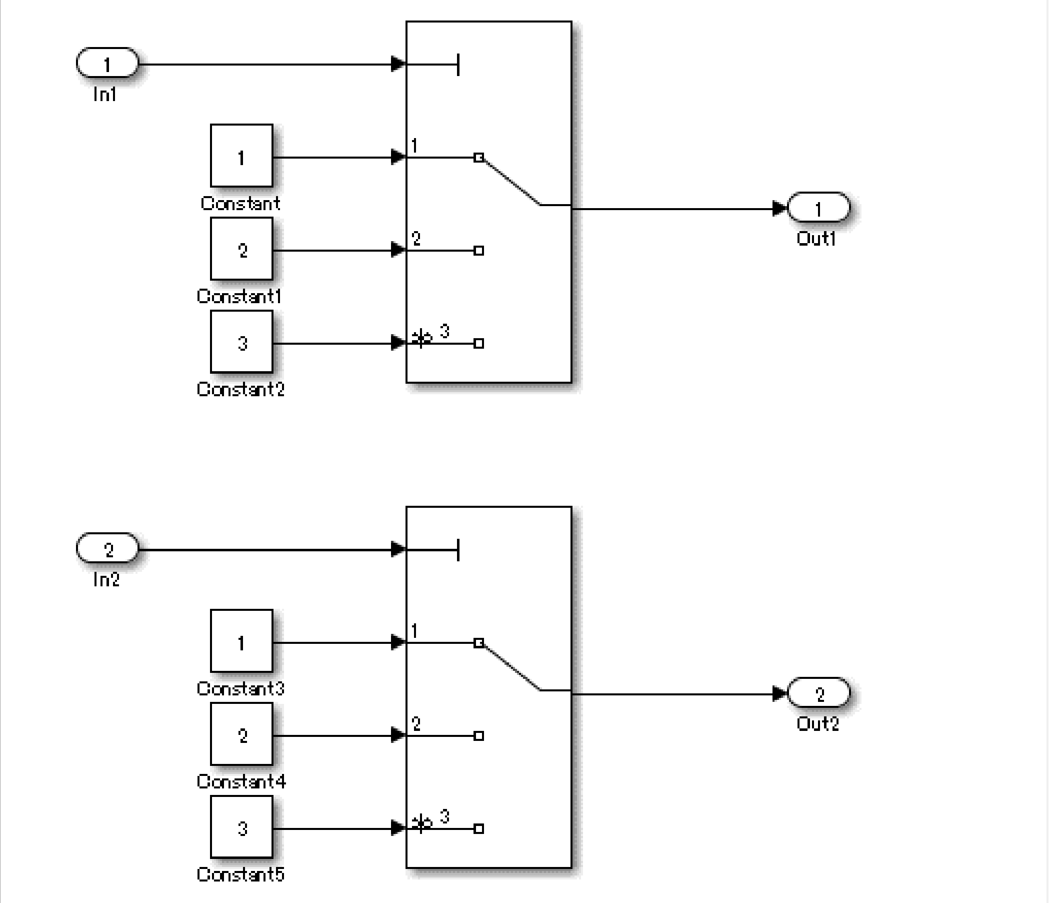
db_0112: Usage of index

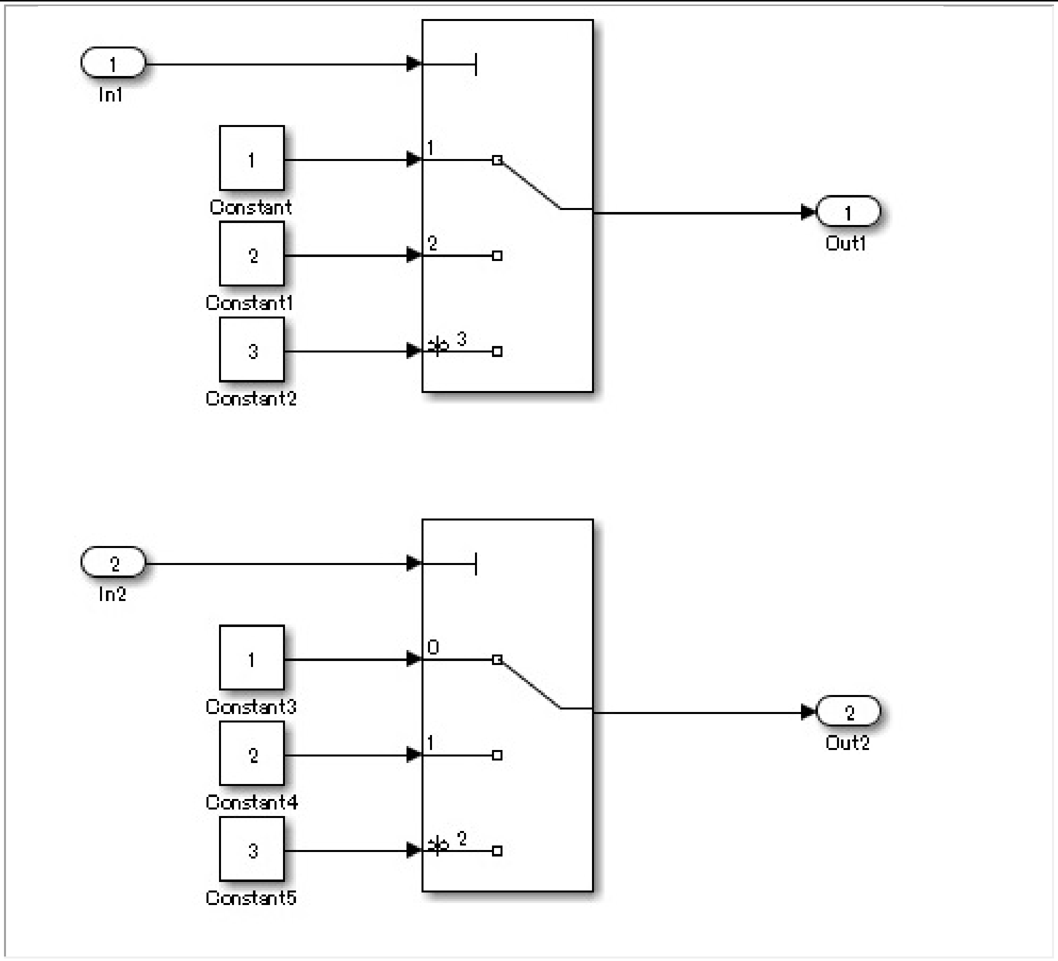
Rule ID: Title	**db_0112: Usage of index**	
Sub ID Recommendations	NA-MAAB: a1/a2 JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	A vector signal shall use a 1-based index mode.	-
	【Correct】 A uniform 0-based index mode is used.	



【Incorrect】
A uniform index mode is not used.

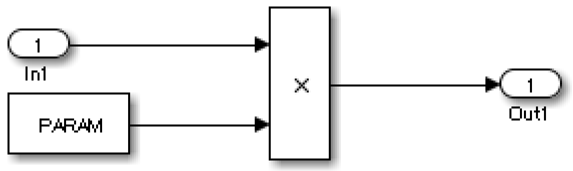
	 1 In1 1 Constant 2 Constant1 3 Constant2 1 2 *3 1 Out1 2 In2 1 Constant3 2 Constant4 3 Constant5 0 1 *2 2 Out2	
a2	A vector signal shall use a 1-based index mode.	-
	【Correct】 A uniform 1-based index mode is used.	

	 1 In1 1 Constant 2 Constant1 3 Constant2 1 2 3 1 Out1 2 In2 1 Constant3 2 Constant4 3 Constant5 1 2 3 2 Out2 【Incorrect】 A uniform index mode is not used. (Same as db_0112, Sub ID_a1).

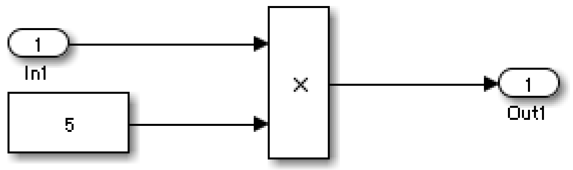
	 1 In1 1 Constant 2 Constant1 3 Constant2 1 2 3 1 Out1 2 In2 1 Constant3 2 Constant4 3 Constant5 0 1 2 2 Out2
Rationale	
Sub ID	**Description**
a1a2	Logic is easier to understand when using a uniform index mode.

jc_0645: Parameter definition for calibration

Rule ID: Title	**jc_0645: Parameter definition for calibration**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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	-
	【Correct】	



【Incorrect】



Rationale	
Sub ID	**Description**
a	A literal constant in the model will propagate as a literal constant in the generated code, making calibration impossible.

jc_0641: Sample time setting

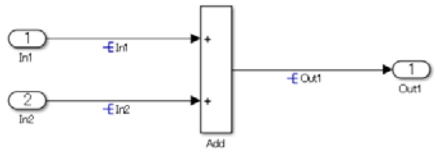
Rule ID: Title	**jc_0641: Sample time setting**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Block parameter {Sample time} shall be set to “-1” (inherited). Exceptions include: ・[Inport] ・[Outport] ・Atomic subsystem ・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] ・[Chart]	-
Rationale		
Sub ID	**Description**	
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.	

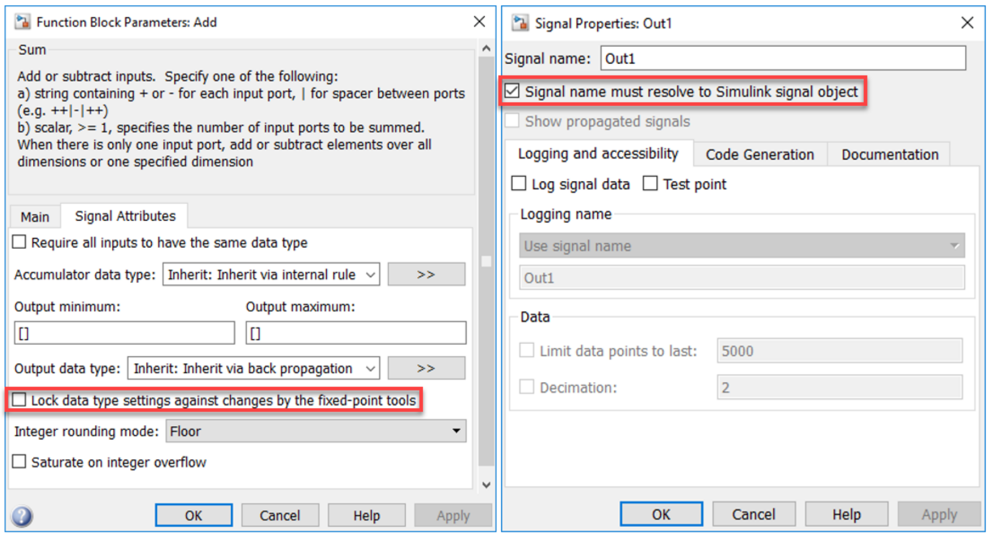
jc_0643: Fixed-point setting

Rule ID: Title	**jc_0643: Fixed-point setting**

Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When block parameters {Data type} is a fixed-point (fixdt) setting and {Scaling} is “Slope and bias”, parameter {Bias} shall be set to “0”.	-
Rationale		
Sub ID	**Description**	
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.	

jc_0644: Type setting

Rule ID: Title	**jc_0644: Type setting**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When the data type is set by a data object, a block or Stateflow data dictionary shall not be used to set the data type. Exceptions (see rationale for more information): ・Inside a reusable function ・[Data Type Conversion] ・Data types set by using “fixdt” ・Boolean type, double type	-
	【Correct】 0611 , not by the block. 	

Function Block Parameters: Add

Sum

Add or subtract inputs. Specify one of the following:
a) string containing + or - for each input port, | for spacer between ports (e.g. ++|-|++)
b) scalar, >= 1, specifies the number of input ports to be summed.
When there is only one input port, add or subtract elements over all dimensions or one specified dimension

Main | Signal Attributes

☐ Require all inputs to have the same data type

Accumulator data type: Inherit: Inherit via internal rule

Output minimum: []

Output maximum: []

Output data type: Inherit: Inherit via back propagation

☐ Lock data type settings against changes by the fixed-point tools

Integer rounding mode: Floor

☐ Saturate on integer overflow

Signal Properties: Out1

Signal name: Out1

☑ Signal name must resolve to Simulink signal object

☐ Show propagated signals

Logging and accessibility | Code Generation | Documentation

☐ Log signal data ☐ Test point

Logging name: Use signal name, Out1

Data

☐ Limit data points to last: 5000

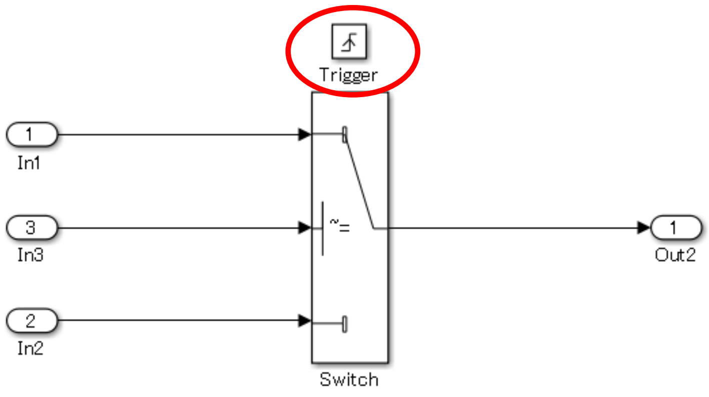
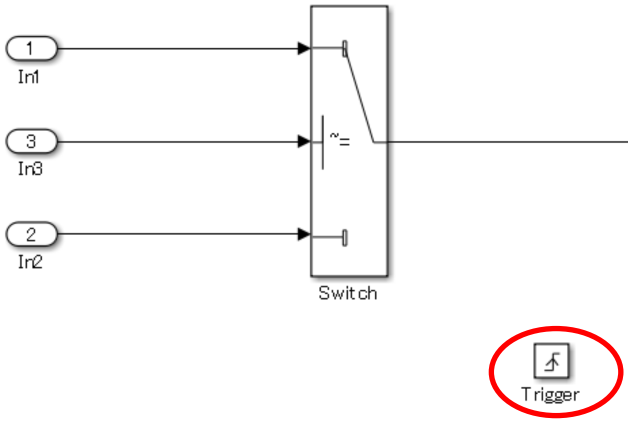
☐ Decimation: 2

Rationale	
Sub ID	**Description**
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: ・Inside a reusable function When all 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. ・[Data Type Conversion] This block is used to explicitly set the data type. ・Data types set by using “fixdt” When fixed-point is selected, 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. ・Boolean type, double type Some block types must be set to Boolean. Double type is generally used in plant models and for Rapid Control Prototyping (RCP), therefore it is not within scope of this rule. Embedded software uses double type in specific situations. Use caution when configuring the settings on these blocks to minimize the use of double type.

3.4. Conditional subsystem relations

db_0146: Block layout in conditional subsystems

Rule ID: Title	**db_0146: Block layout in conditional subsystems**
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b
MATLAB® Version	All

Rule		
Sub ID	**Description**	**Custom Parameter**
a	Conditional input blocks shall be positioned at the top of the subsystem.	-
	【Correct】 Trigger 1 In1 3 In3 2 In2 ~= 1 Out2 Switch 【Incorrect】 1 In1 3 In3 2 In2 ~= 1 Out2 Switch Trigger	
b	The position of these blocks shall be defined by the project: ・[For Each] ・[For Iterator] ・[While Iterator]	Location layout

Rationale	
Sub ID	**Description**
ab	・Unifying the internal and external layout of the conditional subsystem improves readability of the model.

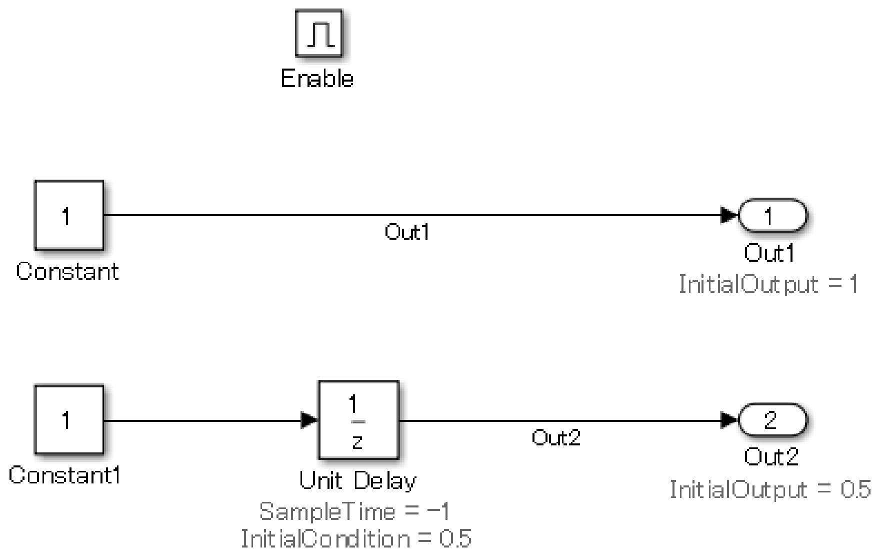
jc_0640: Initial value settings for Outport blocks in conditional subsystems

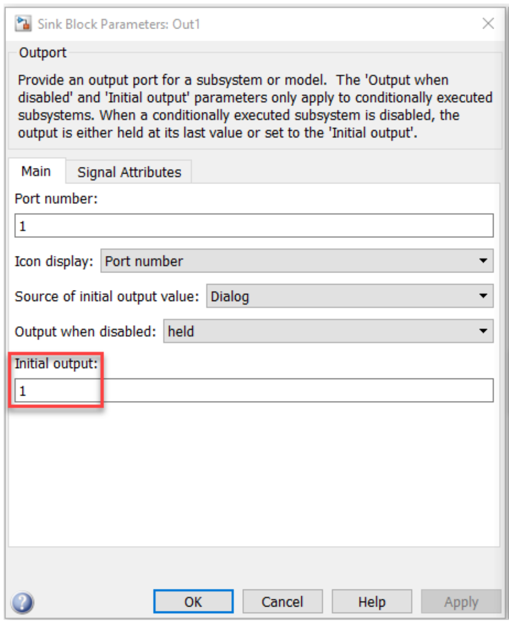
Rule ID: Title	**jc_0640: Initial value settings for Outport blocks in conditional subsystems**
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a
MATLAB® Version	All

Rule		
Sub ID	**Description**	**Custom Parameter**
a	When both conditions are met for a conditional subsystem: - Includes a block with initial conditions (i.e. [Constant] and [Delay]) - Connects to [Outport] The initial condition shall be defined on [Outport]. However, when the output signal from a conditional subsystem is connected to [Merge], the initial condition shall be defined on [Merge].	-

【Correct】

The initial condition is defined.




Sink Block Parameters: Out1

Outport

Provide an output port for a subsystem or model. The 'Output when disabled' and 'Initial output' parameters only apply to conditionally executed subsystems. When a conditionally executed subsystem is disabled, the output is either held at its last value or set to the 'Initial output'.

Main | Signal Attributes

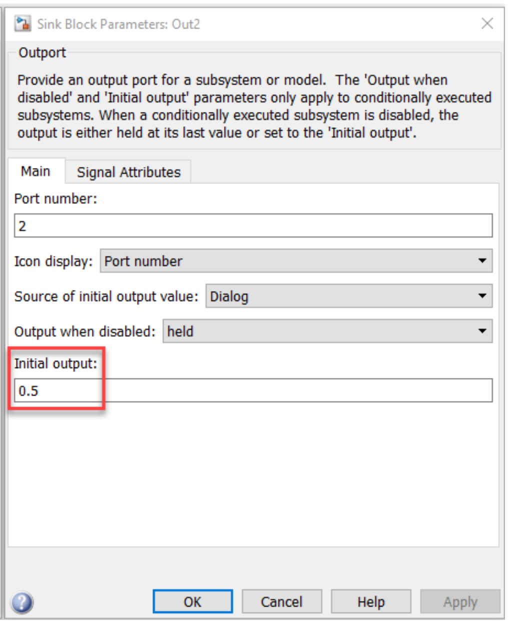
Port number: 1

Icon display: Port number

Source of initial output value: Dialog

Output when disabled: held

Initial output: 1


Sink Block Parameters: Out2

Outport

Provide an output port for a subsystem or model. The 'Output when disabled' and 'Initial output' parameters only apply to conditionally executed subsystems. When a conditionally executed subsystem is disabled, the output is either held at its last value or set to the 'Initial output'.

Main | Signal Attributes

Port number: 2

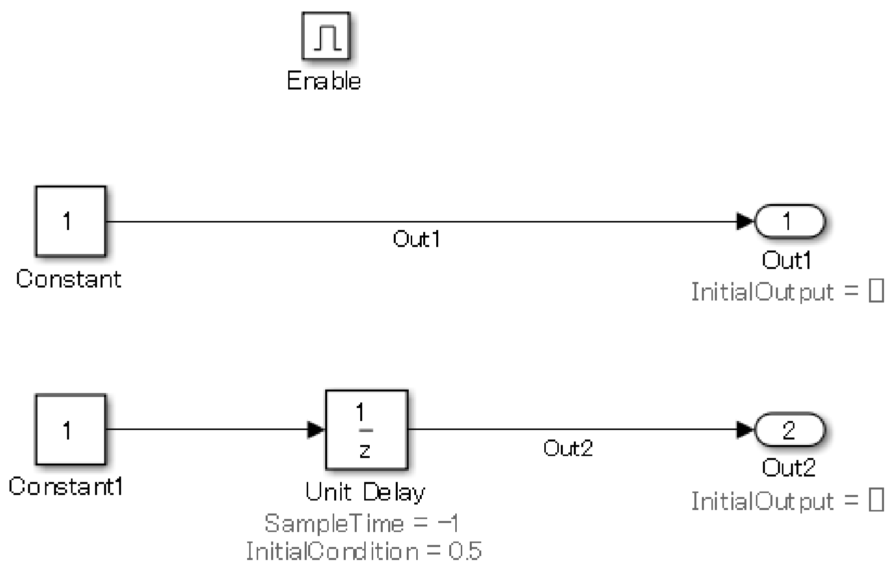
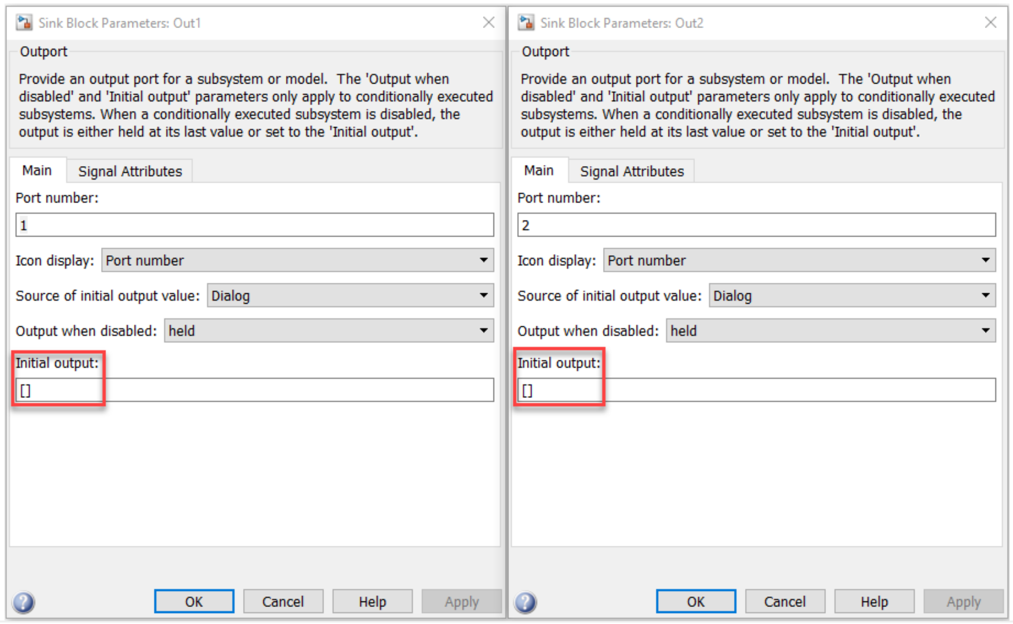
Icon display: Port number

Source of initial output value: Dialog

Output when disabled: held

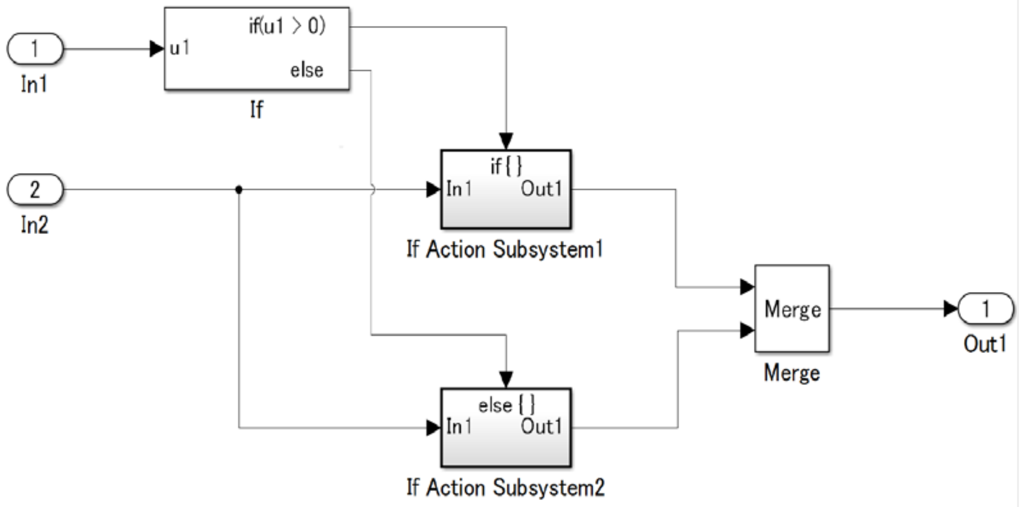
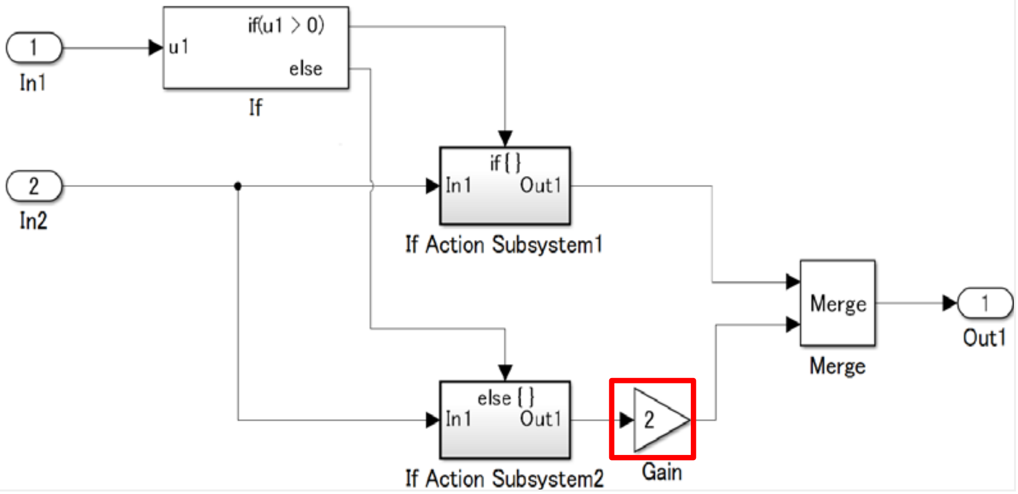
Initial output: 0.5

【Incorrect】

	The initial condition is undefined.  Enable 1 Constant Out1 1 Out1 InitialOutput = [] 1 Constant1 1/z Unit Delay SampleTime = -1 InitialCondition = 0.5 Out2 2 Out2 InitialOutput = []  Sink Block Parameters: Out1 Outport Provide an output port for a subsystem or model. The 'Output when disabled' and 'Initial output' parameters only apply to conditionally executed subsystems. When a conditionally executed subsystem is disabled, the output is either held at its last value or set to the 'Initial output'. Main / Signal Attributes Port number: 1 Icon display: Port number Source of initial output value: Dialog Output when disabled: held Initial output: [] OK Cancel Help Apply Sink Block Parameters: Out2 Outport Provide an output port for a subsystem or model. The 'Output when disabled' and 'Initial output' parameters only apply to conditionally executed subsystems. When a conditionally executed subsystem is disabled, the output is either held at its last value or set to the 'Initial output'. Main / Signal Attributes Port number: 2 Icon display: Port number Source of initial output value: Dialog Output when disabled: held Initial output: [] OK Cancel Help Apply
Rationale	
Sub ID	**Description**
a	· The model may not behave as intended when the initial condition is unclear.

jc_0659: Usage restrictions of signal lines input to Merge blocks

Rule ID: Title	**jc_0659: Usage restrictions of signal lines input to Merge blocks**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

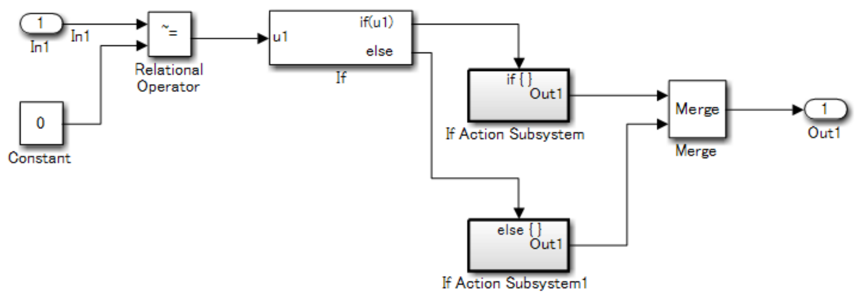
a	Only conditional subsystem output signals shall input to [Merge].	-
	【Correct】 Conditional subsystem output signal is input to [Merge].  1 In1 u1 if(u1 > 0) else If 2 In2 if { } In1 Out1 If Action Subsystem1 Merge Merge 1 Out1 else { } In1 Out1 If Action Subsystem2 【Incorrect】 0  1 In1 u1 if(u1 > 0) else If 2 In2 if { } In1 Out1 If Action Subsystem1 Merge Merge 1 Out1 else { } In1 Out1 If Action Subsystem2 2 Gain	

Rationale	
Sub ID	**Description**
a	・Prevents the simulation from proceeding as intended.

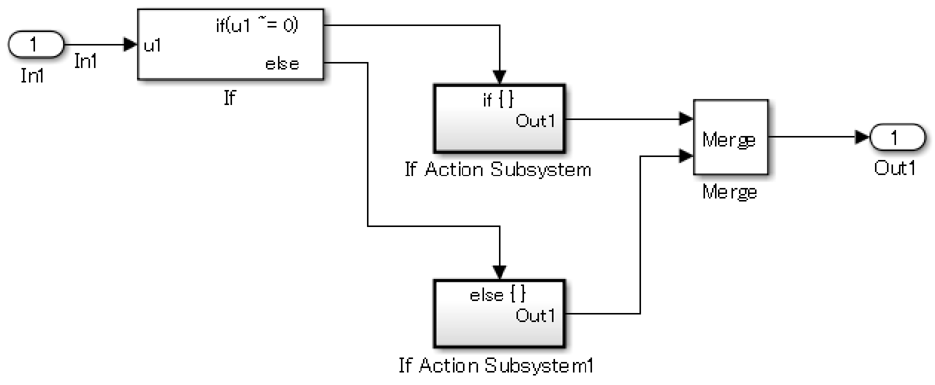
na_0003: Usage of If blocks

Rule ID: Title	**na_0003: Usage of If blocks**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	For [If], the {If expression} and {Elseif expression} shall be used only to define input signals.	-
	【Correct】	

The {If expression} only defines the input variables.



【Incorrect】
The {If expression} defines a comparison operation.



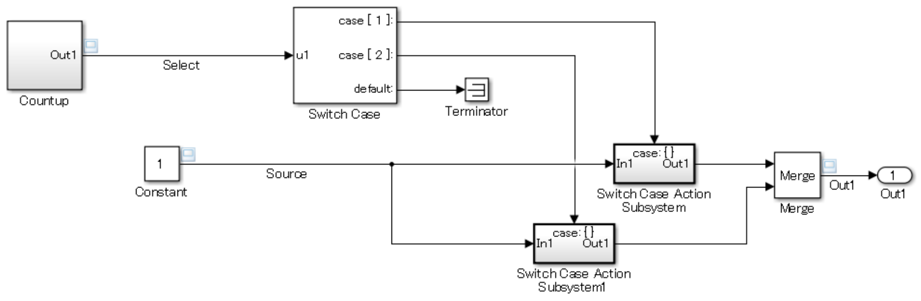
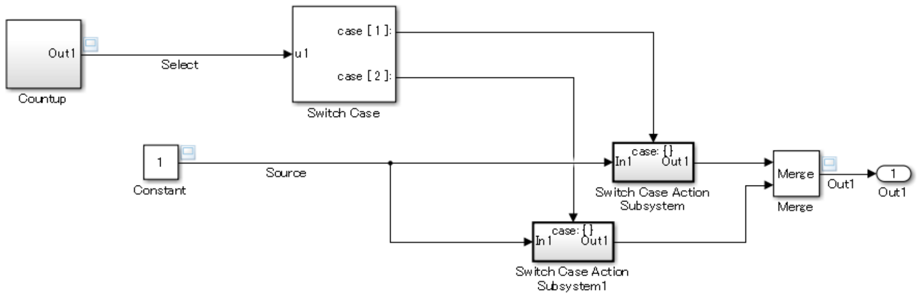
Rationale	
Sub ID	**Description**
a	・Visual comprehension of control conditions is easier when logical operations are described outside of [If]. ・Describing logical operations outside of [If] allows verification to focus on the logical operation.

jc_0656: Usage of Conditional Control blocks

Rule ID: Title	**jc_0656: Usage of Conditional Control blocks**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

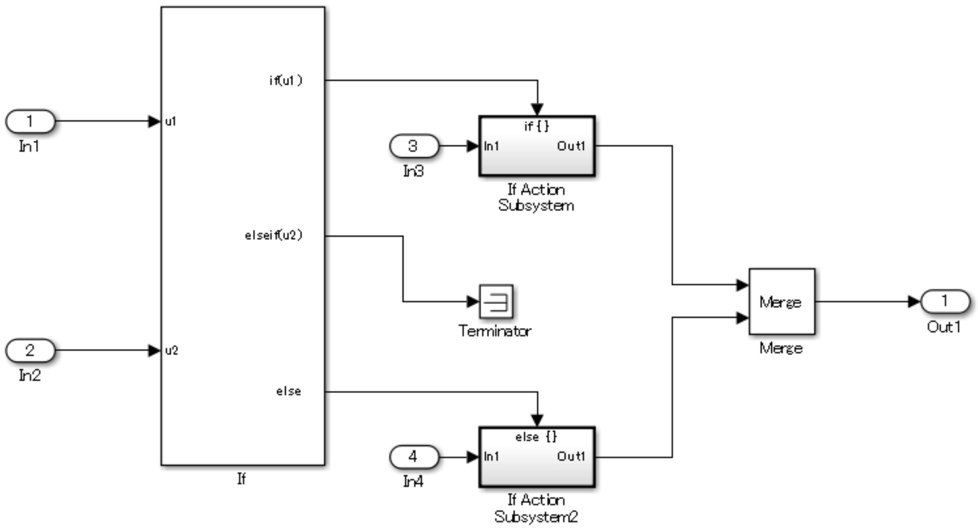
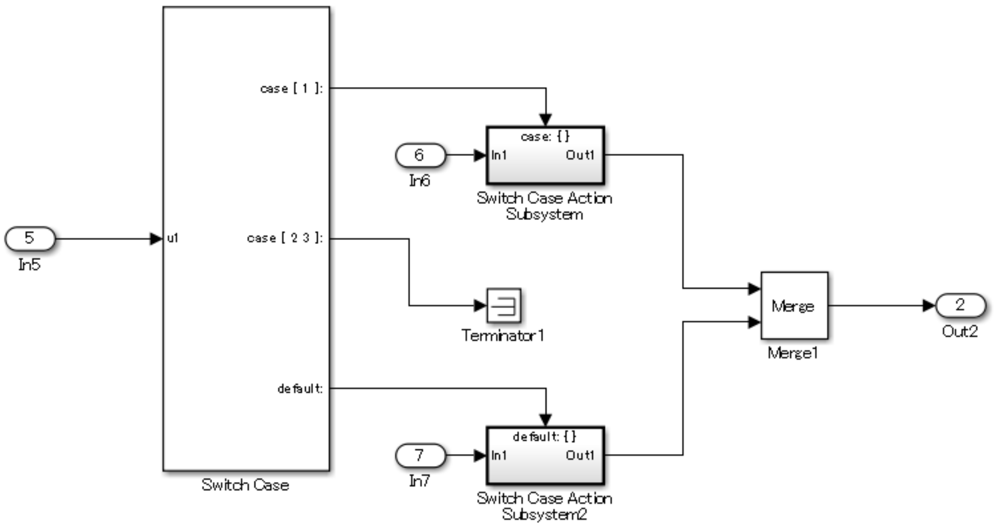
a	These block parameters shall be used to make all actions in the conditions explicit:
	・For [If], select {Show else condition}
	・For [Switch Case], select {Show default case}
		-

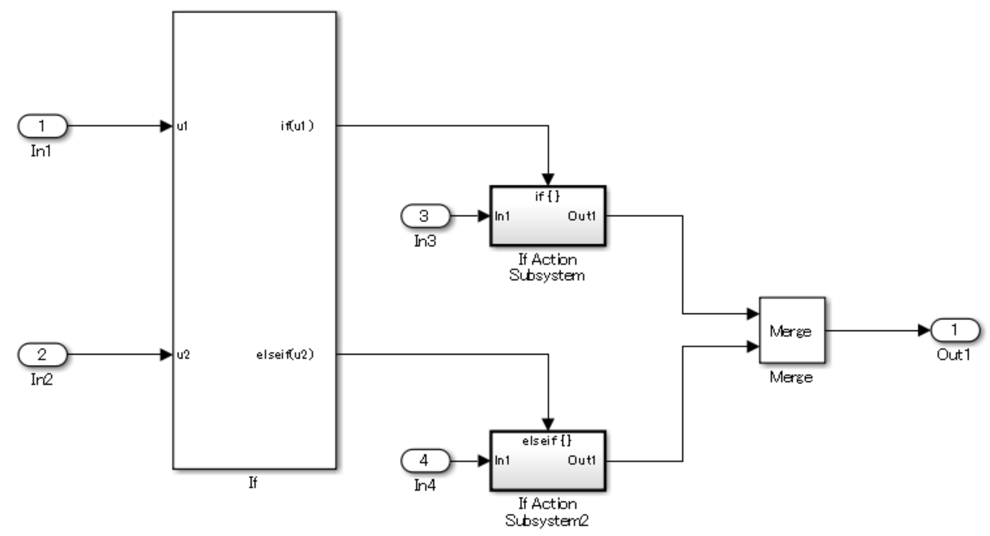
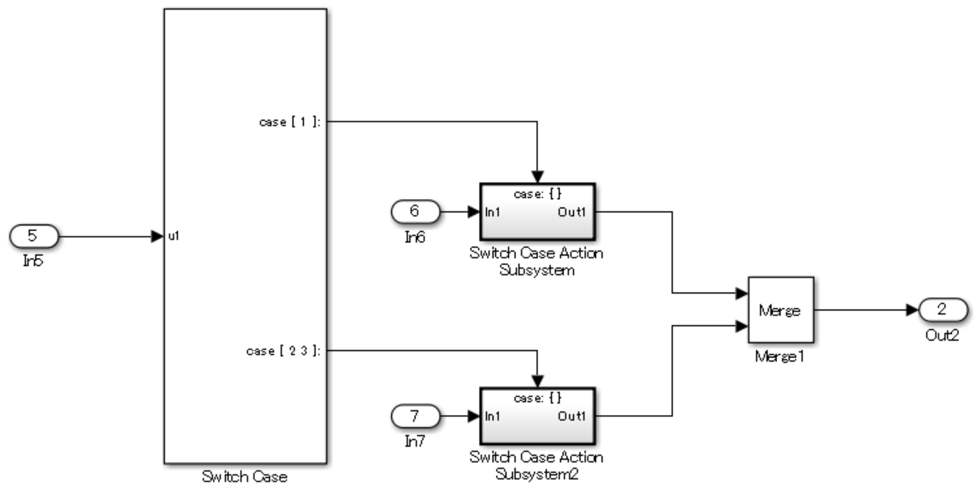
	【Correct】 Default behavior	

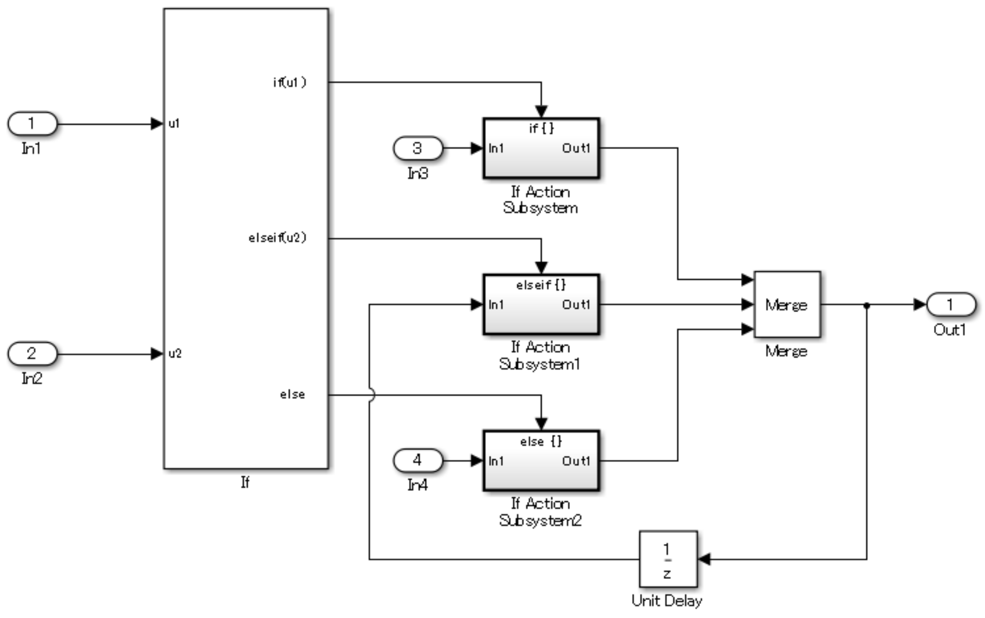
	 Out1 Countup Select u1 case [1]: case [2]: default: Switch Case Terminator 1 Constant Source case: { } In1 Out1 Switch Case Action Subsystem Switch Case Action Subsystem1 Merge Out1 1 Out1 【Incorrect】 No default behavior  Out1 Countup Select u1 case [1]: case [2]: Switch Case 1 Constant Source case: { } In1 Out1 Switch Case Action Subsystem Switch Case Action Subsystem1 Merge Out1 1 Out1
Rationale	
Sub ID	**Description**
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 .

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

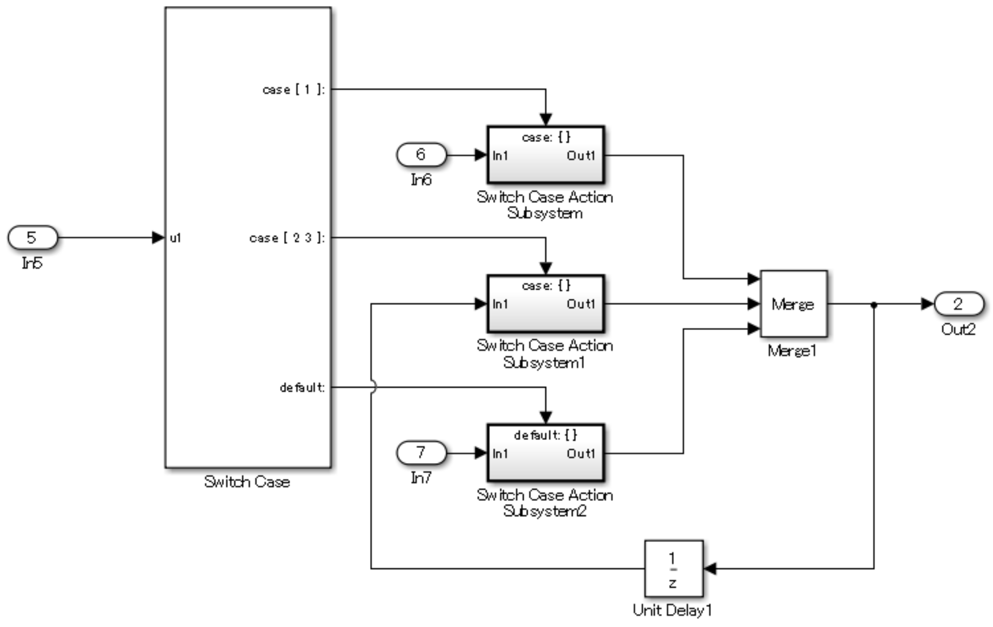
Rule ID: Title	**jc_0657: Retention of output value based on Conditional Control Flow blocks and Merge blocks**	
Sub ID Recommendations	NA-MAAB: a2 JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	Unused action ports shall connect to [Terminator] when these conditions are met: - Past value is retained - [Merge] and a conditional flow block, such as [If] or [Switch Case], are used to switch functions.	-

	【Correct】 [If] example  1 In1 u1 2 In2 u2 if(u1) elseif(u2) else If 3 In3 if { } In1 Out1 If Action Subsystem Terminator 4 In4 else { } In1 Out1 If Action Subsystem2 Merge Merge 1 Out1 [Switch Case] example  5 In5 u1 case [1]: case [2 3]: default: Switch Case 6 In6 case: { } In1 Out1 Switch Case Action Subsystem Terminator1 7 In7 default: { } In1 Out1 Switch Case Action Subsystem2 Merge Merge1 2 Out2 【Incorrect】 [If] example

	 1 In1 u1 if(u1) 2 In2 u2 elseif(u2) If 3 In3 if { } In1 Out1 If Action Subsystem 4 In4 elseif { } In1 Out1 If Action Subsystem2 Merge Merge 1 Out1 [Switch Case] example  5 In5 u1 case [1]: case [2 3]: Switch Case 6 In6 case: { } In1 Out1 Switch Case Action Subsystem 7 In7 case: { } In1 Out1 Switch Case Action Subsystem2 Merge Merge1 2 Out2	
a2	A feedback loop using [Delay] shall be implemented when these conditions are met: - Past value is retained - [Merge] and a conditional flow block, such as [If] or [Switch Case], are used to switch functions.	-
	【Correct】 [if] example	

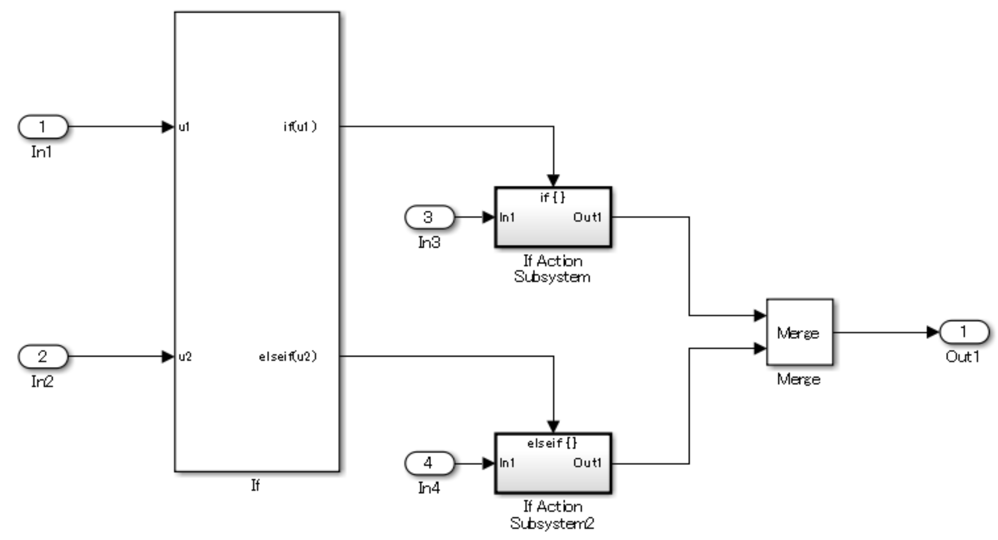


[Switch Case] example

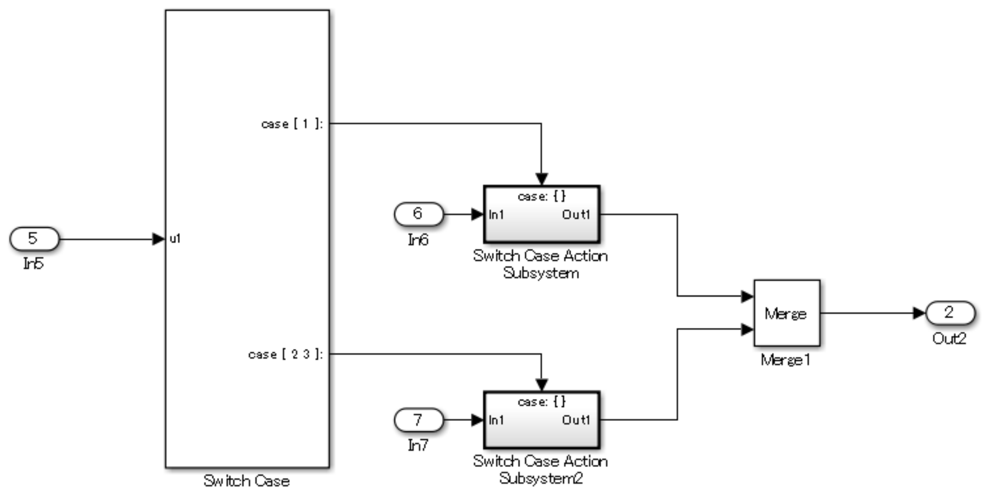


【Incorrect】

[If] example



[Switch Case] example

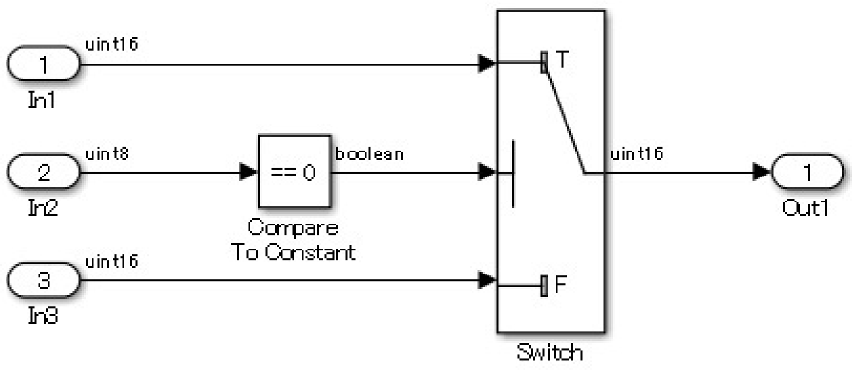
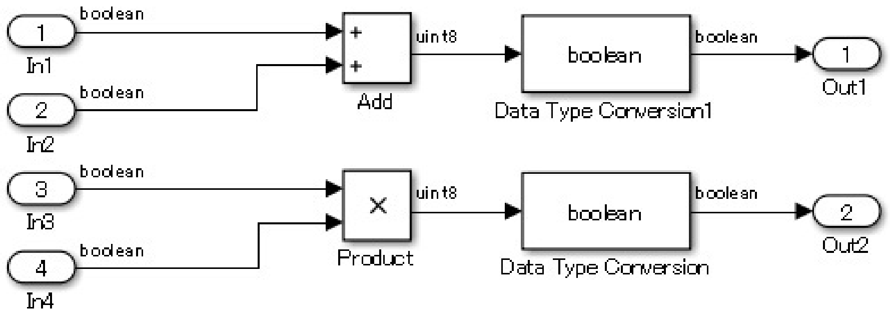
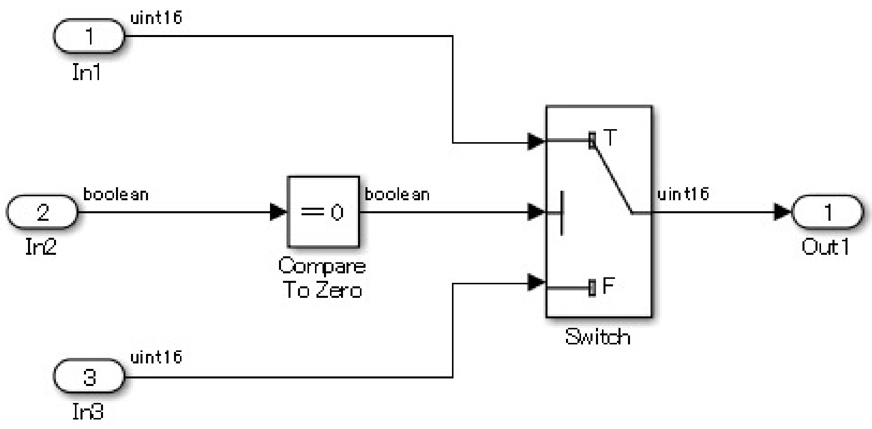


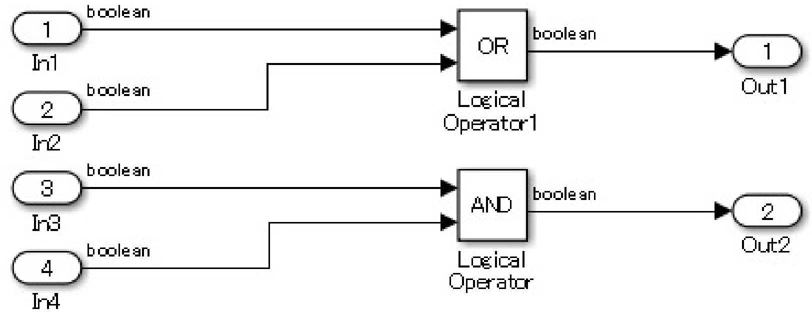
Rationale	
Sub ID	**Description**
a1	・Improves code efficiency. ・Connections to [Terminator] can be used when past values are held other than by the default (else).
a2	・Retaining past values is explicit.

3.5. Operation blocks

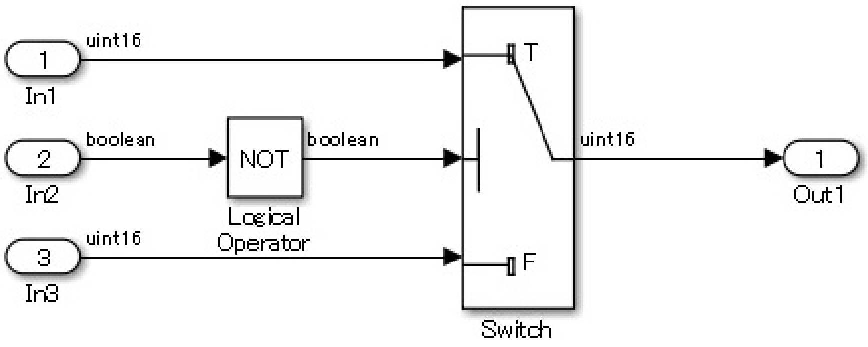
na_0002: Appropriate usage of basic logical and numerical operations

Rule ID: Title	**na_0002: Appropriate usage of basic logical and numerical operations**
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b
MATLAB® Version	All
Rule	

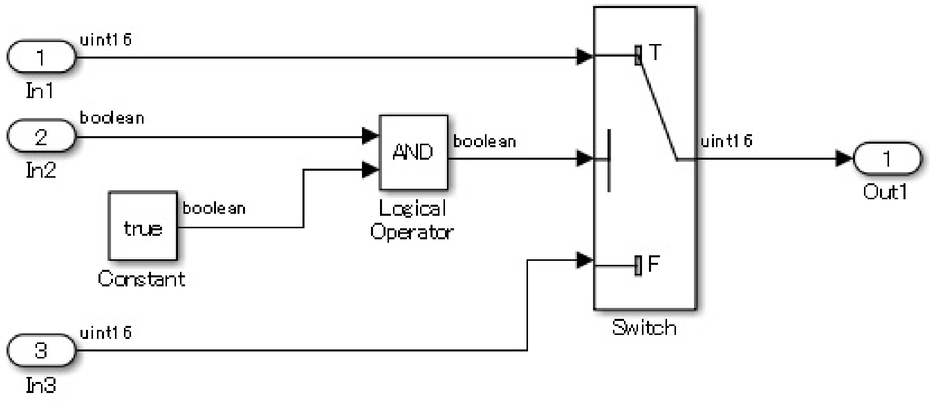
Sub ID	Description	Custom Parameter
a	Logical signals shall not connect to blocks that operate on numerical signals.	Blocks receiving numerical signals
	【Correct】 Numerical values are compared to determine if they are equal.  1 In1 uint16 2 In2 uint8 == 0 Compare To Constant boolean 3 In3 uint16 T F Switch uint16 1 Out1 【Incorrect】 A logical output is connected directly to the input of blocks that process numerical inputs.  1 In1 boolean 2 In2 boolean + + Add uint8 boolean Data Type Conversion1 boolean 1 Out1 3 In3 boolean 4 In4 boolean × Product uint8 boolean Data Type Conversion boolean 2 Out2 A logical signal is compared with a numerical value.  1 In1 uint16 2 In2 boolean = 0 Compare To Zero boolean 3 In3 uint16 T F Switch uint16 1 Out1	
b	Numerical signals shall not connect to blocks that operate on logical signals.	Blocks receiving logical signals
	【Correct】	



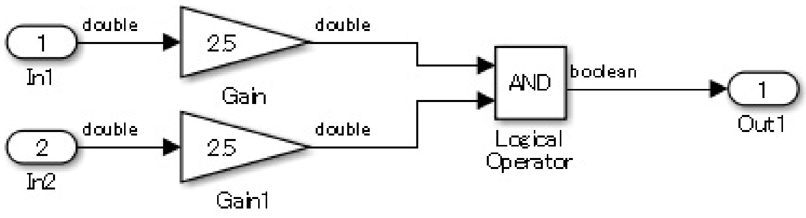
Logical signal is inverted by using a logical operation.



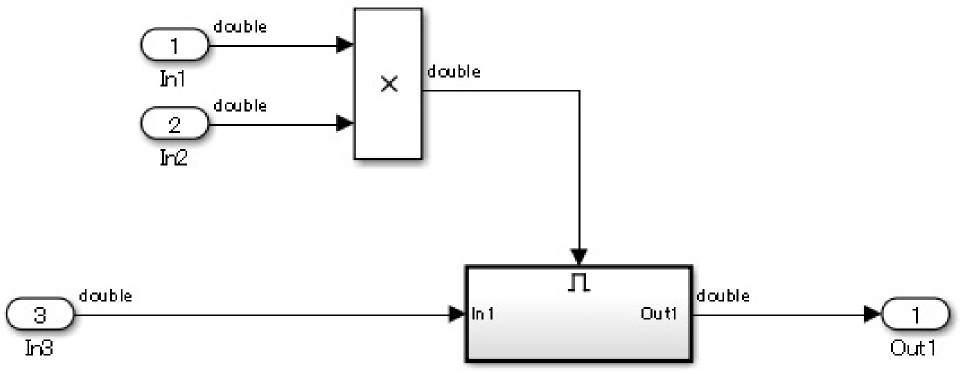
Logical signal is evaluated by using a logical operation.



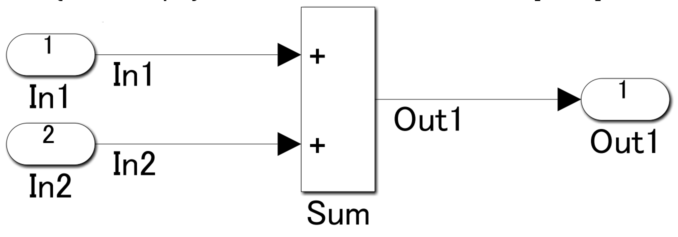
【Incorrect】
・A block that is used to perform logical operations is being used to perform numerical operations.
・A numerical output is connected to the input of blocks that process logical inputs.

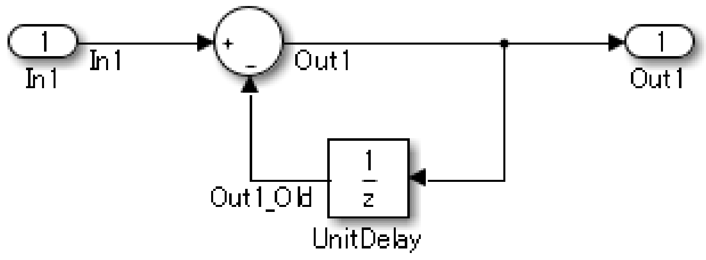
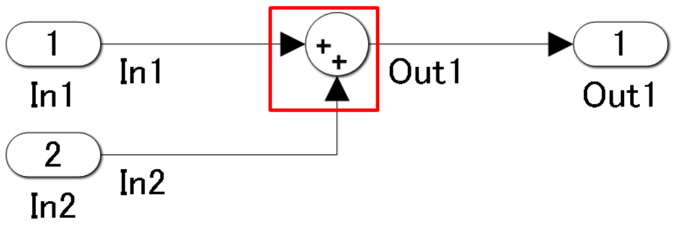
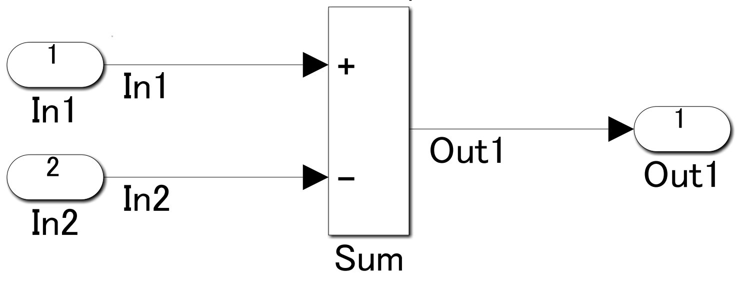
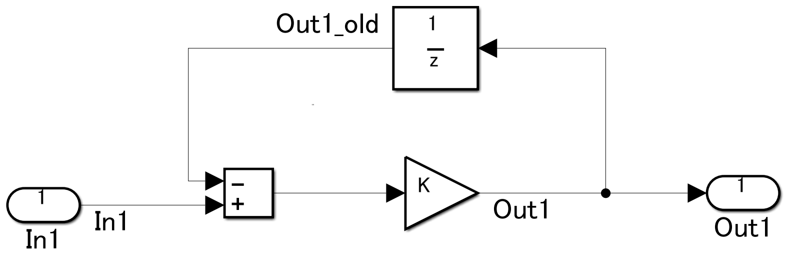


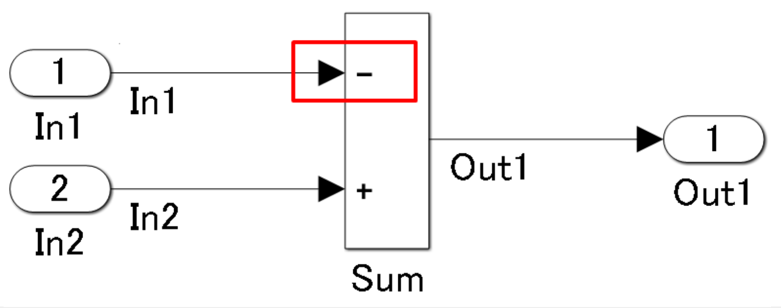
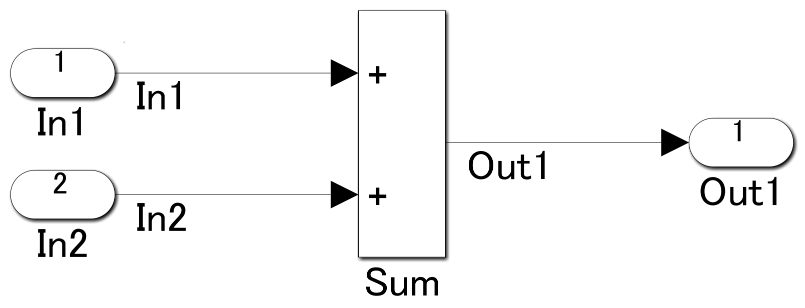
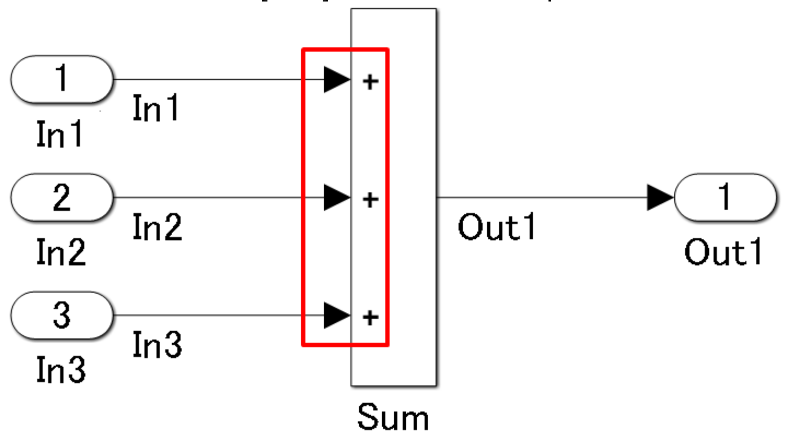
・A block that is used to perform numerical operations is being used to perform logical operations.
  Inputs other than logical values can be provided to the block. However, [Enable Port] can receive only logical signals that have On/Off.

	[Product] performs logical operations when it connects the numerical operations result to a block that receives the logical value [Enable Port].  double 1 In1 double 2 In2 × double 3 double In3 In1 Out1 double 1 Out1
Rationale	
Sub ID	**Description**
ab	・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.

jc_0121: Usage of add and subtraction blocks

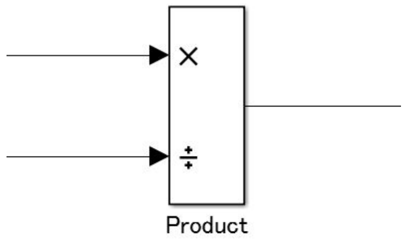
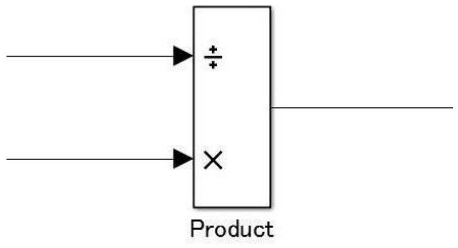
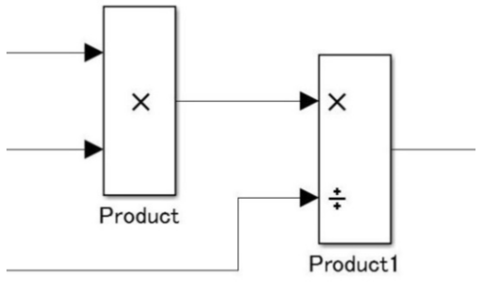
Rule ID: Title	**jc_0121: Usage of add and subtraction blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The {icon shape} of the add and subtraction [Sum] block shall be “rectangular”. When used in a feedback loop, the {icon shape} can be “round”.	-
	【Correct】 The {icon shape} for the add and subtraction [Sum] block is “rectangular”.  1 In1 In1 2 In2 In2 + + Sum Out1 1 Out1 The second input to the add and subtraction [Sum] block is a feedback loop, so the {icon shape} is “round”.	

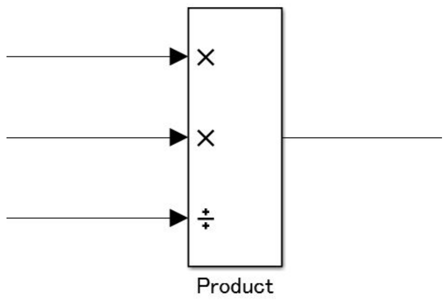
	 1 In1 In1 + - Out1 1 Out1 1 z Out1_Old UnitDelay 【Incorrect】 This is not a feedback loop, but the {icon shape} of the add and subtraction [Sum] block is “round”.  1 In1 In1 + + Out1 1 Out1 2 In2 In2	
b	The “+” mark shall be used for the first input to the add and subtraction [Sum] block. For a feedback loop, the first input can be set by using the “-” mark.	-
	【Correct】 The “+” mark is used for the first input to the add and subtraction [Sum] block.  1 In1 In1 2 In2 In2 + − Sum Out1 1 Out1 The second input to the add and subtraction [Sum] block is a feedback loop, so the “-” mark is used.  Out1_old 1 z − + K 1 In1 In1 Out1 1 Out1 【Incorrect】 The sign for the first input to the add and subtraction [Sum] block is the “-” mark.	

	 1 In1 In1 2 In2 In2 − + Sum Out1 1 Out1	
c	The add and subtraction [Sum] block shall not have more than two inputs.	-
	【Correct】 The add and subtraction [Sum] block has no more than two inputs.  1 In1 In1 2 In2 In2 + + Sum Out1 1 Out1 【Incorrect】 The add and subtraction [Sum] block has three inputs.  1 In1 In1 2 In2 In2 3 In3 In3 + + + Sum Out1 1 Out1	
Rationale		
Sub ID	**Description**	
a	Adherence to the guideline improves readability of the model.	
b	Readability of the control specification improves when the sign for the first input is consistent.	
c	The order of operations is clearly defined.	

jc_0610: Operator order for multiplication and division blocks

Rule ID: Title	**jc_0610: Operator order for multiplication and division blocks**
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b, c
MATLAB® Version	All

Rule		
Sub ID	Description	Custom Parameter
a	The “*” mark shall be used for the first input to a multiplication and division [Product] block.	-
	【Correct】 The “*” mark is used for the first input to the multiplication and division [Product] block.  【Incorrect】 The “/” mark is used for the first input to the multiplication and division [Product] block.  	
b	The multiplication and division [Product] block shall have no more than two inputs.	-
	【Correct】 The block has two inputs.  【Incorrect】 The block has three inputs.	

	
Rationale	
Sub ID	**Description**
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÷1st input)) × 2nd input). However, if division is performed later, the number of operations can be reduced.
b	・The order of operations is clearly defined.

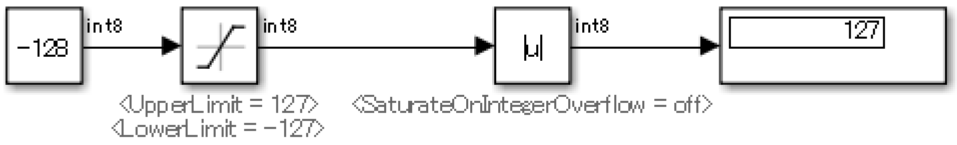
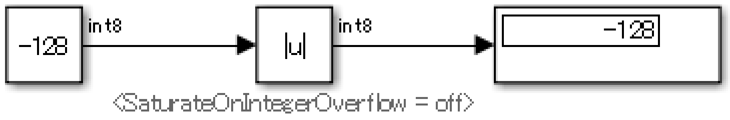
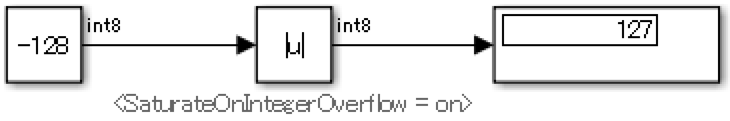
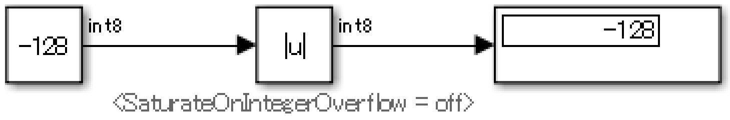
jc_0611: Input sign for multiplication and division blocks

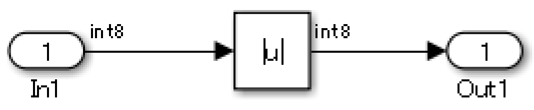
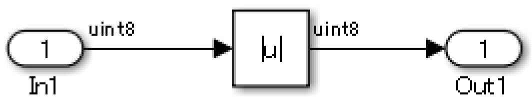
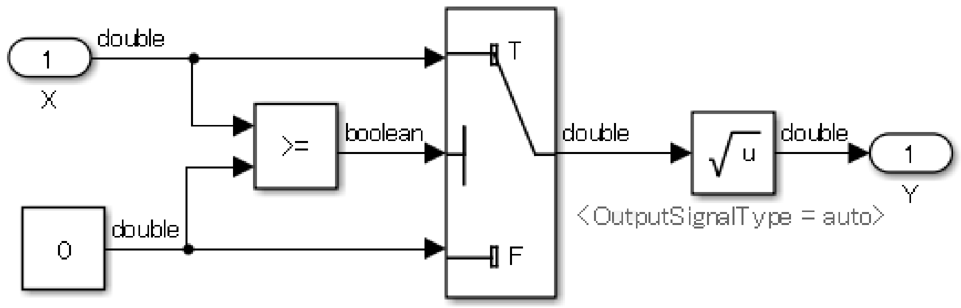
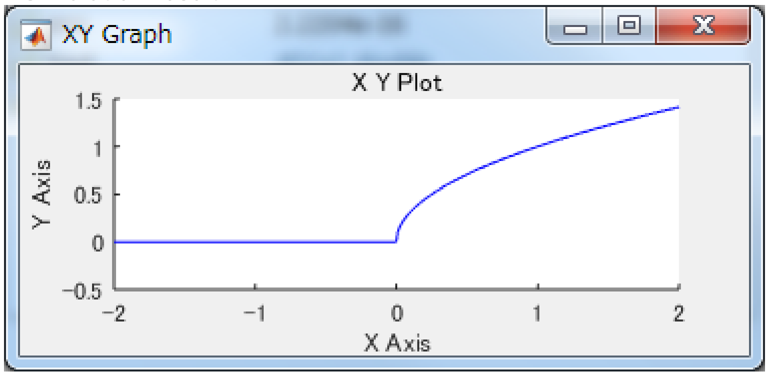
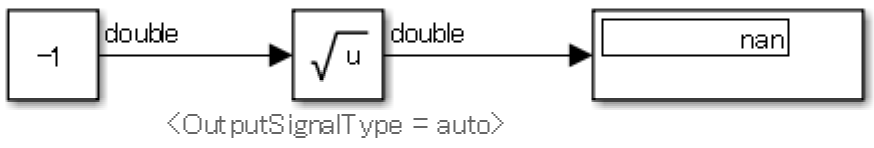
Rule ID: Title	**jc_0611: Input sign for multiplication and division blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When using fixed-point values as the input to the multiplication and division [Product] block, the sign of the data type shall be the same for all input signals.	-
Rationale		
Sub ID	**Description**	
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.	

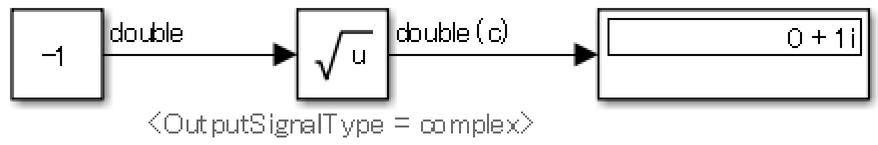
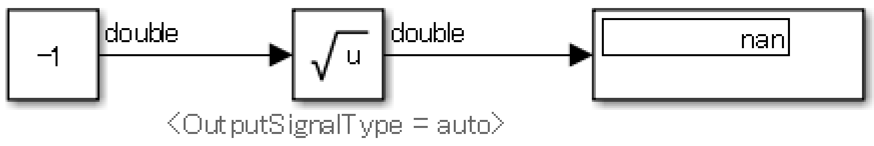
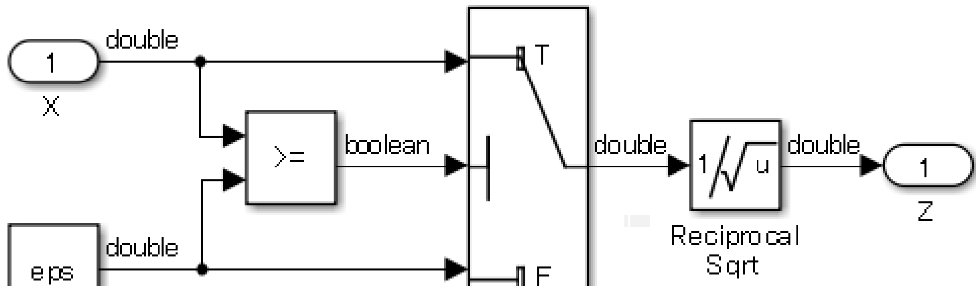
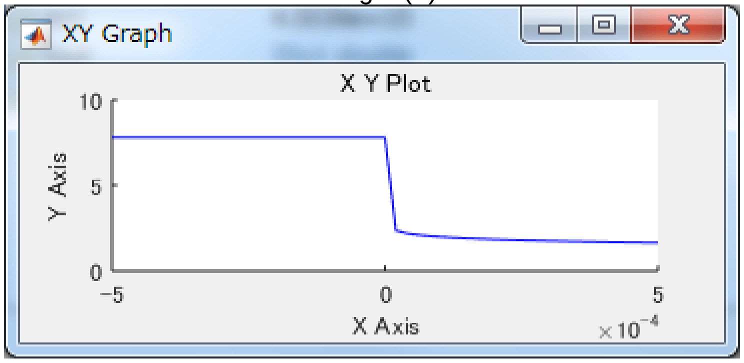
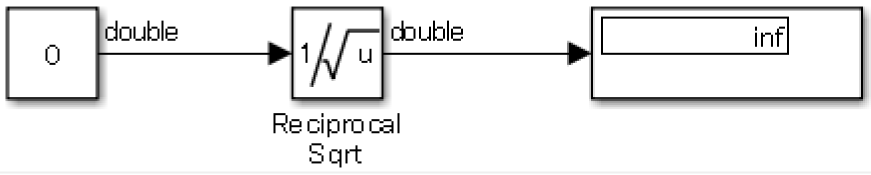
jc_0794: Division in Simulink

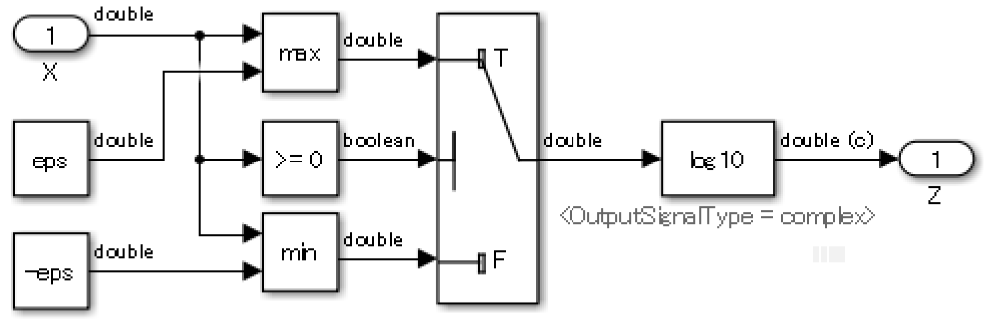
Rule ID: Title	**jc_0794: Division in Simulink**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When using division, implementation of the algorithm shall avoid division by zero.	-
Rationale		
Sub ID	**Description**	
a	・Deviation from the rule can cause unintended operation and code generation results.	

jc_0805: Numerical operation block inputs

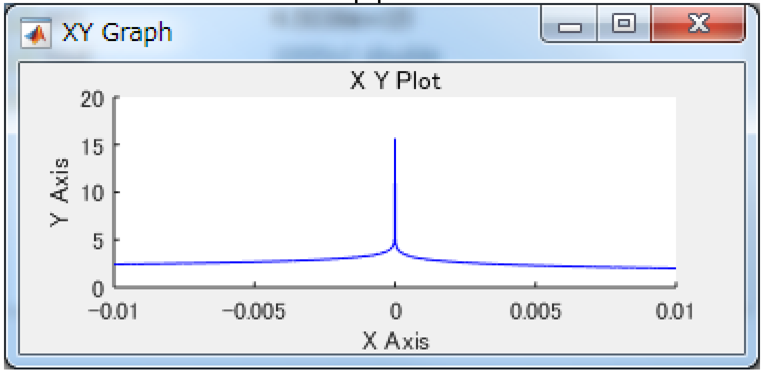
Rule ID: Title	**jc_0805: Numerical operation block inputs**	
Sub ID Recommendations	NA-MAAB: a1/a2, b, c1/c2, d, e, f1/f2, g, h, i, j JMAAB: a1/a2, b, c1/c2, d, e, f1/f2, g, h, i, j	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	When using [Abs] with signed integer types, the input shall not be the most negative value.	-
	【Correct】  -128 int8 int8 |u| int8 127 <UpperLimit = 127> <LowerLimit = -127> <SaturateOnIntegerOverflow = off> 【Incorrect】  -128 int8 |u| int8 -128 <SaturateOnIntegerOverflow = off>	
a2	[Abs] block parameter {Saturation on Integer Overflow} shall be selected.	-
	【Correct】  -128 int8 |u| int8 127 <SaturateOnIntegerOverflow = on> 【Incorrect】  -128 int8 |u| int8 -128 <SaturateOnIntegerOverflow = off>	
b	Input to [Abs] shall not be unsigned integer types or fixed-point types.	-
	【Correct】	

	 【Incorrect】 	
c1	Input to [Sqrt] shall not be a negative value.	-
	【Correct】 Negative number is saturated with 0.  1 X double >= boolean 0 double T F double √u double 1 Y \<OutputSignalType = auto\> Simulation result  【Incorrect】  −1 double √u double nan \<OutputSignalType = auto\>	
c2	[Sqrt] block parameter {Output Signal Type} shall be set to “complex”.	-
	【Correct】	

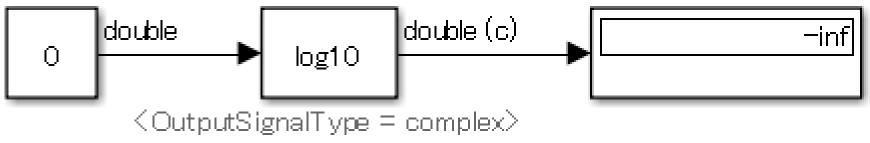
	 -1 double u double (c) 0 + 1i <OutputSignalType = complex> 【Incorrect】  -1 double u double nan <OutputSignalType = auto>	
d	Input to [Reciprocal Sqrt] shall not be less than zero.	-
	【Correct】 Less than eps saturated with eps  1 X double eps double >= boolean T F double 1/√u Reciprocal Sqrt double 1 Z Simulation result: Plot as Y=log10(Z)  【Incorrect】 	
e	When using [Math Function] and block parameter {Function} is set to “log” or “log10”, the input to the block shall not be zero.	-
	【Correct】 Replace within ±eps with ±eps	



Simulation result: Plot as Y = |Z|

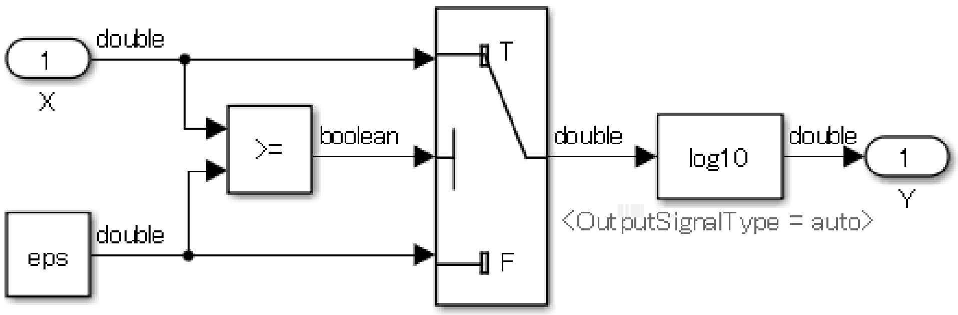


【Incorrect】

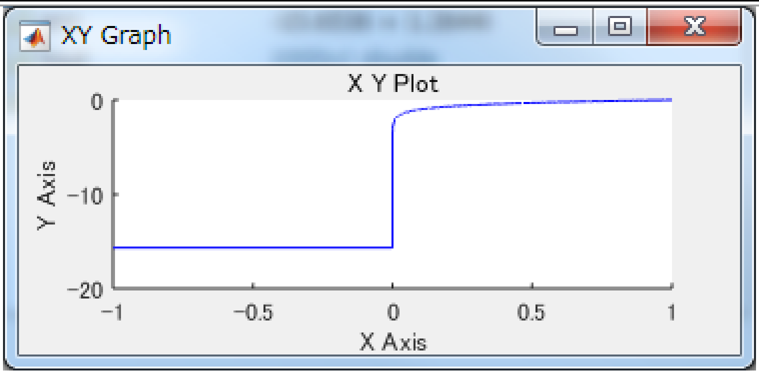
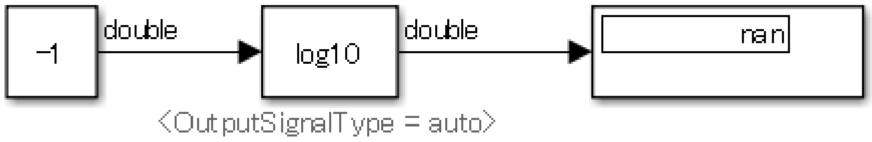
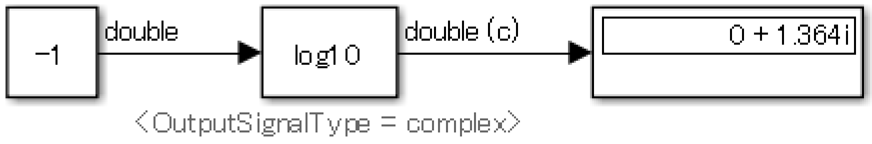
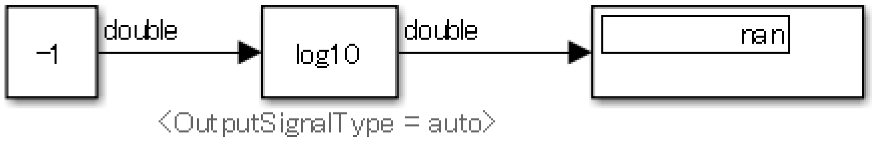
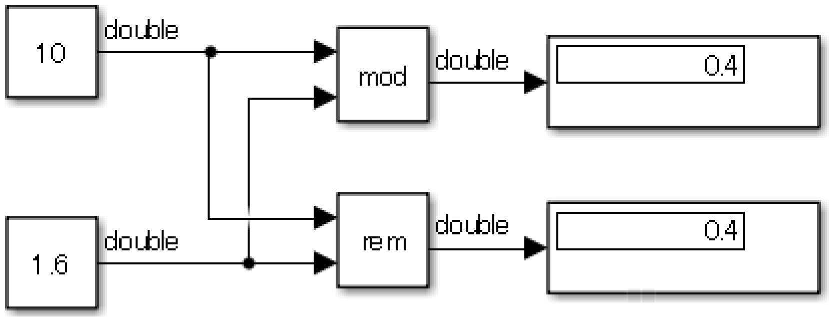


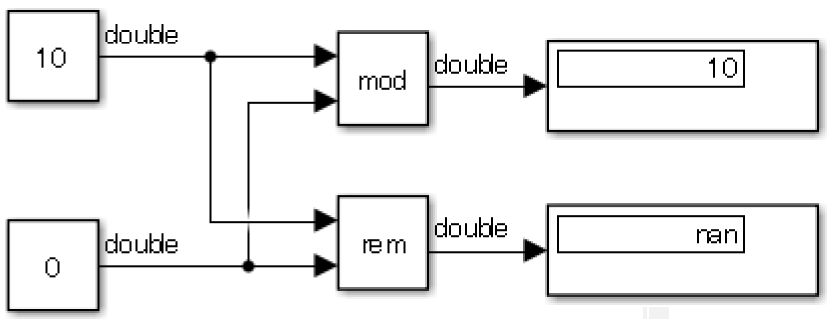
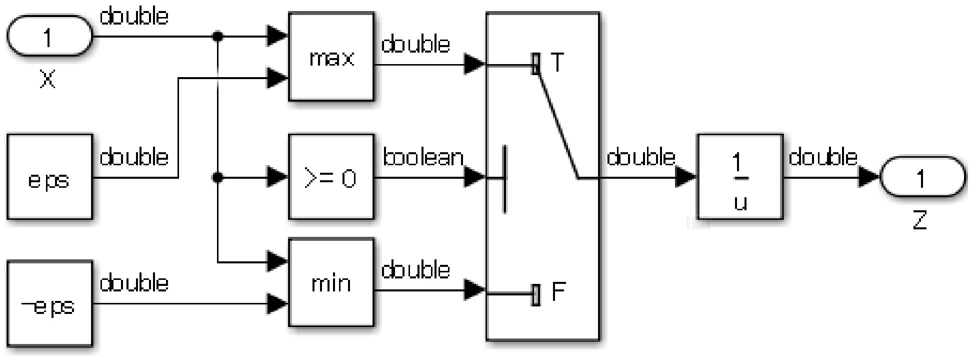
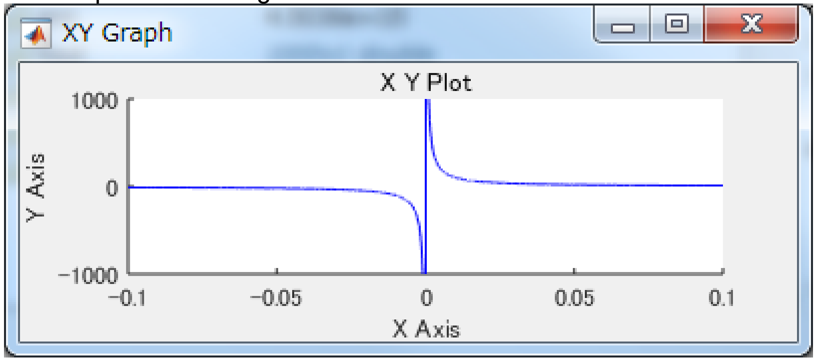
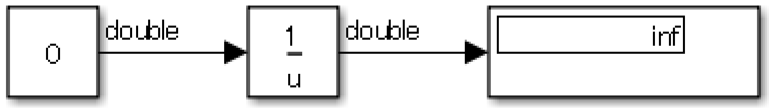
f1	When using [Math Function] and block parameter {Function} is set to “log” or “log10”, the input to the block shall not be a negative number.	-

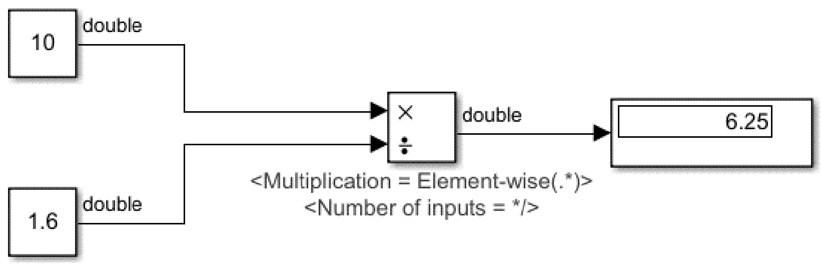
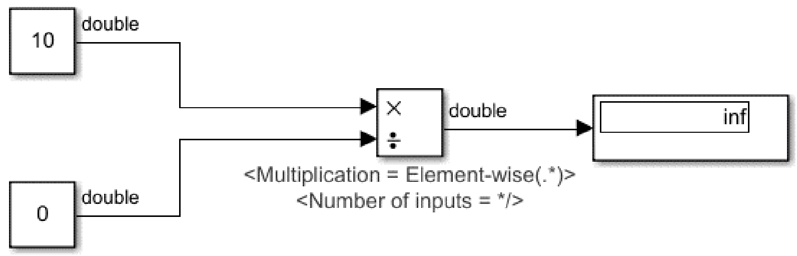
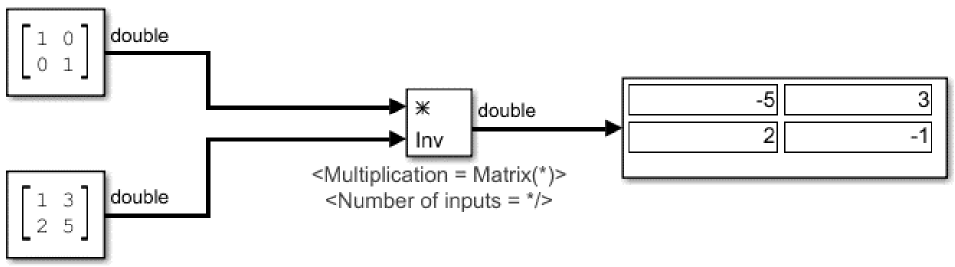
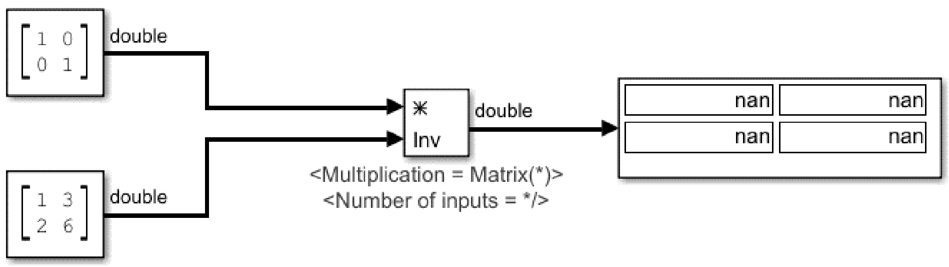
【Correct】
When the input is less than eps, the value is saturated to eps. Less than eps saturated with eps.



Simulation result

	 【Incorrect】  -1 double log10 double nan <OutputSignalType = auto>	
f2	When using [Math Function] and block parameter {Function} is set to “log” or “log10”, block parameter {Output Signal Type} shall be set to “complex”.	-
	【Correct】  -1 double log10 double (c) 0 + 1.364i <OutputSignalType = complex> 【Incorrect】  -1 double log10 double nan <OutputSignalType = auto>	
g	When using [Math Function] and block parameter {Function} is set to “mod” or “rem”, the second argument input shall not be zero.	-
	【Correct】  10 double mod double 0.4 1.6 double rem double 0.4 【Incorrect】	

	 10 double mod double 10 0 double rem double nan	
h	When using [Math Function] and block parameter {Function} is set to “reciprocal”, the input to the block shall not be zero.	-
	【Correct】 Replace within ±eps with ±eps  1 X double eps double −eps double max double >= 0 boolean min double T F double 1/u double 1 Z Simulation result: Simulation results is not inf, but since it is close to zero, the change in the output value is significant.  【Incorrect】 	
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 “*/”.)	-
	【Correct】	

	 10 double 1.6 double × ÷ double 6.25 <Multiplication = Element-wise(.*)> <Number of inputs = */> 【Incorrect】  10 double 0 double × ÷ double inf <Multiplication = Element-wise(.*)> <Number of inputs = */>	
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 “*/”.)	-
	【Correct】  [1 0; 0 1] double [1 3; 2 5] double * Inv double -5 3 2 -1 <Multiplication = Matrix(*)> <Number of inputs = */> 【Incorrect】  [1 0; 0 1] double [1 3; 2 6] double * Inv double nan nan nan nan <Multiplication = Matrix(*)> <Number of inputs = */>	

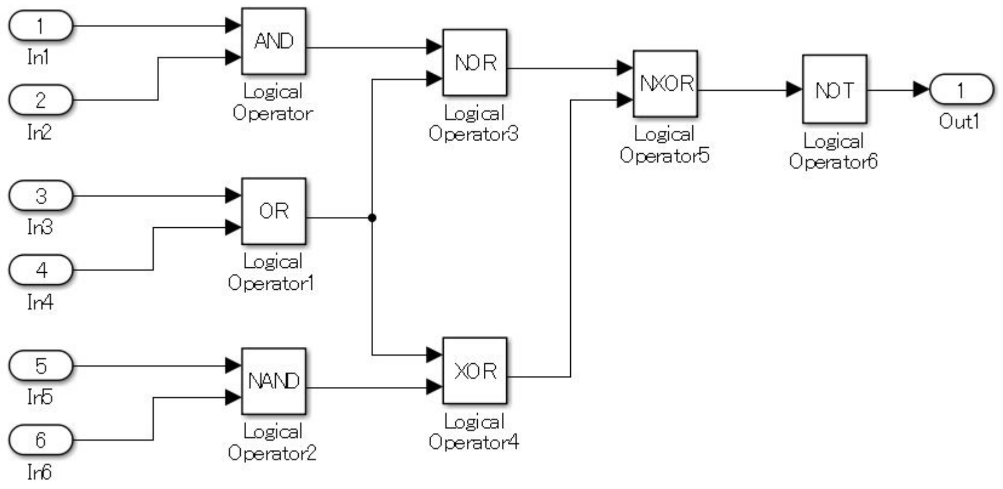
Rationale

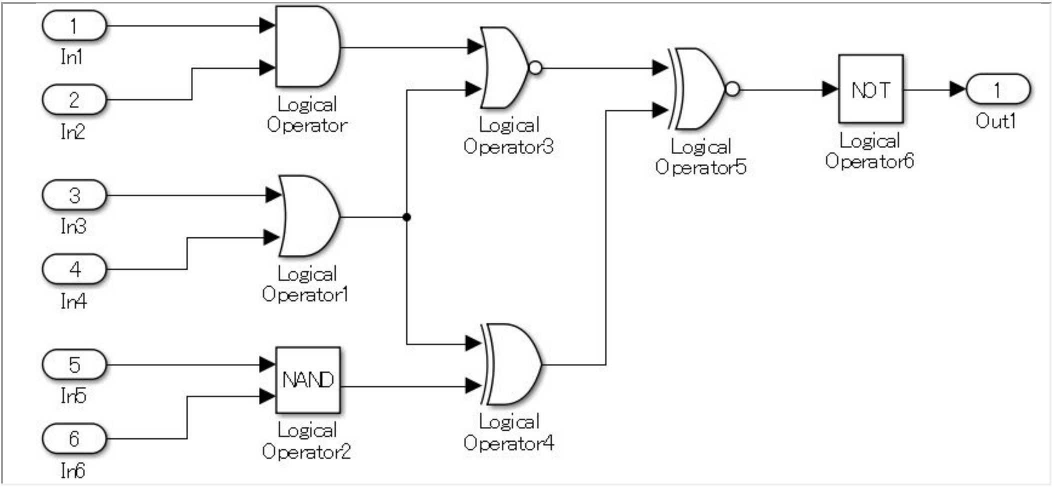
Sub ID	Description
a1c1de f1ghij	The result of entering an invalid value is implementation dependent. Deviation from the rules can result in unintended behavior.
a2	Correct settings prevent unintended behavior that can result from using invalid values.
b	The block can become optimized out of the generated code, resulting in a block that you cannot trace to the generated code.
c2f2	Correct settings prevent unintended behavior that can result from using negative values.

jc_0622: Usage of Fcn blocks

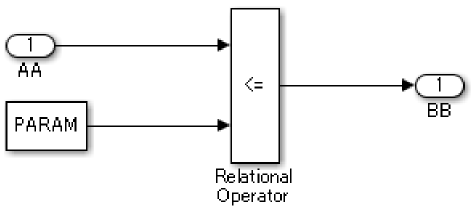
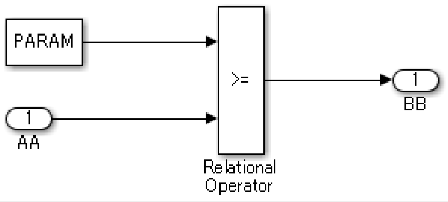
Rule ID: Title	**jc_0622: Usage of Fcn blocks**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When [Fcn] has operators with different priorities, parentheses shall be used to specify the priority order.	-
Rationale		
Sub ID	**Description**	
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.	

jc_0621: Usage of Logical Operator blocks

Rule ID: Title	**jc_0621: Usage of Logical Operator blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The {icon shape} for [Logical Operator] shall be set to “rectangular”.	-
	【Correct】 The icon shape for [Logical Operator] is set to “rectangular”.  1 In1 2 In2 3 In3 4 In4 5 In5 6 In6 AND Logical Operator OR Logical Operator1 NAND Logical Operator2 NOR Logical Operator3 XOR Logical Operator4 NXOR Logical Operator5 NOT Logical Operator6 1 Out1 【Incorrect】 Some of the icon shapes for [Logical Operator] are not “rectangular”.	

	 1 In1 2 In2 Logical Operator 3 In3 4 In4 Logical Operator1 5 In5 6 In6 NAND Logical Operator2 Logical Operator3 Logical Operator4 Logical Operator5 NOT Logical Operator6 1 Out1
Rationale	
Sub ID	**Description**
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.

jc_0131: Usage of Relational Operator blocks

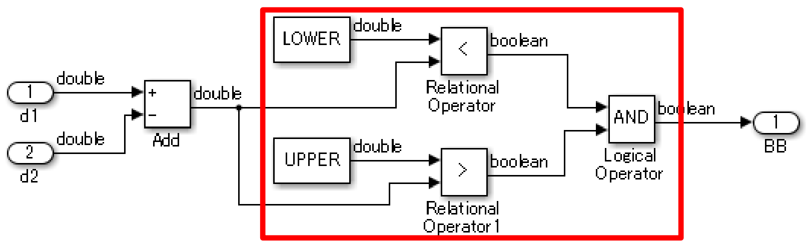
Rule ID: Title	**jc_0131: Usage of Relational Operator blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When using [Relational Operator] for comparison of signals and constants, the second (bottom) input shall be used as the constant input.	-
	【Correct】  【Incorrect】 	
Rationale		
Sub ID	**Description**	

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

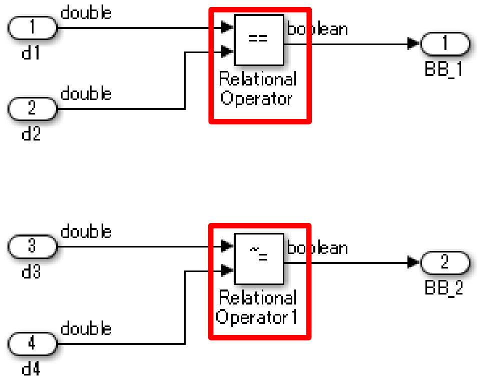
jc_0800: Comparing floating-point types in Simulink

Rule ID: Title	**jc_0800: Comparing floating-point types in Simulink**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Equivalence comparison operators (==, ~=) shall not be used on floating-point data types.	-

【Correct】



【Incorrect】
Uses floating-point type equivalence comparison operations (==, ~=).



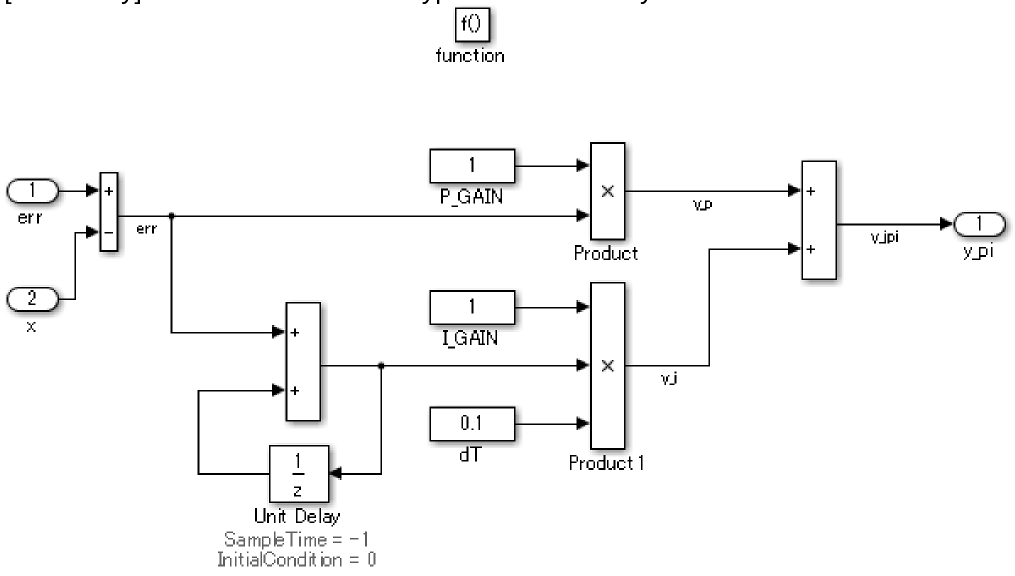
Rationale	
Sub ID	**Description**
a	Due to the characteristics of the floating-point, since the error is included in the value, the result of the equivalence comparison operation may be false when it was expected to be true.

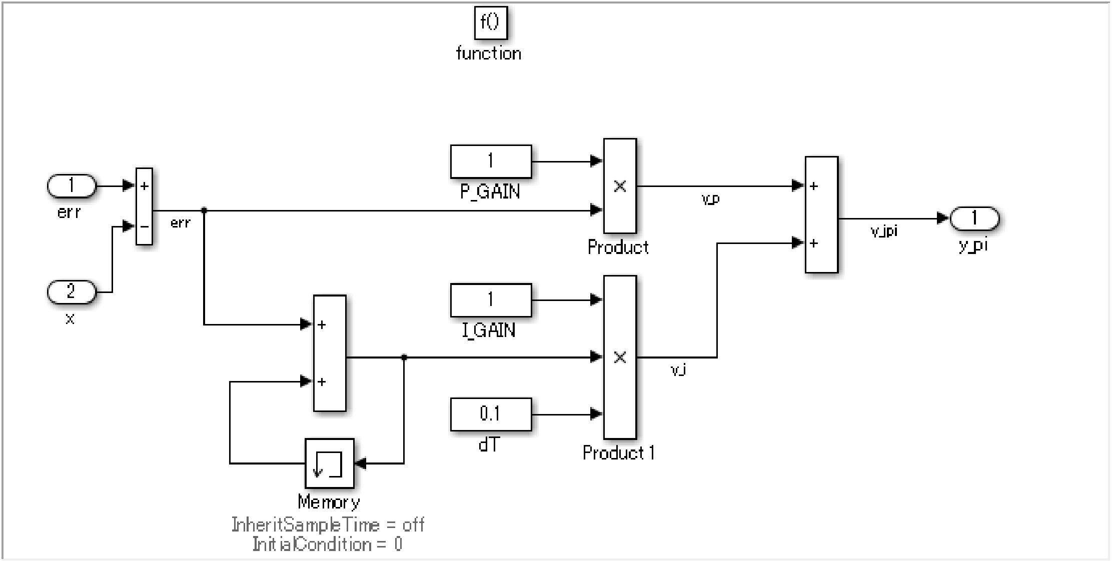
jc_0626: Usage of Lookup Table blocks

Rule ID: Title	**jc_0626: Usage of Lookup Table blocks**
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b
MATLAB® Version	All
Rule	

Sub ID	Description	Custom Parameter
a	[Lookup Table Dynamic] block parameter {Lookup Method} shall be set to “Interpolation – Use End Values”.	-
b	These [n-D Lookup Table] block parameters shall be set: - 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}.	-
Rationale		
Sub ID	**Description**	
ab	When an unexpected value is entered for [Lookup Table], the output is determined by using the extrapolation method and can become an impossible value or cause the [Lookup Table] output to overflow.	

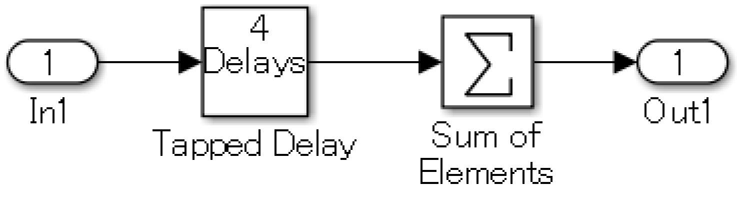
jc_0623: Usage of continuous-time Delay blocks and discrete-time Delay blocks

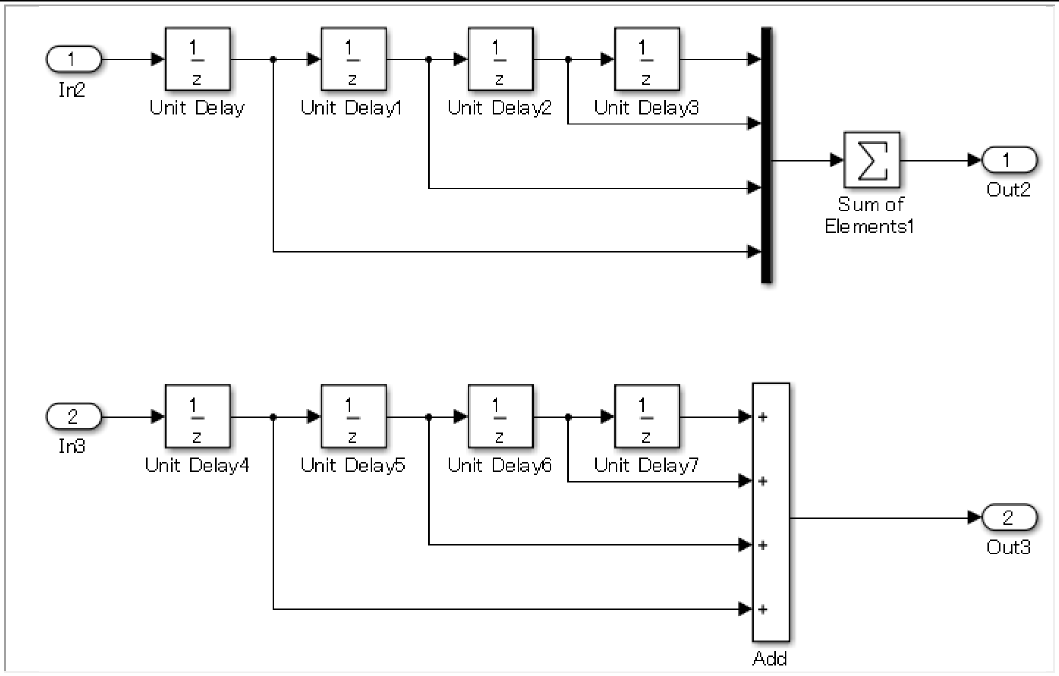
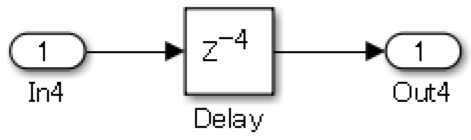
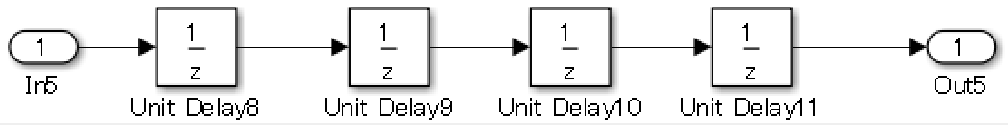
Rule ID: Title	**jc_0623: Usage of continuous-time Delay blocks and discrete-time Delay blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Unit Delay] or [Delay] shall be used in a discrete model or subsystem. [Memory] shall be used in a continuous type model or subsystem.	-
	【Correct】 [Unit Delay] is used in the discrete type model or subsystem.  f() function 1 err 2 x err 1 P_GAIN Product v_p v_pi 1 y_pi 1 I_GAIN v_i 0.1 dT Product1 1/z Unit Delay SampleTime = -1 InitialCondition = 0 【Incorrect】 [Memory] is used in the discrete type model or subsystem.	

	 f() function 1 err 2 x err 1 P_GAIN Product v_p 1 I_GAIN 0.1 dT Product1 v_i v_ipi 1 y_pi Memory InheritSampleTime = off InitialCondition = 0

Rationale	
Sub ID	**Description**
a	Adherence to the rule improves readability of the model.

jc_0624: Usage of Tapped Delay blocks/Delay blocks

Rule ID: Title	**jc_0624: Usage of Tapped Delay blocks/Delay blocks**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When holding previous past values, [Tapped Delay] shall be used to create a vector signal from all held values.	-
	【Correct】 [Tapped Delay] is used.  1 In1 4 Delays Tapped Delay Sum of Elements 1 Out1 【Incorrect】 [Tapped Delay] is not used.	

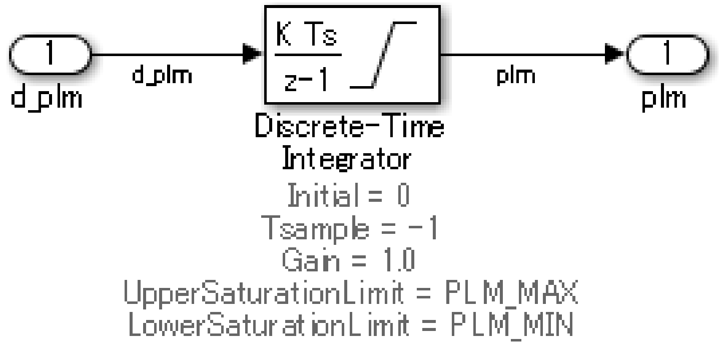
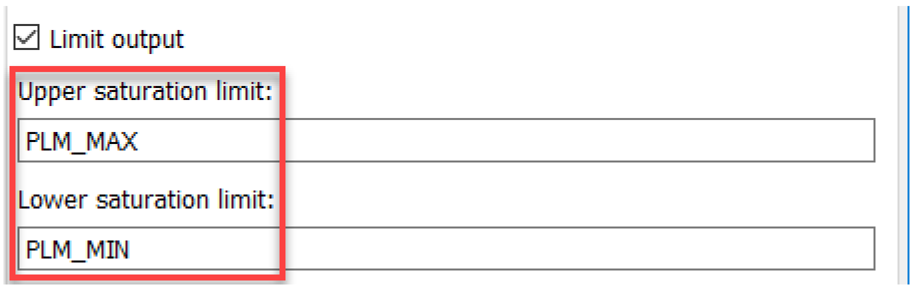
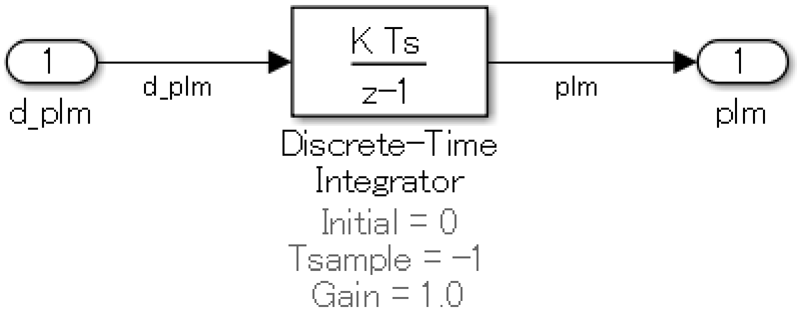
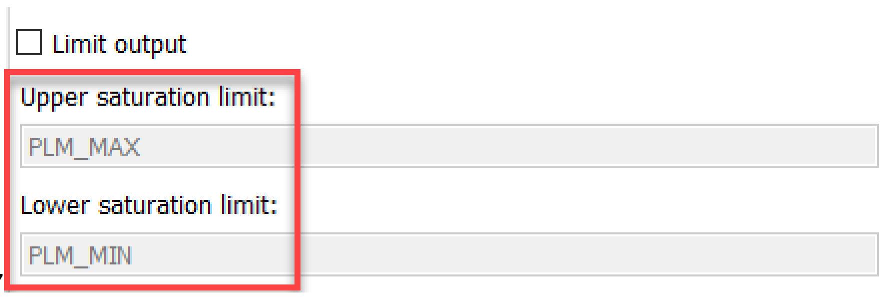
	 1 In2 Unit Delay Unit Delay1 Unit Delay2 Unit Delay3 Sum of Elements1 1 Out2 2 In3 Unit Delay4 Unit Delay5 Unit Delay6 Unit Delay7 Add 2 Out3	
b	When holding past values, [Delay] shall be used to obtain the oldest value only.	-
	【Correct】 [Delay] is used.  【Incorrect】 [Delay] is not used.  1 In5 Unit Delay8 Unit Delay9 Unit Delay10 Unit Delay11 1 Out5	

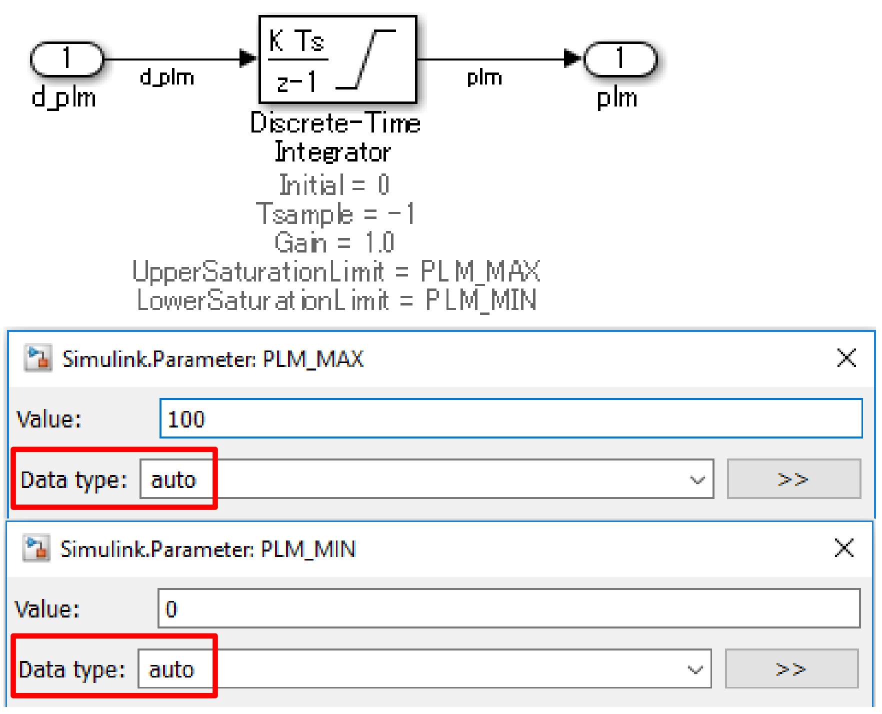
Rationale	
Sub ID	**Description**
a	[Tapped Delay] is set with arrays that hold past values, which improves code readability to assist code efficiency.
B	Improves model readability and code efficiency.

jc_0627: Usage of Discrete-Time Integrator blocks

Rule ID: Title	**jc_0627: Discrete-Time Integrator blocks**
Sub ID Recommendations	NA-MAAB: a JMAAB: a, b
MATLAB® Version	All
Rule	

Sub ID	Description	Custom Parameter

a	[Discrete-Time Integrator] block parameters {Upper saturation limit} and {Lower saturation limit} shall be defined.	-
	【Correct】 Block parameters {Upper saturation limit} and {Lower saturation limit} are defined.  1 d_plm d_plm K Ts z-1 plm 1 plm Discrete-Time Integrator Initial = 0 Tsample = -1 Gain = 1.0 UpperSaturationLimit = PLM_MAX LowerSaturationLimit = PLM_MIN  ☑ Limit output Upper saturation limit: PLM_MAX Lower saturation limit: PLM_MIN 【Incorrect】 Block parameters {Upper saturation limit} and {Lower saturation limit} are not defined.  1 d_plm d_plm K Ts z-1 plm 1 plm Discrete-Time Integrator Initial = 0 Tsample = -1 Gain = 1.0  ☐ Limit output Upper saturation limit: PLM_MAX Lower saturation limit: PLM_MIN Jc_0627	
b	When [Discrete-Time Integrator] block parameters {Upper saturation limit} and {Lower saturation limit} are defined as `Simulink.Parameter`, parameter {Data type} shall be set to “auto”.	-
	【Correct】 {Data type} is set to “auto”.	



Simulink.Parameter: PLM_MAX

Value: 100

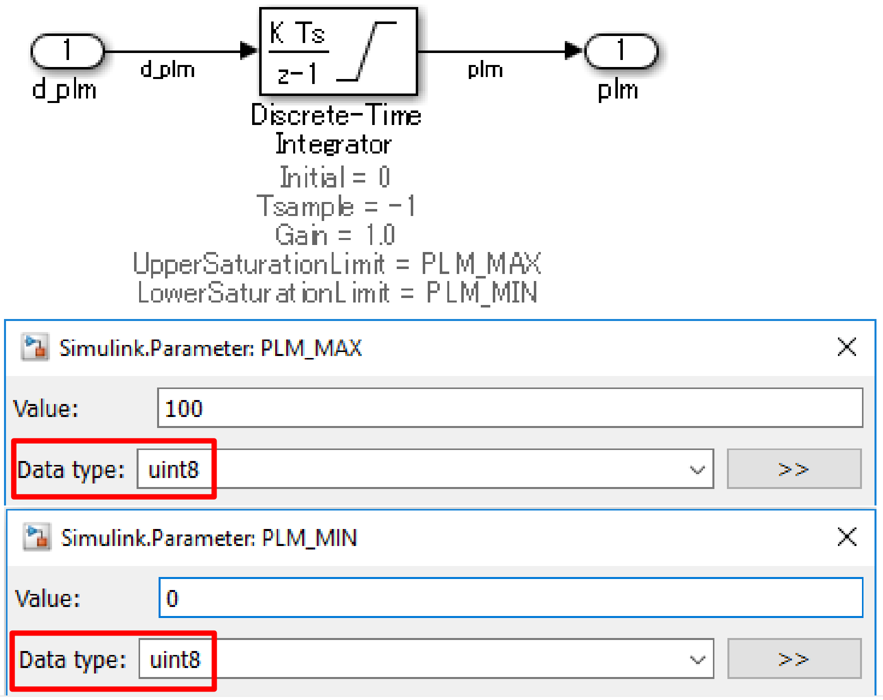
Data type: auto

Simulink.Parameter: PLM_MIN

Value: 0

Data type: auto

【Incorrect】
{Data type} is not set to "auto".



Simulink.Parameter: PLM_MAX

Value: 100

Data type: uint8

Simulink.Parameter: PLM_MIN

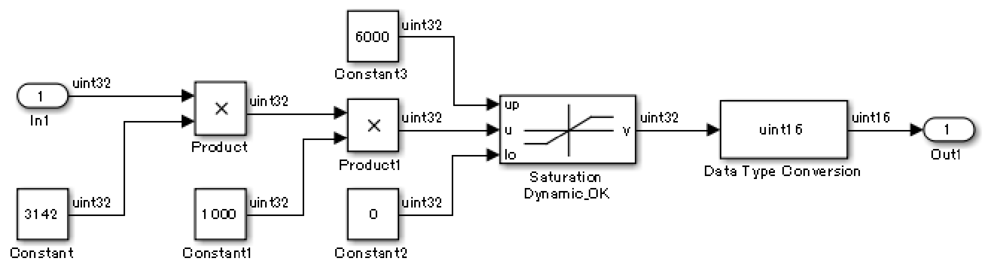
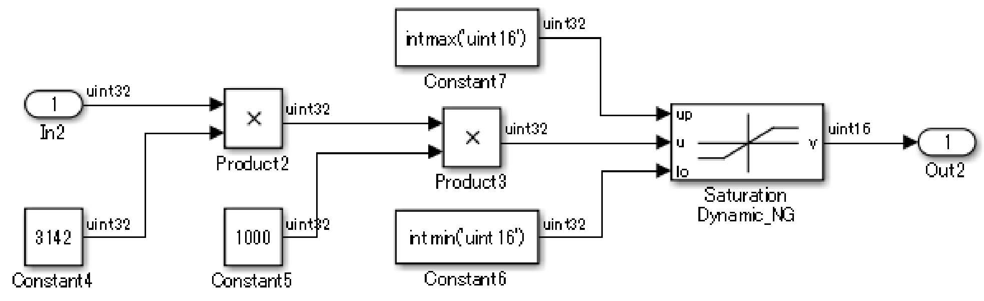
Value: 0

Data type: uint8

Rationale

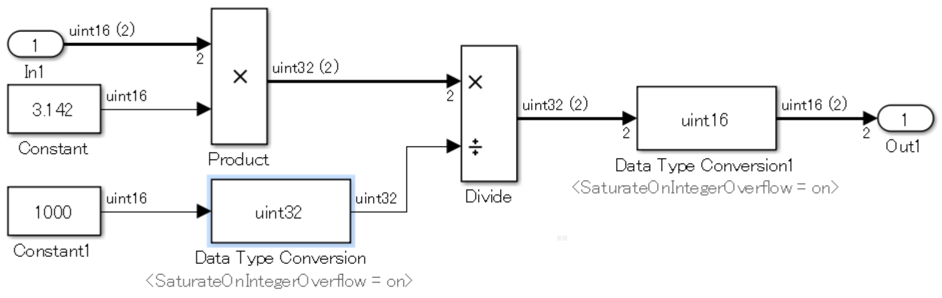
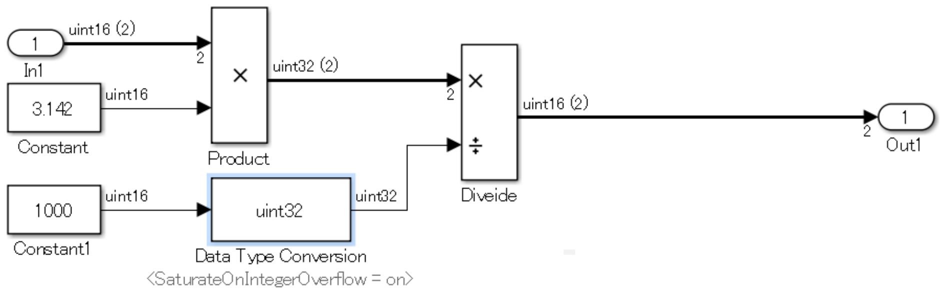
Sub ID	Description
a	Avoids block output overflow and prevents other computation blocks that use the output of this block from producing unexpected results.
B	Simulation errors occur when {Data type} is set to a value other than "auto", "single", or "double".

jc_0628: Usage of Saturation blocks

Rule ID: Title	**jc_0628: Usage of Saturation blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Saturation] and [Saturation Dynamic] shall be used to limit physical quantity. Type conversion shall not be used. The upper and lower limits for the data type maximum and minimum values shall not be set.	-
	【Correct】 [Saturation Dynamic] is used to limit physical quantity. Type conversion is not being used.  1 In1 uint32 3142 Constant Product 1000 Constant1 6000 Constant3 Product1 0 Constant2 up u lo y Saturation Dynamic_OK uint16 Data Type Conversion Out1 【Incorrect】 [Saturation Dynamic] 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.  1 In2 uint32 3142 Constant4 Product2 1000 Constant5 intmax('uint16') Constant7 Product3 intmin('uint16') Constant6 up u lo y Saturation Dynamic_NG uint16 Out2	
Rationale		
Sub ID	**Description**	
a	Consistent use of [Saturation] improves maintainability of the model.	

jc_0651: Implementing a type conversion

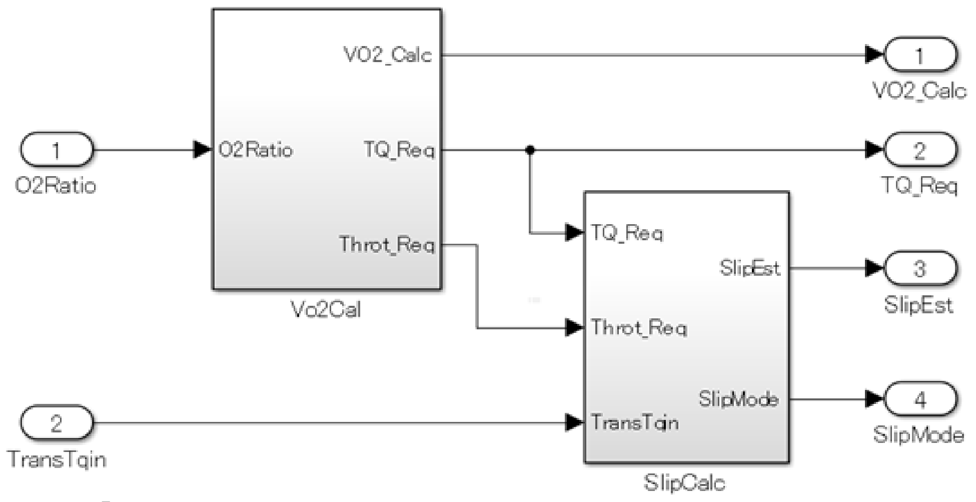
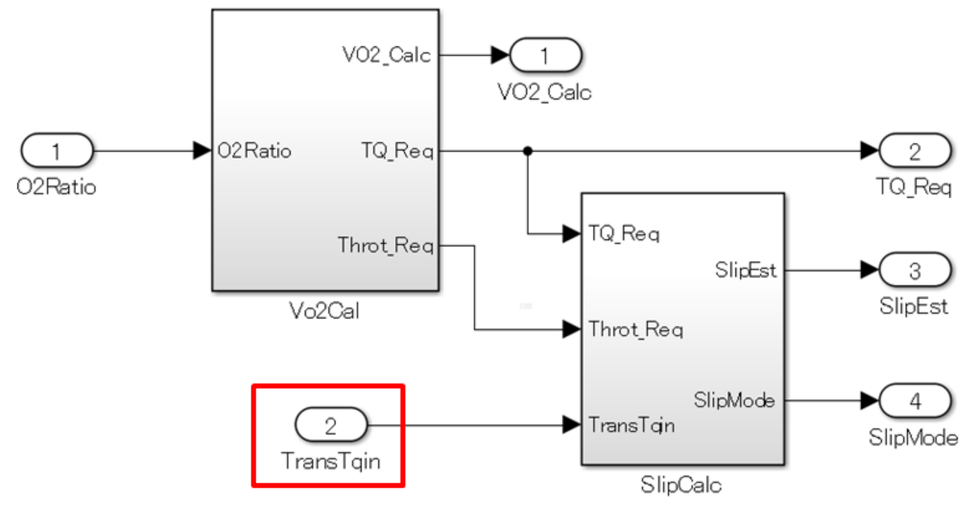
Rule ID: Title	**jc_0651: Implementing a type conversion**

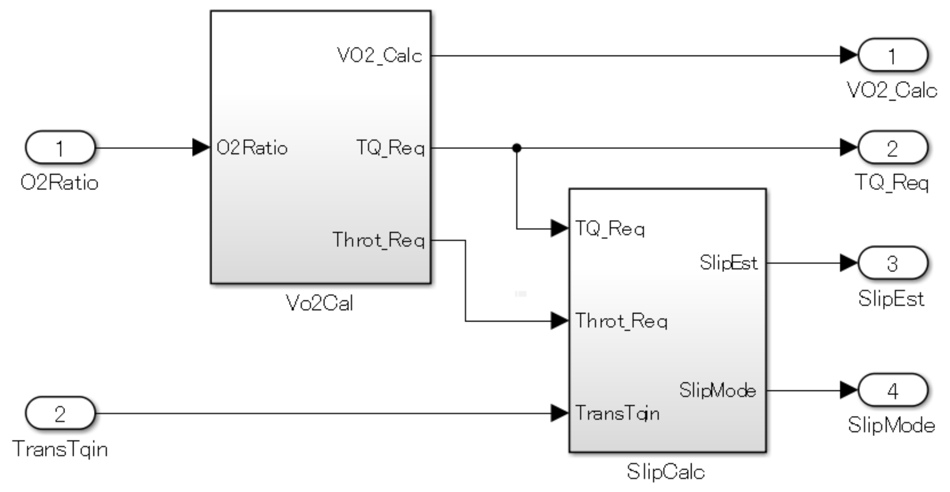
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Data Type Conversion] shall be used when changing the data type of the block output signal.	-
	【Correct】 [Data Type Conversion] is used to convert the data type of the [Divide] output signal.  1 In1 uint16 (2) 3.142 Constant uint16 1000 Constant1 uint16 2 Product uint32 (2) uint32 Data Type Conversion <SaturateOnIntegerOverflow = on> uint32 2 Divide uint32 (2) 2 uint16 Data Type Conversion1 <SaturateOnIntegerOverflow = on> uint16 (2) 2 1 Out1 【Incorrect】 [Data Type Conversion] is not used to convert the data type of the [Divide] output signal.  1 In1 uint16 (2) 3.142 Constant uint16 1000 Constant1 uint16 2 Product uint32 (2) uint32 Data Type Conversion <SaturateOnIntegerOverflow = on> uint32 2 Diveide uint16 (2) 2 1 Out1	
Rationale		
Sub ID	**Description**	
a	Dividing the math operations and type cast can help to clarify the order of execution and data type for each expression.	

3.6. Other blocks

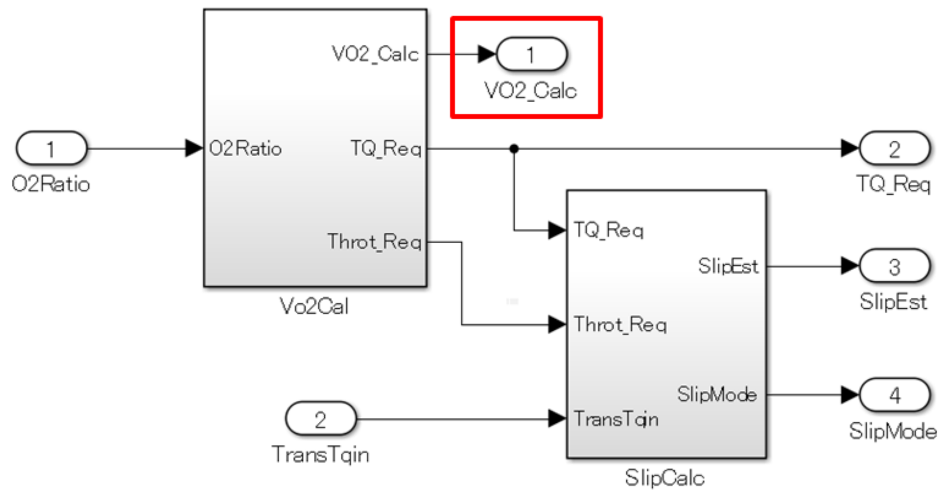
db_0042: Usage of Inport and Outport blocks

Rule ID: Title	**db_0042: Usage of Inport and Outport blocks**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Inport] shall be positioned on the left side of the diagram, but can be moved to prevent the crossing of signals.	-
	【Correct】 [Inport] is positioned on the left side of the diagram.	

	 VO2_Calc 1 O2Ratio O2Ratio TQ_Req Throt_Req Vo2Cal 2 TransTqin TQ_Req SlipEst Throt_Req SlipMode TransTqin SlipCalc 1 VO2_Calc 2 TQ_Req 3 SlipEst 4 SlipMode 【Incorrect】 [Inport] is not positioned on the left side of the diagram.  VO2_Calc 1 VO2_Calc 1 O2Ratio O2Ratio TQ_Req Throt_Req Vo2Cal 2 TransTqin TQ_Req SlipEst Throt_Req SlipMode TransTqin SlipCalc 2 TQ_Req 3 SlipEst 4 SlipMode	
b	[Outport] shall be positioned on the right side of the diagram, but can be moved to prevent the crossing of signals.	-
	【Correct】 [Outport] is positioned on the right side of the diagram.	

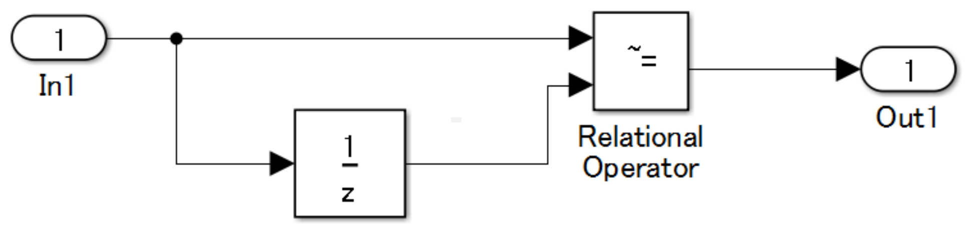


【Incorrect】
[Outport] is not positioned on the right side of the diagram.

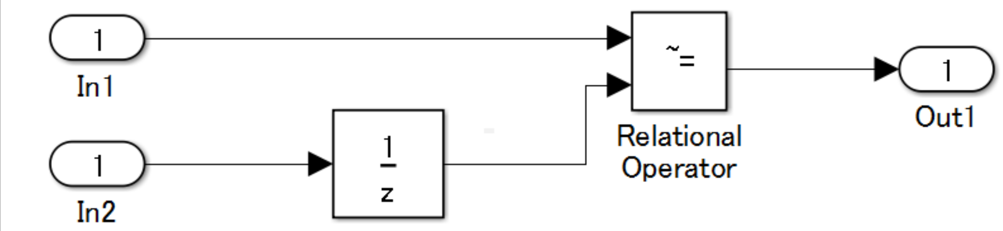


c	Duplicate [Inport] shall be prohibited.	-

【Correct】
One [Inport] is used..

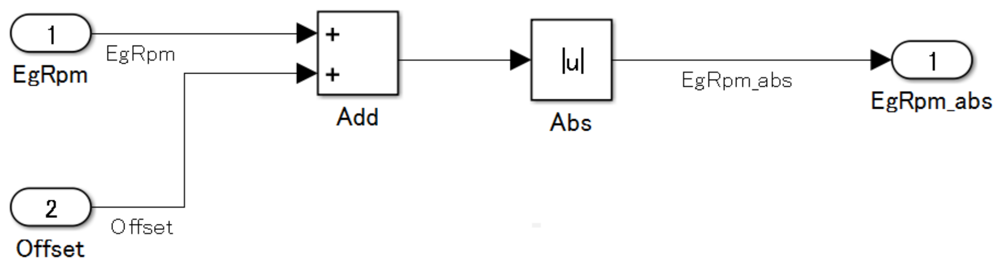
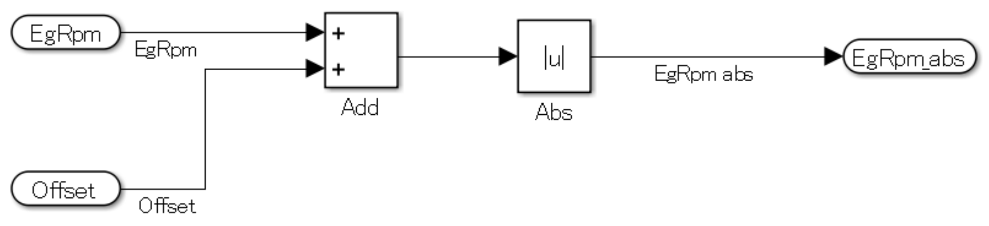


【Incorrect】
[Inport] is duplicated.

	 1 In1 1 In2 1/z ~= Relational Operator 1 Out1

Rationale	
Sub ID	**Description**
abc	Defined operation rules improve readability.

jc_0081: Inport/Outport block icon display

Rule ID: Title	**jc_0081: Inport/Outport block icon display**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	For [Inport] and [Outport], block parameter {Icon Display} shall be set to "Port number".	-
	【Correct】 The {icon display} for [Inport] and [Outport] is "Port number".  1 EgRpm EgRpm 2 Offset Offset + + Add |u| Abs EgRpm_abs 1 EgRpm_abs 【Incorrect】 The {icon display} for [Inport] and [Outport] is not "Port number".  EgRpm EgRpm Offset Offset + + Add |u| Abs EgRpm abs EgRpm_abs	

Rationale	
Sub ID	**Description**
a	・Improves readability by displaying the port number of [Inport] and [Outport]. ・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] or [Outport] the same as the block names.

na_0011: Scope of Goto/From blocks

Rule ID: Title	**na_0011: Scope of Goto/From blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Goto] block parameter {tag visibility} shall be set to “Local”.	-
Rationale		
Sub ID	**Description**	
a	・When hierarchies of [Goto] and corresponding [From] are different, the connection relationships can be difficult to understand. ・Simulation errors can occur when hierarchies of [Goto] and corresponding [From] are different and a virtual subsystem changes to an Atomic subsystem.	

jc_0161: Definition of Data Store Memory blocks

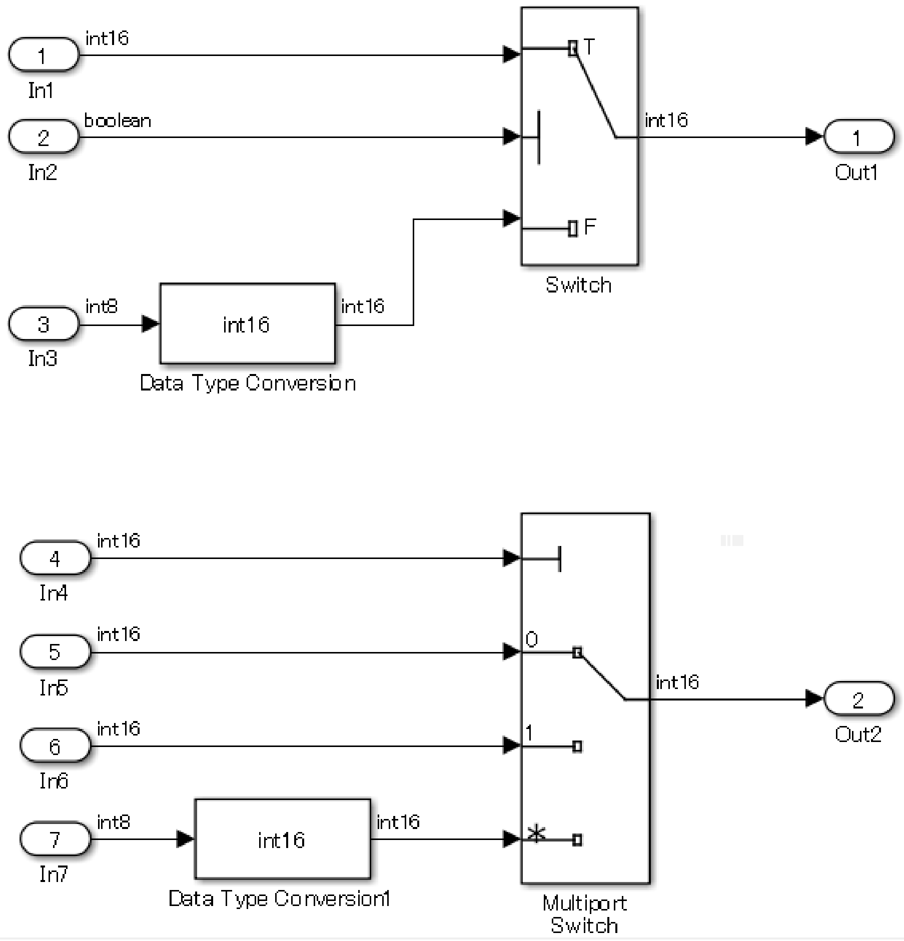
Rule ID: Title	**jc_0161: Definition of Data Store Memory blocks**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The smallest scope level shall be used to define [Data Store Memory].	-
b	Only data required for execution and code generation shall be defined in [Data Store Memory].	-
Rationale		
Sub ID	**Description**	
a	・Readability improves when usage is limited.	
b	・Unused [Data Store Memory] data can affect maintenance and operability.	

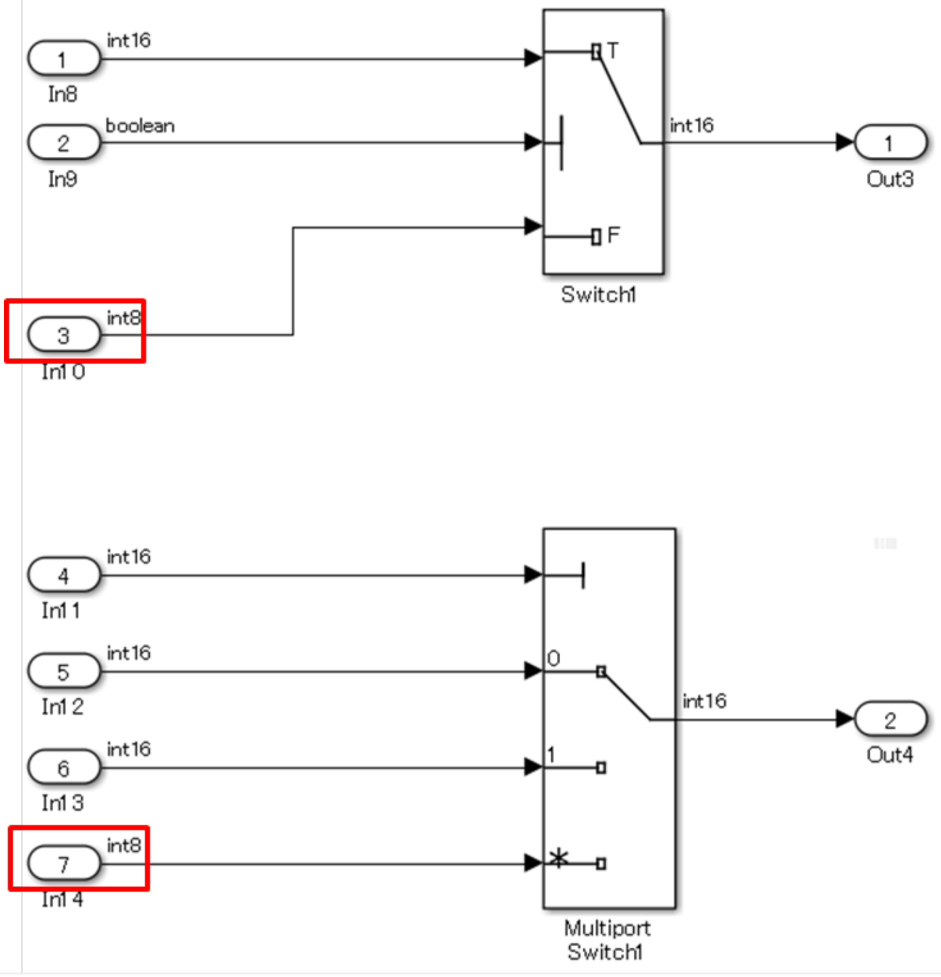
jc_0141: Usage of Switch blocks

Rule ID: Title	**jc_0141: Usage of Switch blocks**
Sub ID Recommendations	NA-MAAB: a JMAAB: a
MATLAB® Version	All
Rule	

Sub ID	Description	Custom Parameter
a	The second [Switch] input condition shall be a logical type. [Switch] block parameter {Criteria for passing first input} shall be set to “u2~=0”.	-
Rationale		
Sub ID	**Description**	
a	· It is easier to understand specifications when the configuration is applied by using Simulink blocks rather than by writing operation expressions in blocks.	

jc_0650: Block input/output data type with switching function

Rule ID: Title	**jc_0650: Block input/output data type with switching function**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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.	-
	【Correct】 The data type for the data port and output port is the same. 	

	【Incorrect】 The data port and output port have different data types. 

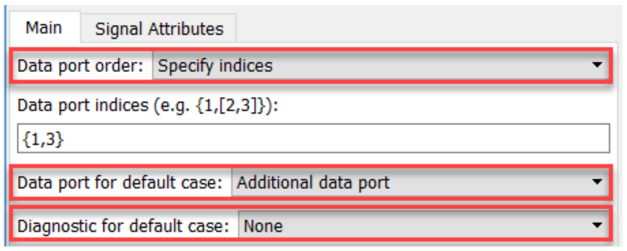
Rationale	
Sub ID	Description
a	Prevents implicit data conversion.

jc_0630: Usage of Multiport Switch blocks

Rule ID: Title	jc_0630: Usage of Multiport Switch blocks	
Sub ID Recommendations	NA-MAAB: a, c JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	Description	Custom Parameter
a	[Multiport Switch] block parameter {Number of data ports} shall be two or more.	-
b	The input to the [Multiport Switch] control port shall be an unsigned integer.	-
c	When [Multiport Switch] block parameter {data port order}is set to "Specify indices", these block parameters shall be set: ・{Data port for default case} to “Additional data port”.	-

・{Diagnostic for default case} to “None”.

【Correct】



Main | Signal Attributes

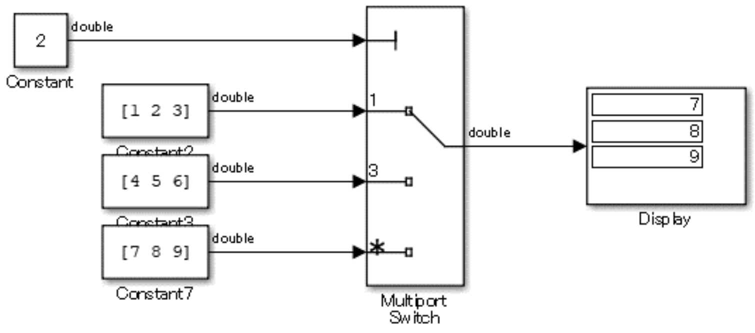
Data port order: Specify indices

Data port indices (e.g. {1,[2,3]}):

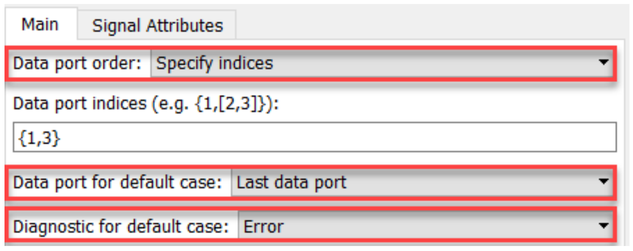
{1,3}

Data port for default case: Additional data port

Diagnostic for default case: None



【Incorrect】



Main | Signal Attributes

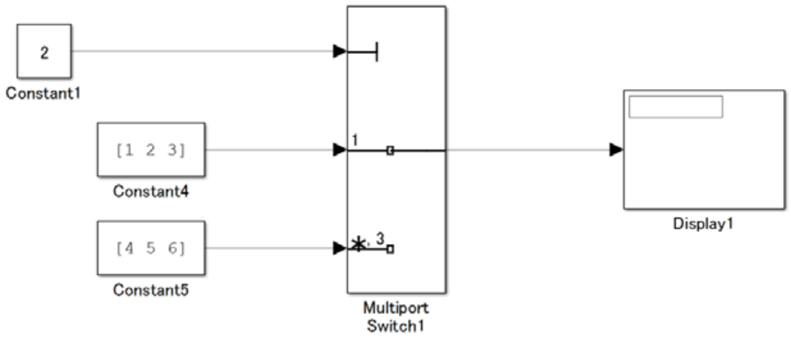
Data port order: Specify indices

Data port indices (e.g. {1,[2,3]}):

{1,3}

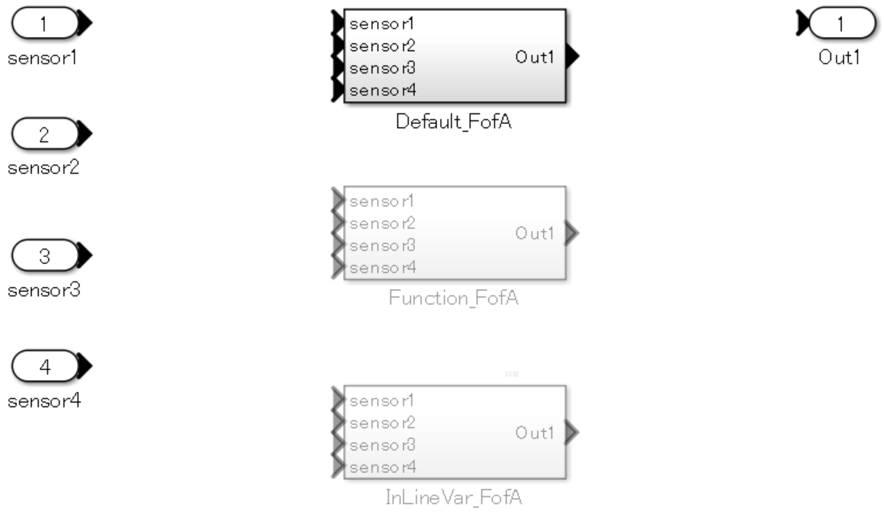
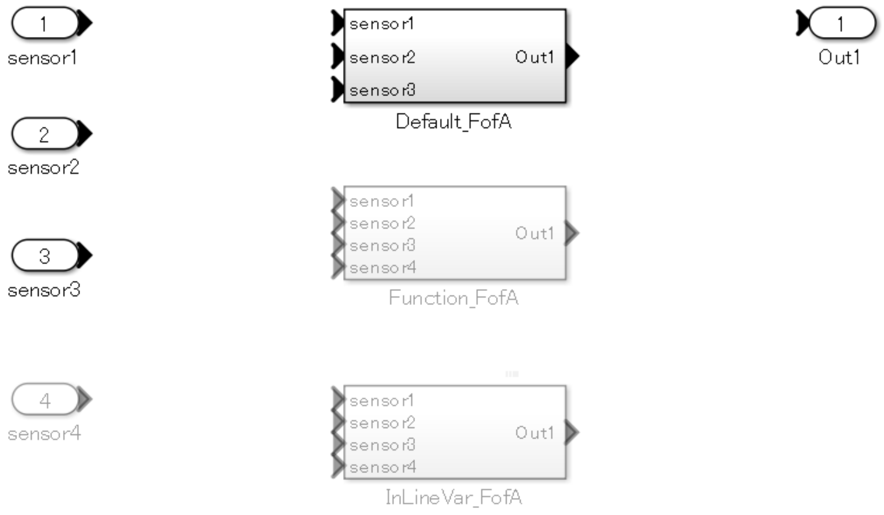
Data port for default case: Last data port

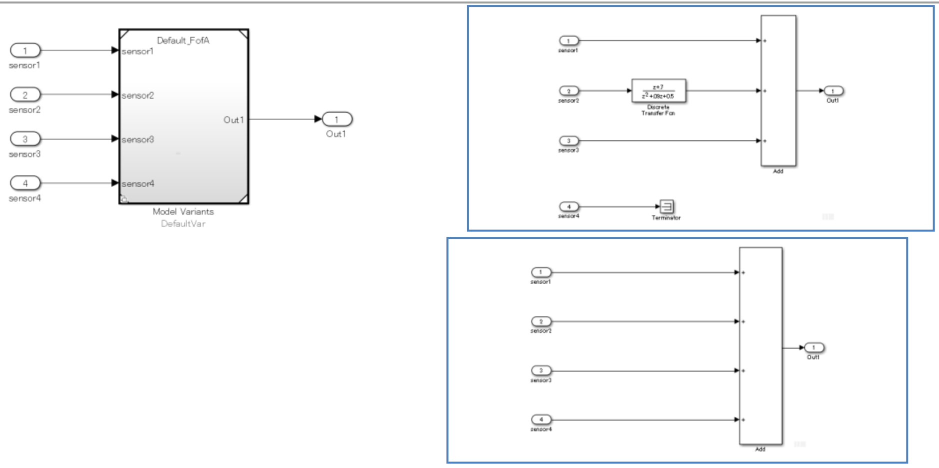
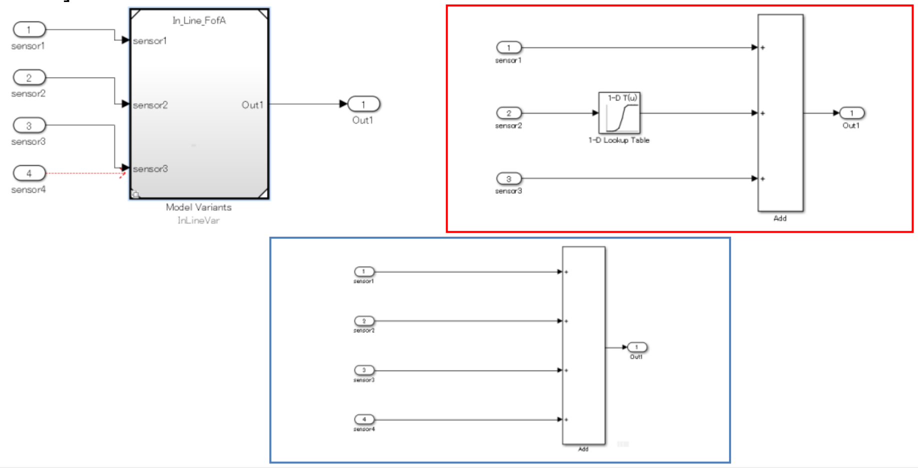
Diagnostic for default case: Error



Rationale	
Sub ID	**Description**
a	・Unintended output can occur when there is only one data port because the block changes to extract scalars from vectors.
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.
c	・When block parameter {Data port order} is set to “Specify indices”, any value that is input to [Multiport Switch], 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.

na_0020: Number of inputs to variant subsystems

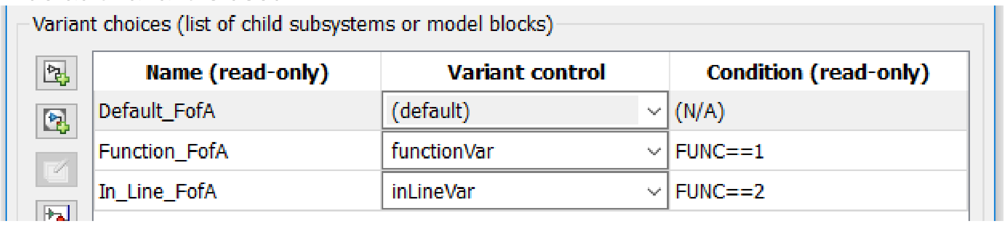
Rule ID: Title	**na_0020: Number of inputs to variant subsystems**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The number of inputs/outputs of a [Variant Subsystem] and its child subsystem or [Model Reference] shall be the same.	-
	【Correct】 The number of inputs to the child subsystem is the same.  【Incorrect】 The number of inputs to the child subsystem is different. 	
b	The number of inputs/outputs for [Model Variants] shall be that same as its referenced model.	-
	【Correct】 The number of inputs to the referenced model is the same as for [Model Variants] .	

	 Default_FofA sensor1 sensor2 sensor3 sensor4 Out1 Model Variants DefaultVar Discrete Transfer Fcn Add Terminator 【Incorrect】 The number of inputs to the referenced model is different than the inputs to [Model Variants].  In_Line_FofA sensor1 sensor2 sensor3 sensor4 Out1 Model Variants InLineVar 1-D T(u) 1-D Lookup Table Add
Rationale	
Sub ID	**Description**
ab	・Unconnected signals can be unintentionally overlooked when the number of inputs/outputs is different.

na_0036: Default variant

Rule ID: Title	**na_0036: Default variant**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
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.	-

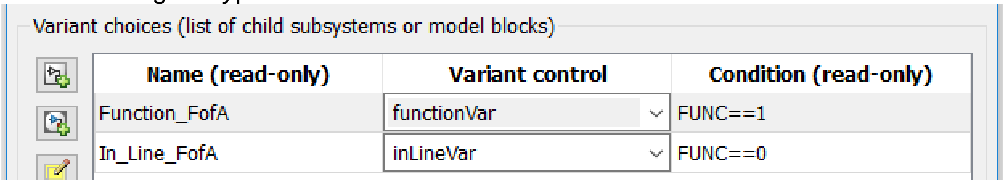
【Correct】
A default variant is used.



Variant choices (list of child subsystems or model blocks)

Name (read-only)	Variant control	Condition (read-only)
Default_FofA	(default)	(N/A)
Function_FofA	functionVar	FUNC==1
In_Line_FofA	inLineVar	FUNC==2

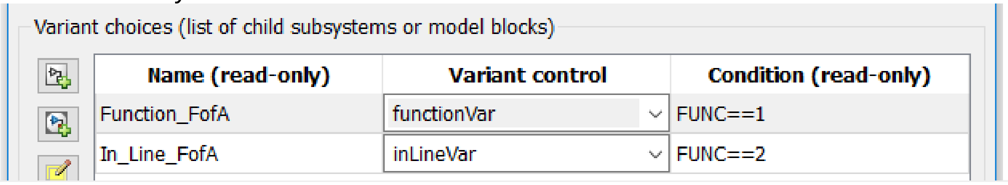
【Correct】
FUNC is a logical type.



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

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

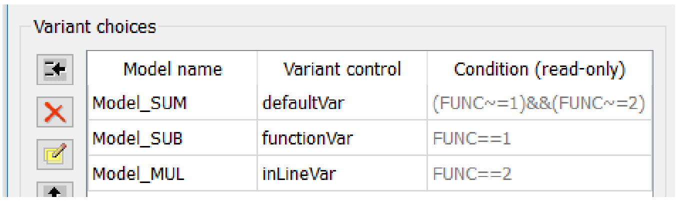


Variant choices (list of child subsystems or model blocks)

Name (read-only)	Variant control	Condition (read-only)
Function_FofA	functionVar	FUNC==1
In_Line_FofA	inLineVar	FUNC==2

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

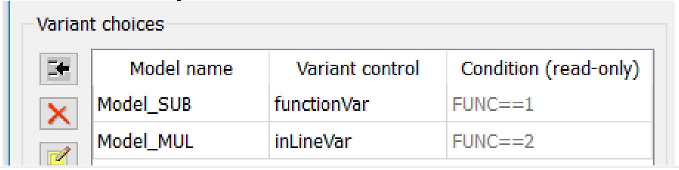
【Correct】
The condition is set so that all values for the conditional variable are covered.



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

【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

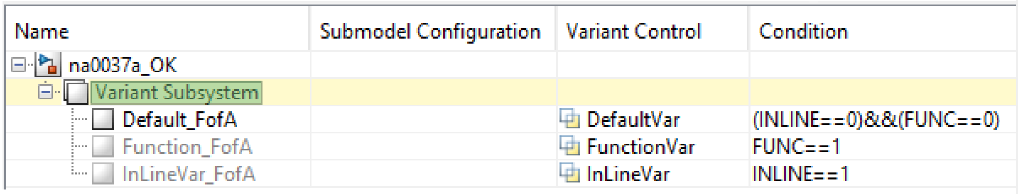
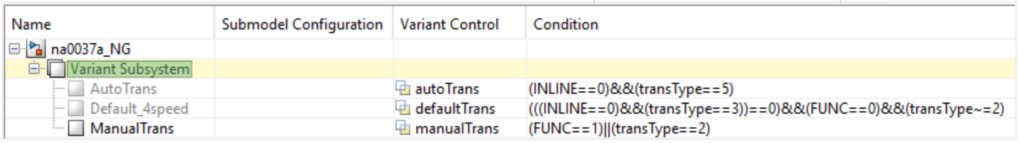
Sub ID	**Description**
ab	· Prevents the omission of conditions. · There may not be an active subsystem when conditions are omitted.

na_0037: Use of single variable for variant condition

Rule ID: Title	**na_0037: Use of single variable for variant condition**

Sub ID Recommendations	NA-MAAB: a JMAAB: a
MATLAB® Version	All

Rule

Sub ID	**Description**	**Custom Parameter**
a	Variant conditions shall be used to prohibit compound conditions that are formed from multiple variables. Exception: When using default variants, conditional expressions that are formed from multiple variables can be used.	-
	【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. 【Incorrect】 The variant condition is set by a compound condition that is formed from multiple variables. 	

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

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	**Description**
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.

4. Stateflow

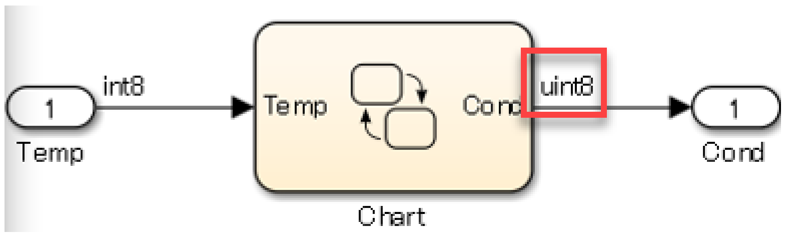
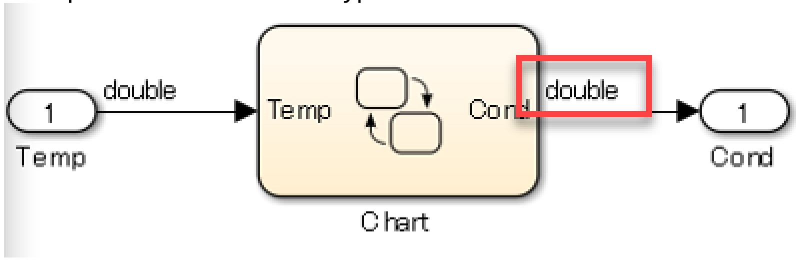
4.1. Stateflow blocks/data/events

db_0122: Stateflow and Simulink interface signals and parameters

Rule ID: Title	**db_0122: Stateflow and Simulink interface signals and parameters**
Sub ID Recommendations	NA-MAAB: a JMAAB: a
MATLAB® Version	All

Rule

Sub ID	**Description**	**Custom Parameter**
a	[Chart] parameter {Use Strong Data Typing with Simulink I/O} shall be selected so that strong data typing between	-

	a Stateflow chart and Simulink is permitted. Note: {Use Strong Data Typing with Simulink I/O} is available only when [Chart] property {Action Language} is set to “C”.	
	【Correct】 Parameter {Use Strong Data Typing with Simulink I/O} is selected, so the input and output are set to “uint8” type.  int8 1 Temp Temp Cond uint8 Chart 1 Cond 【Incorrect】 Parameter {Use Strong Data Typing with Simulink I/O} is not selected, so the input and output are set to “double” type.  double 1 Temp Temp Cond double Chart 1 Cond	

Rationale	
Sub ID	**Description**
a	· When {Use Strong Data Typing with Simulink I/O} is not selected, the Simulink signal data type that can input and output to [Chart] is set to “double” type. As a result, type conversion is required prior to input and after output, which increases the number of blocks and decreases readability. · When {Use Strong Data Typing with Simulink I/O} is not selected, the Simulink signal data type that can input and output to [Chart] is set to “double” type. However, input data of any type in [Chart] can connect directly with that signal. When these two signals have different data types, an implicit data type conversion occurs. By selecting {Use Strong Data Typing with Simulink I/O}, the implicit data type conversion does not take place and a data type inconsistency error is generated. This prevents misunderstandings due to differences in data type, thus improving readability.

db_0123: Stateflow port names

Rule ID: Title	**db_0123: Stateflow port names**
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported
MATLAB® Version	All

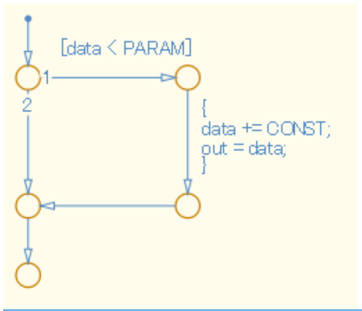
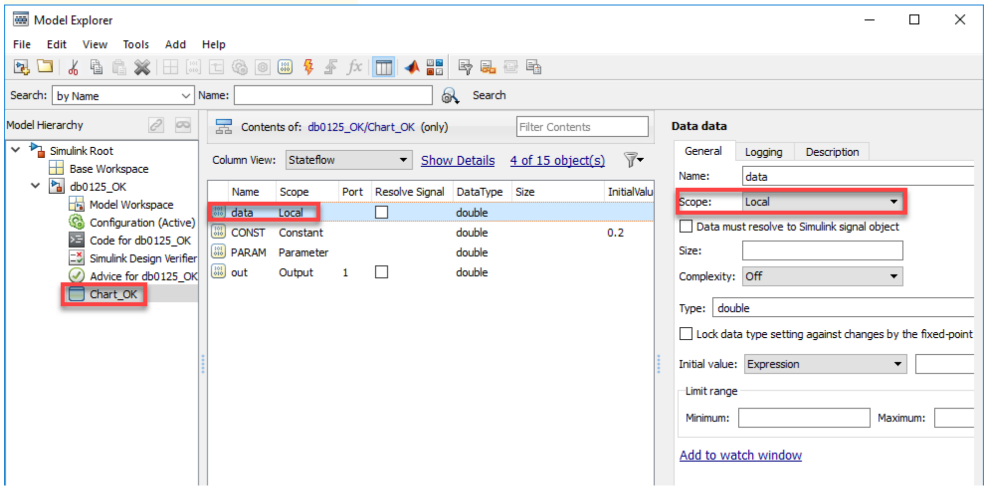
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The name of a Stateflow input/output shall be the same as the corresponding signal.	-

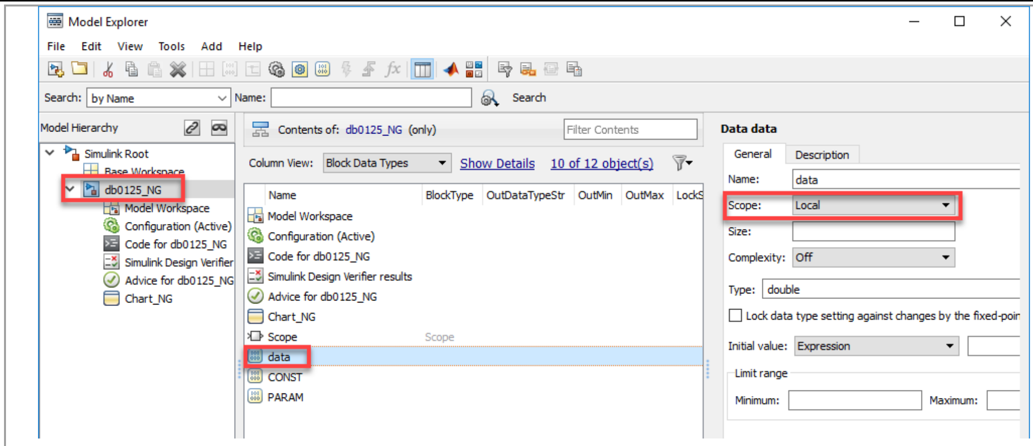
	Exception: Reusable Stateflow blocks can have different port names.	

Rationale	
Sub ID	**Description**
a	・Improves readability. ・Code generation may not be possible.

db_0125: Stateflow local data

Rule ID: Title	**db_0125: Stateflow local data**	
Sub ID Recommendations	NA-MAAB: a, b, c, d JMAAB: a, b, c, d	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Scope shall not define “Local” local data at the machine level.	-

	【Correct】  [data < PARAM]
	1
	2
	{
	data += CONST;
	out = data;
	}  Model Explorer
	Contents of: db0125_OK/Chart_OK (only)
	Column View: Stateflow
	Name / Scope / Port / Resolve Signal / DataType / Size / InitialValue
	data / Local / / ☐ / double / /
	CONST / Constant / / / double / / 0.2
	PARAM / Parameter / / / double / /
	out / Output / 1 / ☐ / double / /
	Data data
	Name: data
	Scope: Local
	☐ Data must resolve to Simulink signal object
	Size:
	Complexity: Off
	Type: double
	☐ Lock data type setting against changes by the fixed-point
	Initial value: Expression
	Limit range Minimum: Maximum: File Edit View Tools Add Help
	Search: by Name Name: Search
	Model Hierarchy
	Simulink Root
	Base Workspace
	db0125_OK
	Model Workspace
	Configuration (Active)
	Code for db0125_OK
	Simulink Design Verifier
	Advice for db0125_OK
	Chart_OK
	Filter Contents
	Show Details
	4 of 15 object(s)
	General Logging Description
	Add to watch window
	【Incorrect】
	Scope has set “Local” local data at the machine level.	



Model Explorer

File Edit View Tools Add Help

Search: by Name Name: Search

Model Hierarchy: Simulink Root, Base Workspace, db0125_NG, Model Workspace, Configuration (Active), Code for db0125_NG, Simulink Design Verifier, Advice for db0125_NG, Chart_NG

Contents of: db0125_NG (only)

Column View: Block Data Types | Show Details | 10 of 12 object(s)

Name	BlockType	OutDataTypeStr	OutMin	OutMax	LockS
Model Workspace					
Configuration (Active)					
Code for db0125_NG					
Simulink Design Verifier results					
Advice for db0125_NG					
Chart_NG					
Scope	Scope				
data					
CONST					
PARAM					

Data data

General | Description

Name: data

Scope: Local

Size:

Complexity: Off

Type: double

Lock data type setting against changes by the fixed-poir

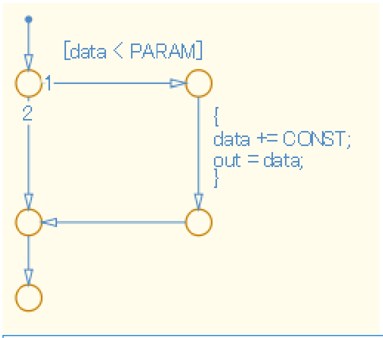
Initial value: Expression

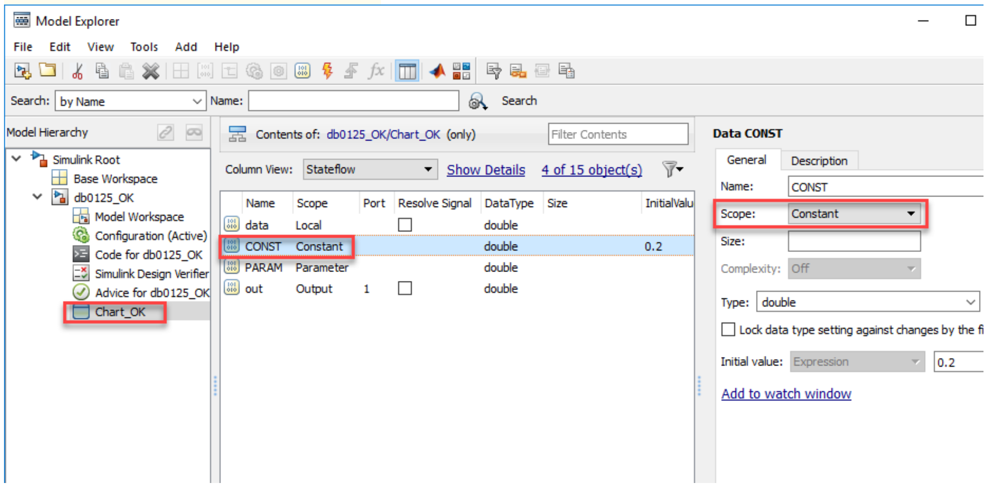
Limit range

Minimum: Maximum:

b	Scope shall not define “Constant” local data at the machine level.	-

【Correct】





Model Explorer

File Edit View Tools Add Help

Search: by Name Name: Search

Model Hierarchy: Simulink Root, Base Workspace, db0125_OK, Model Workspace, Configuration (Active), Code for db0125_OK, Simulink Design Verifier, Advice for db0125_OK, Chart_OK

Contents of: db0125_OK/Chart_OK (only)

Column View: Stateflow | Show Details | 4 of 15 object(s)

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValu
data	Local		☐	double		
CONST	Constant			double		0.2
PARAM	Parameter			double		
out	Output	1	☐	double		

Data CONST

General | Description

Name: CONST

Scope: Constant

Size:

Complexity: Off

Type: double

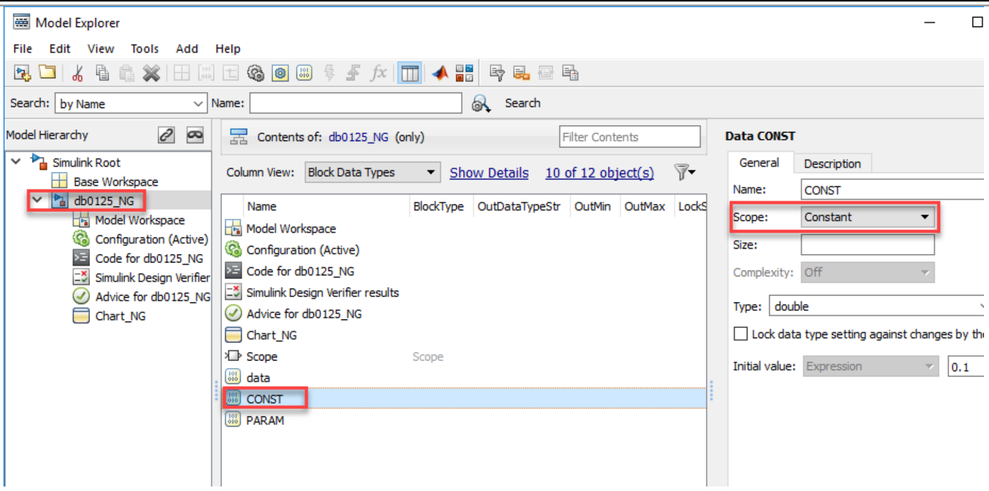
Lock data type setting against changes by the fi

Initial value: Expression 0.2

Add to watch window

【Incorrect】

Scope has set “Constant” local data at the machine level.



Model Explorer

Model Hierarchy: Simulink Root, Base Workspace, db0125_NG, Model Workspace, Configuration (Active), Code for db0125_NG, Simulink Design Verifier, Advice for db0125_NG, Chart_NG

Contents of: db0125_NG (only)

Column View: Block Data Types — 10 of 12 object(s)

Name	BlockType	OutDataTypeStr	OutMin	OutMax	LockS
Model Workspace					
Configuration (Active)					
Code for db0125_NG					
Simulink Design Verifier results					
Advice for db0125_NG					
Chart_NG					
Scope	Scope				
data					
CONST					
PARAM					

Data CONST

Name: CONST

Scope: Constant

Size:

Complexity: Off

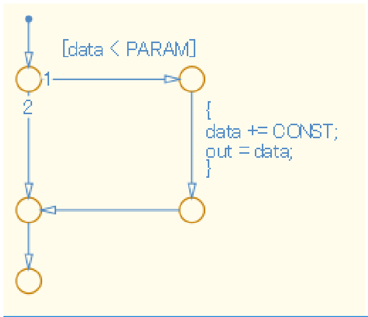
Type: double

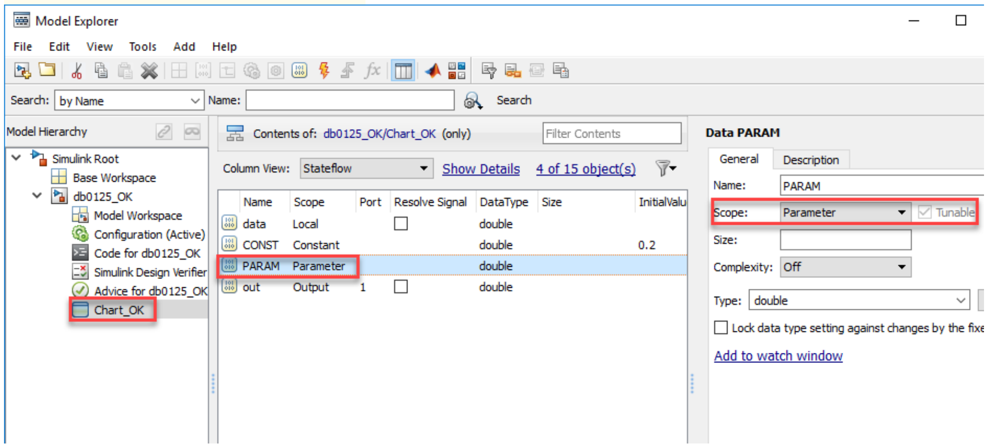
Lock data type setting against changes by th

Initial value: Expression 0.1

c	Scope shall not define “Parameter” local data at the machine level.	-

【Correct】





Model Explorer

Model Hierarchy: Simulink Root, Base Workspace, db0125_OK, Model Workspace, Configuration (Active), Code for db0125_OK, Simulink Design Verifier, Advice for db0125_OK, Chart_OK

Contents of: db0125_OK/Chart_OK (only)

Column View: Stateflow — 4 of 15 object(s)

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValu
data	Local		☐	double		
CONST	Constant			double		0.2
PARAM	Parameter			double		
out	Output	1	☐	double		

Data PARAM

Name: PARAM

Scope: Parameter ✓ Tunable

Size:

Complexity: Off

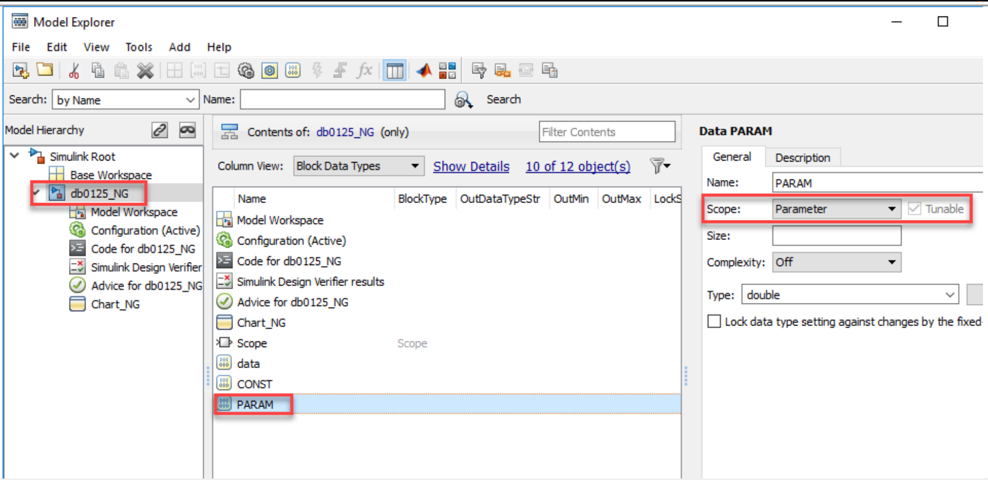
Type: double

Lock data type setting against changes by the fixe

Add to watch window

【Incorrect】

Scope has set “Parameter” local data at the machine level.



Model Explorer

File Edit View Tools Add Help

Search: by Name Name: Search

Model Hierarchy

- Simulink Root
  - Base Workspace
  - db0125_NG
    - Model Workspace
    - Configuration (Active)
    - Code for db0125_NG
    - Simulink Design Verifier
    - Advice for db0125_NG
    - Chart_NG

Contents of: db0125_NG (only) Filter Contents

Column View: Block Data Types Show Details 10 of 12 object(s)

Name	BlockType	OutDataTypeStr	OutMin	OutMax	LockS
Model Workspace					
Configuration (Active)					
Code for db0125_NG					
Simulink Design Verifier results					
Advice for db0125_NG					
Chart_NG					
Scope	Scope				
data					
CONST					
PARAM					

Data PARAM

General Description

Name: PARAM

Scope: Parameter ☑ Tunable

Size:

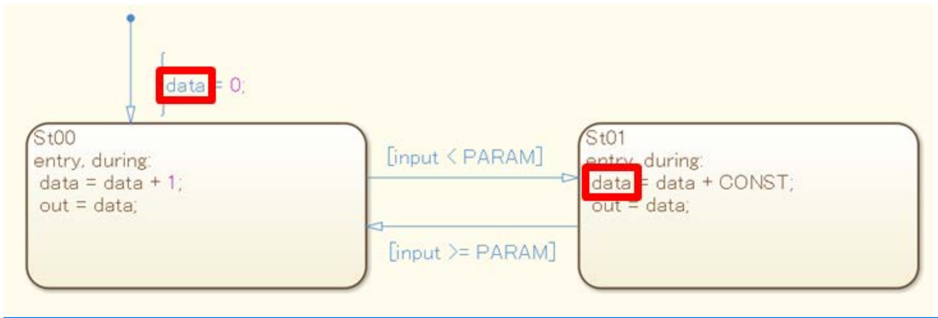
Complexity: Off

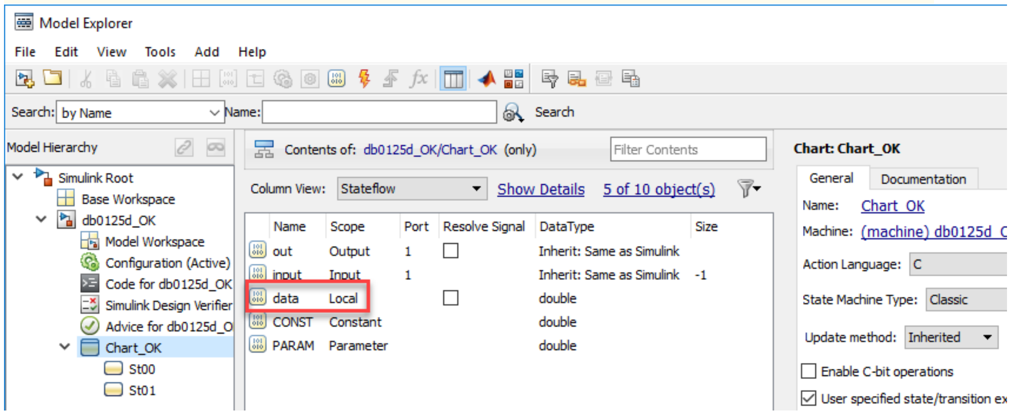
Type: double

☐ Lock data type setting against changes by the fixed

d	A Stateflow block with parent-child relationships shall not include local data with the same name.	-

【Correct】





Model Explorer

File Edit View Tools Add Help

Search: by Name Name: Search

Model Hierarchy

- Simulink Root
  - Base Workspace
  - db0125d_OK
    - Model Workspace
    - Configuration (Active)
    - Code for db0125d_OK
    - Simulink Design Verifier
    - Advice for db0125d_O
    - Chart_OK
      - St00
      - St01

Contents of: db0125d_OK/Chart_OK (only) Filter Contents

Column View: Stateflow Show Details 5 of 10 object(s)

Name	Scope	Port	Resolve Signal	DataType	Size
out	Output	1	☐	Inherit: Same as Simulink	
input	Input	1		Inherit: Same as Simulink	-1
data	Local		☐	double	
CONST	Constant			double	
PARAM	Parameter			double	

Chart: Chart_OK

General Documentation

Name: Chart_OK

Machine: (machine) db0125d_C

Action Language: C

State Machine Type: Classic

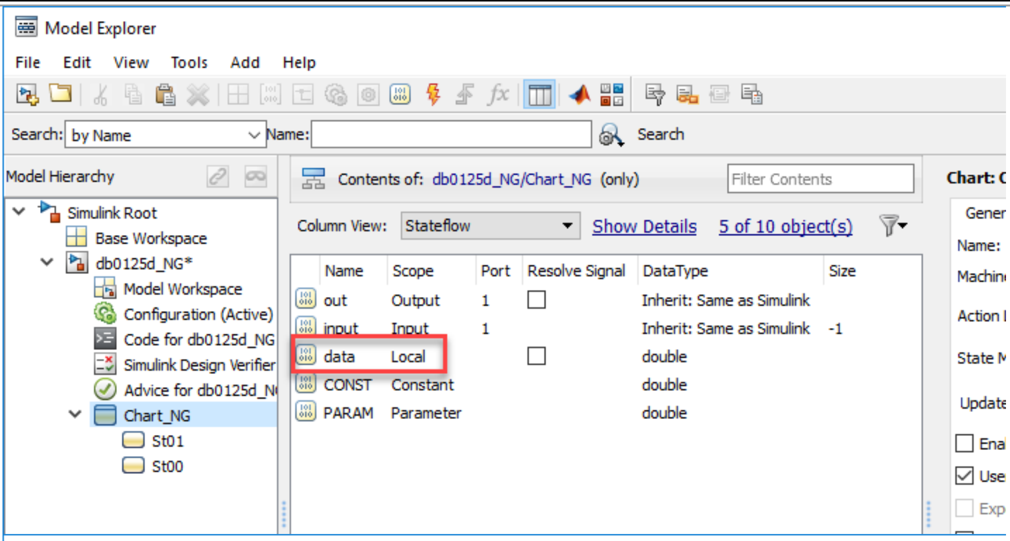
Update method: Inherited

☐ Enable C-bit operations

☑ User specified state/transition ex

【Incorrect】

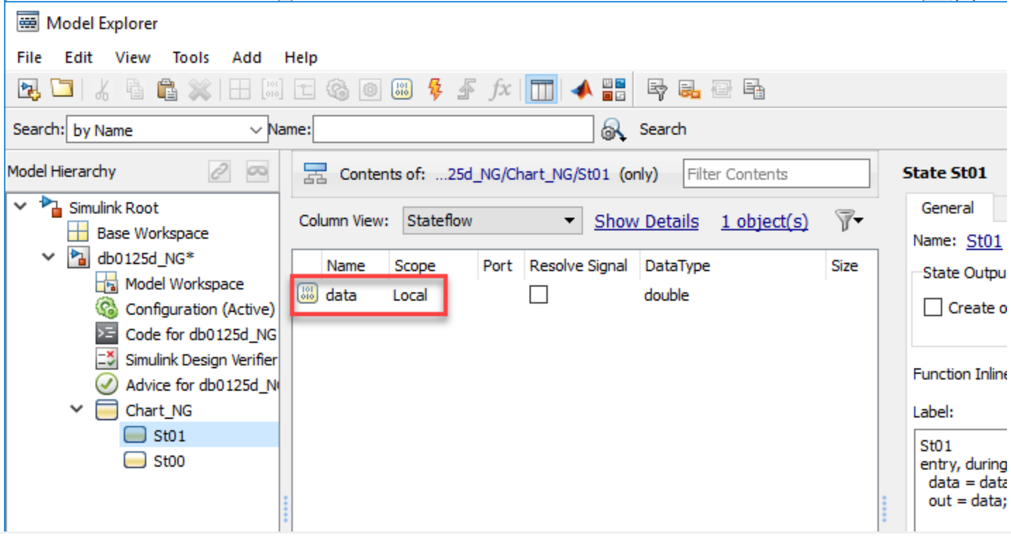
A Stateflow block with parent-child relationships has local data with the same name.



Contents of: db0125d_NG/Chart_NG (only)

Column View: Stateflow

Name	Scope	Port	Resolve Signal	DataType	Size
out	Output	1	☐	Inherit: Same as Simulink	
input	Input	1		Inherit: Same as Simulink	-1
data	Local		☐	double	
CONST	Constant			double	
PARAM	Parameter			double	



Contents of: ...25d_NG/Chart_NG/St01 (only)

Column View: Stateflow

Name	Scope	Port	Resolve Signal	DataType	Size
data	Local		☐	double	

Name: St01

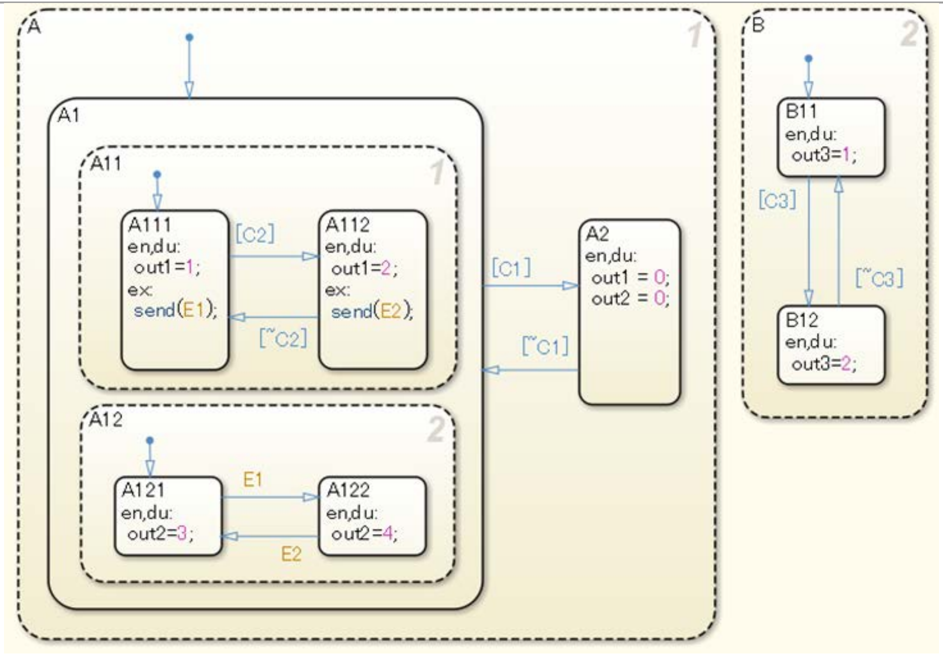
```
St01
entry, during
  data = data
  out = data;
```

Rationale	
Sub ID	**Description**
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.
abc	・Adherence to the rules prevent the definition from disappearing when copying a Stateflow block to another model.
d	・ When a Stateflow block with parent-child relationships includes local data with the same name, readability decreases due to lack of clarity with regard to the influence of the local data.

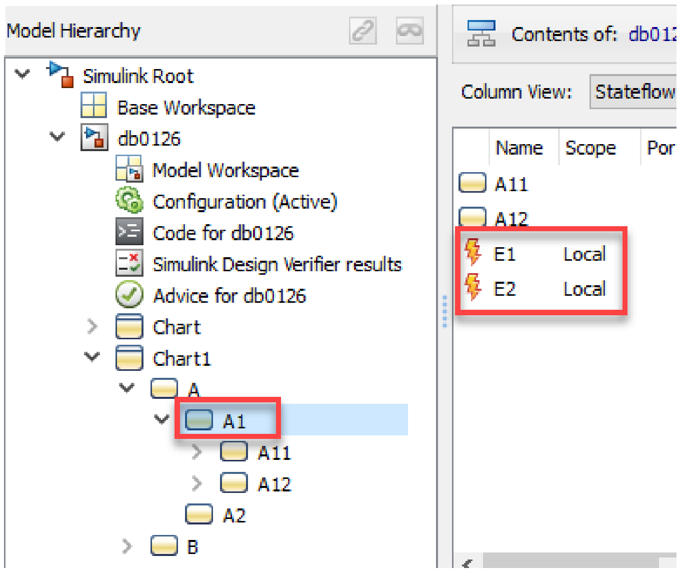
db_0126: Defining Stateflow events

Rule ID: Title	**db_0126: Defining Stateflow events**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

a	Stateflow events shall be defined by the smallest scope level in the Stateflow block being used.	-



【Correct】


Model Hierarchy

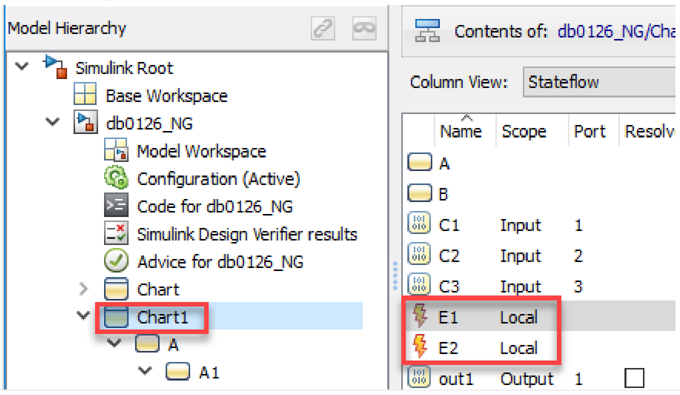
- Simulink Root
  - Base Workspace
  - db0126
    - Model Workspace
    - Configuration (Active)
    - Code for db0126
    - Simulink Design Verifier results
    - Advice for db0126
    - Chart
    - Chart1
      - A
        - A1
          - A11
          - A12
        - A2
      - B

Contents of: db012

Column View: Stateflow

Name	Scope	Por
A11		
A12		
E1	Local	
E2	Local	

【Incorrect】


Model Hierarchy

- Simulink Root
  - Base Workspace
  - db0126_NG
    - Model Workspace
    - Configuration (Active)
    - Code for db0126_NG
    - Simulink Design Verifier results
    - Advice for db0126_NG
    - Chart
    - Chart1
      - A
        - A1

Contents of: db0126_NG/Cha

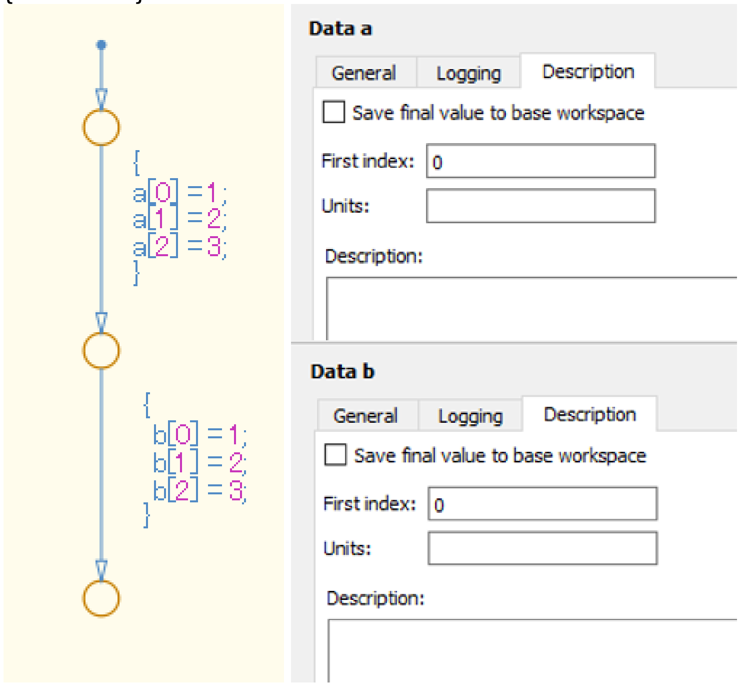
Column View: Stateflow

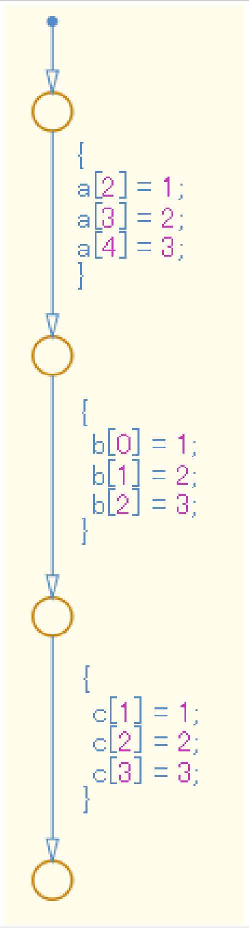
Name	Scope	Port	Resolv
A			
B			
C1	Input	1	
C2	Input	2	
C3	Input	3	
E1	Local		
E2	Local		
out1	Output	1	☐

Rationale

Sub ID	Description
a	· Limiting use locations increases reliability.

jc_0701: Usable number for first index

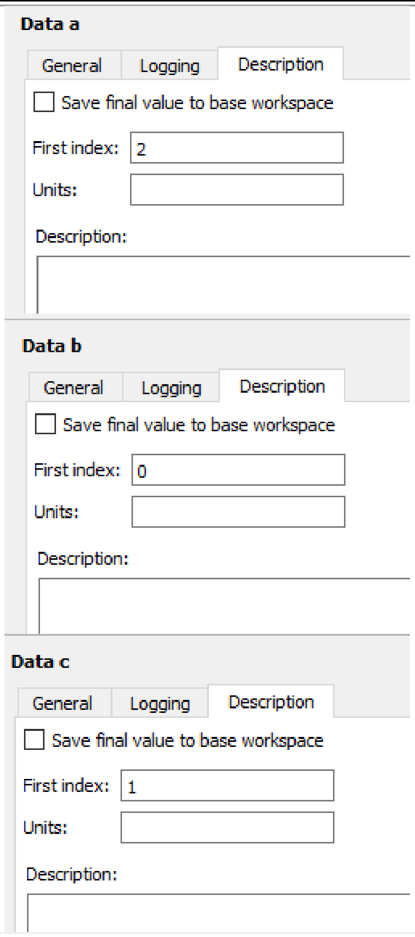
Rule ID: Title	jc_0701: Usable number for first index	
Sub ID Recommendations	NA-MAAB: a1/a2 JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	Description	Custom Parameter
a1	When [Chart] property {Action Language} is set to “C”, Stateflow data property {First index} shall be set to “0”.	-
	【Correct】 {First index} is set to “0”.  Data a: First index: 0 Data b: First index: 0 【Incorrect】 {First index} is set to a combination of “0”, “1”, and “2”.	



```
{
a[2] = 1;
a[3] = 2;
a[4] = 3;
}
```

```
{
b[0] = 1;
b[1] = 2;
b[2] = 3;
}
```

```
{
c[1] = 1;
c[2] = 2;
c[3] = 3;
}
```



Data a

General | Logging | Description

☐ Save final value to base workspace

First index: 2

Units:

Description:

Data b

General | Logging | Description

☐ Save final value to base workspace

First index: 0

Units:

Description:

Data c

General | Logging | Description

☐ Save final value to base workspace

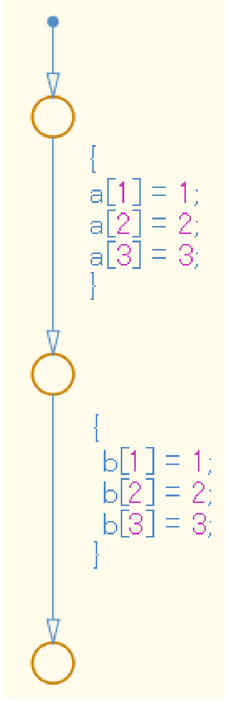
First index: 1

Units:

Description:

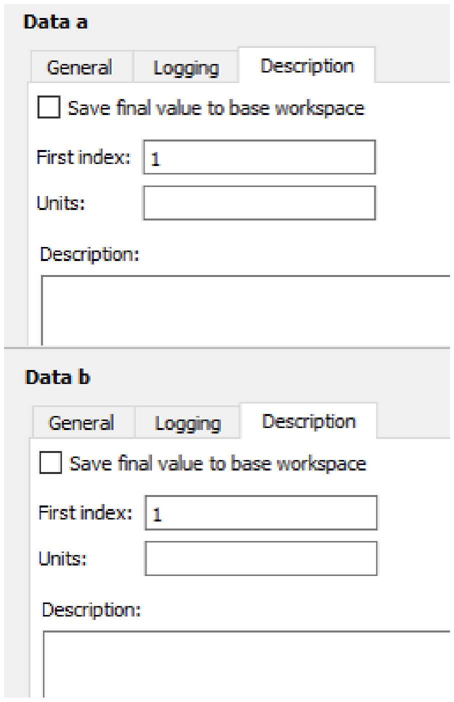
a2	When [Chart] property {Action Language} is set to “C”, Stateflow data property {First index} shall be set to “1”.	-

【Correct】
The {First index} is set to “1”.



```
{
a[1] = 1;
a[2] = 2;
a[3] = 3;
}
```

```
{
b[1] = 1;
b[2] = 2;
b[3] = 3;
}
```



Data a

General | Logging | Description

☐ Save final value to base workspace

First index: 1

Units:

Description:

Data b

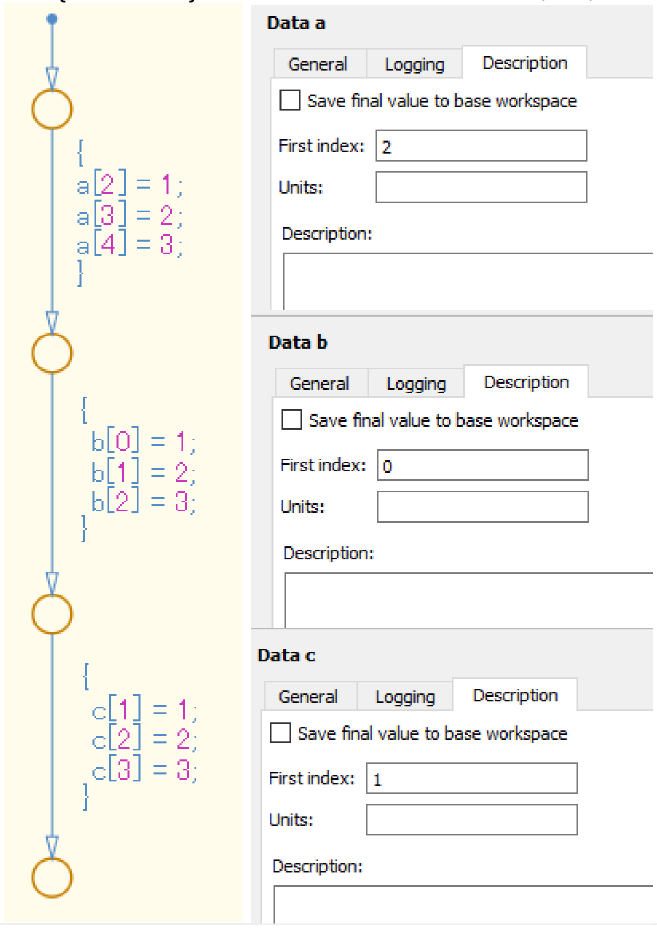
General | Logging | Description

☐ Save final value to base workspace

First index: 1

Units:

Description:

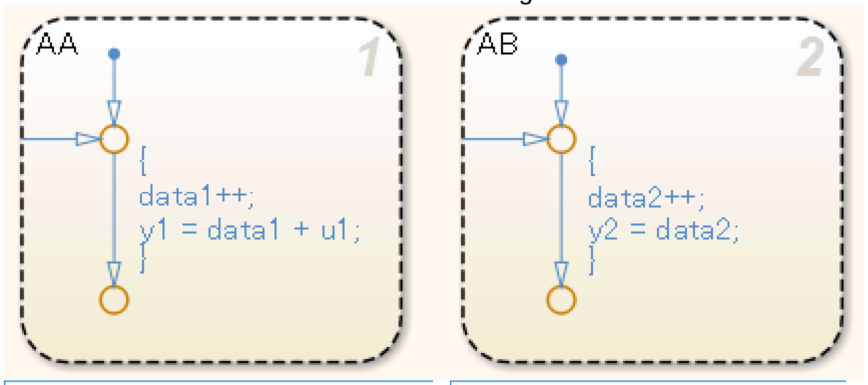
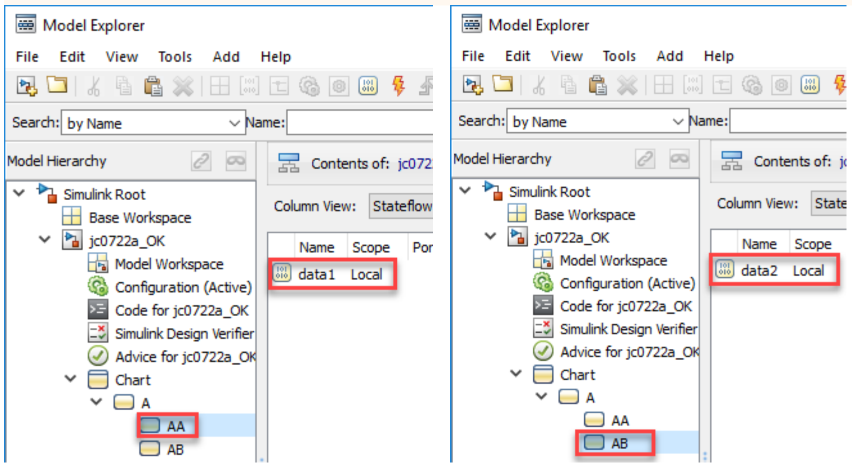
	【Incorrect】 The {First index} is set to a combination of “0”, “1”, and “2”.  { a[2] = 1; a[3] = 2; a[4] = 3; } { b[0] = 1; b[1] = 2; b[2] = 3; } { c[1] = 1; c[2] = 2; c[3] = 3; } **Data a** ☐ Save final value to base workspace First index: 2 Units: Description: **Data b** ☐ Save final value to base workspace First index: 0 Units: Description: **Data c** ☐ Save final value to base workspace First index: 1 Units: Description:
Rationale	
Sub ID	**Description**
a1	・Logic becomes easier to understand when {First index} is uniform.
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.

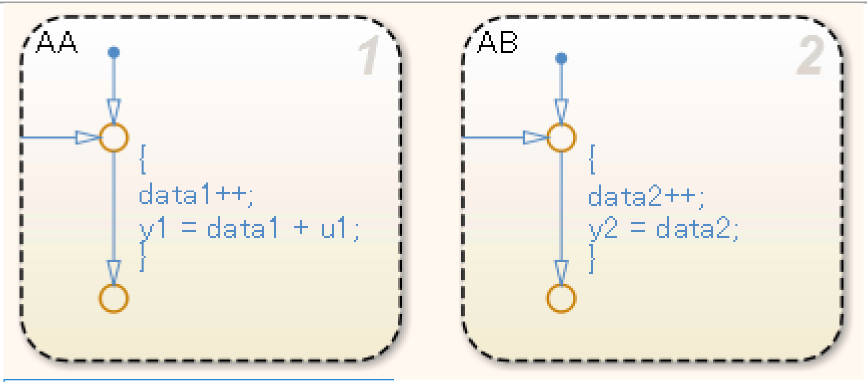
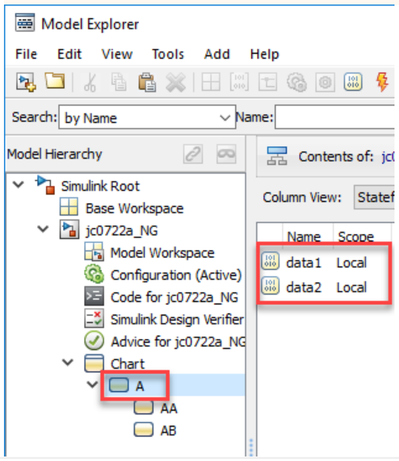
jc_0712: Execution timing for default transition path

Rule ID: Title	**jc_0712: Execution timing for default transition path**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Chart] property {Execute (enter) Chart At Initialization} shall not be selected.	-
Rationale		
Sub ID	**Description**	
a	・Using the same settings for each [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. (See manual for further details)

jc_0722: Local data definition in parallel states

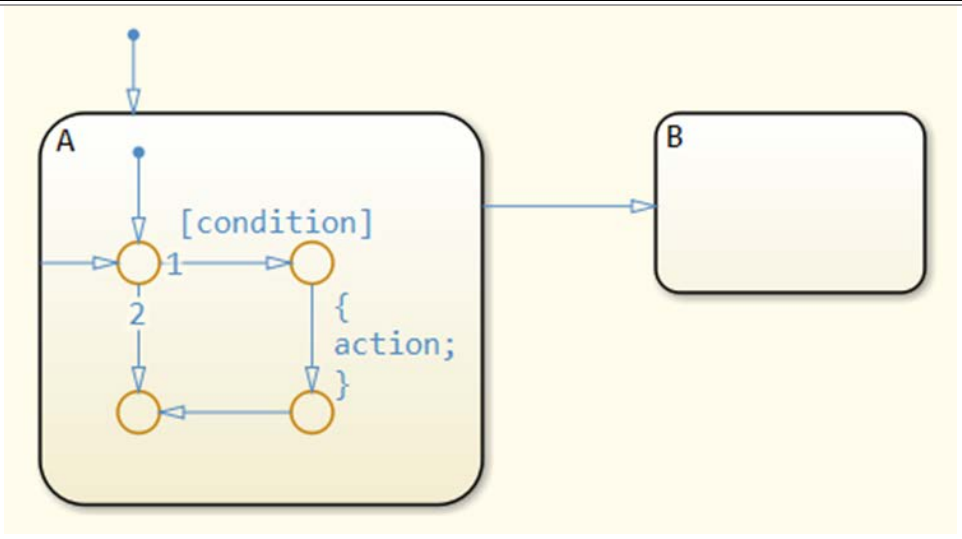
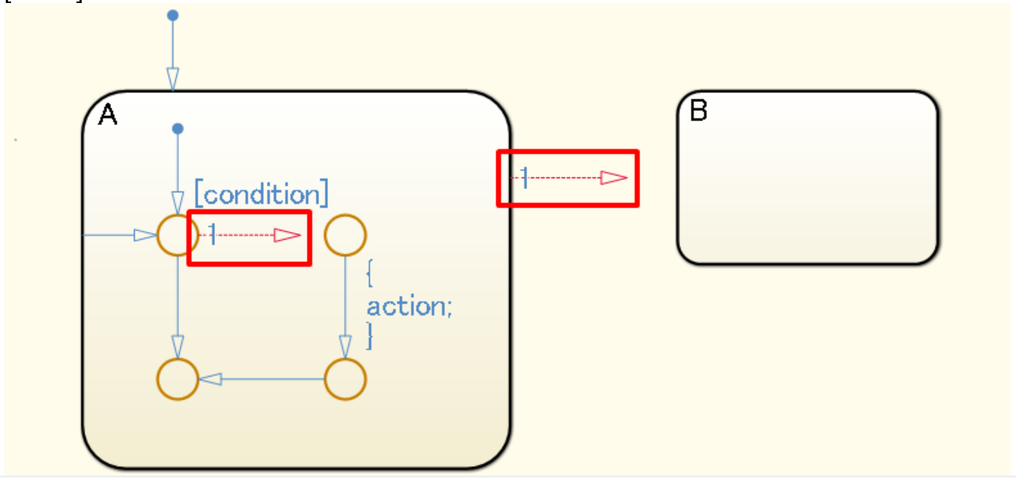
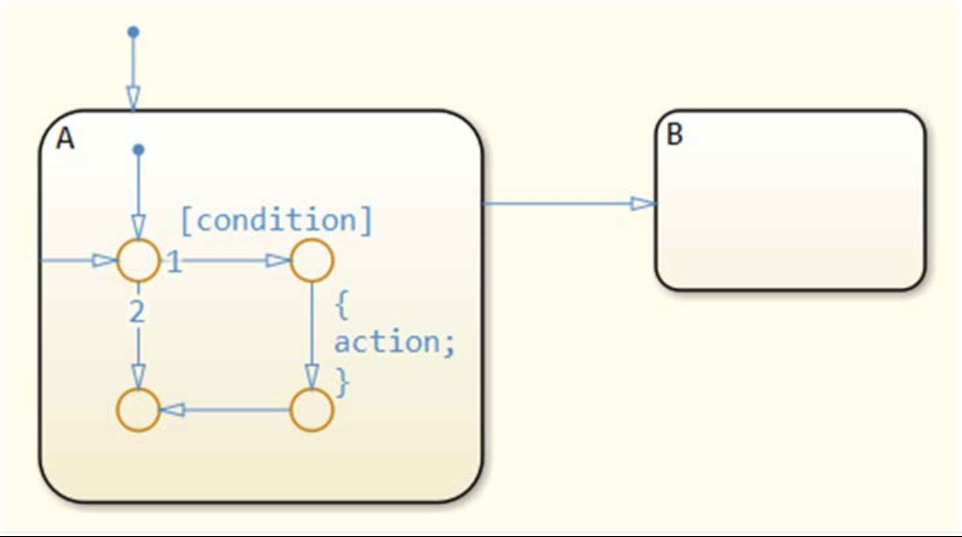
Rule ID: Title	**jc_0722: Local data definition in parallel states**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Local variables that are completed in one state shall be defined in that state.	-
	【Correct】 Local variables are defined in the state being used.  { data1++; y1 = data1 + u1; } { data2++; y2 = data2; }  Model Explorer File Edit View Tools Add Help Search: by Name Name: Model Hierarchy Contents of: jc072 Column View: Stateflow Simulink Root Base Workspace jc0722a_OK Model Workspace Configuration (Active) Code for jc0722a_OK Simulink Design Verifier Advice for jc0722a_OK Chart A AA AB Name Scope data1 Local data2 Local 【Incorrect】 Local variables are not defined in the state being used.	

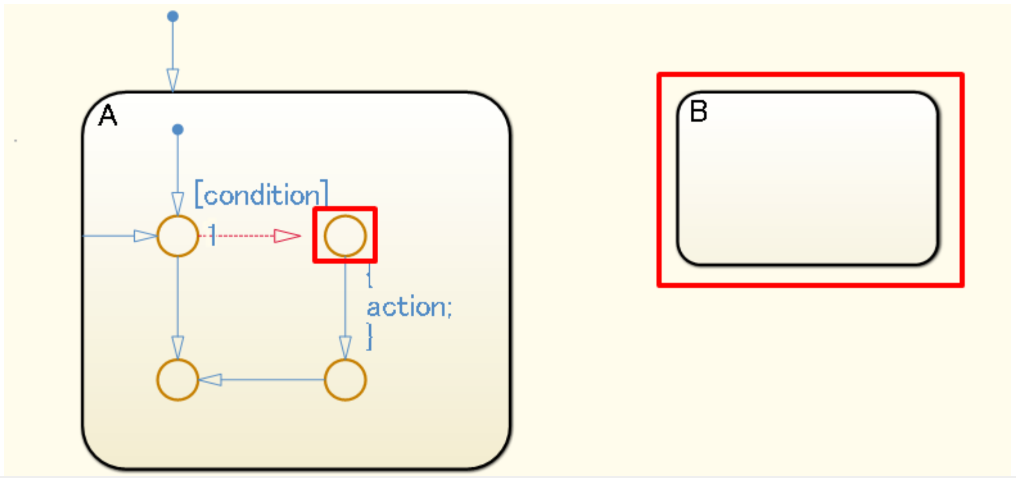
	 `{ data1++; y1 = data1 + u1; }` `{ data2++; y2 = data2; }`  Model Explorer File Edit View Tools Add Help Search: by Name Name: Model Hierarchy Contents of: jc Column View: Statef Simulink Root Base Workspace jc0722a_NG Model Workspace Configuration (Active) Code for jc0722a_NG Simulink Design Verifier Advice for jc0722a_NG Chart A AA AB Name / Scope: data1 Local; data2 Local
Rationale	
Sub ID	**Description**
a	・Readability and maintainability can be improved by explicitly limiting the valid range of the variables, thereby avoiding unintended references and changes.

4.2. Stateflow diagram

jc_0797: Unconnected transitions / states / connective junctions

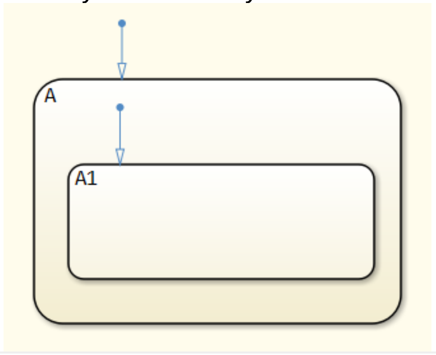
Rule ID: Title	**jc_0797: Unconnected transitions / states / connective junctions**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Chart] shall not have unconnected transitions.	-
	【Correct】 [Chart] does not have unconnected transitions.	

	 【Incorrect】 [Chart] has unconnected transitions. 	
b	[Chart] shall not have unconnected exclusive (OR) states and connective junctions without a transition source.	-
	【Correct】 [Chart] does not have unconnected exclusive (OR) states or connective junctions without a transition source. 	

	【Incorrect】 [Chart] has unconnected exclusive (OR) states and connective junctions without a transition source. 

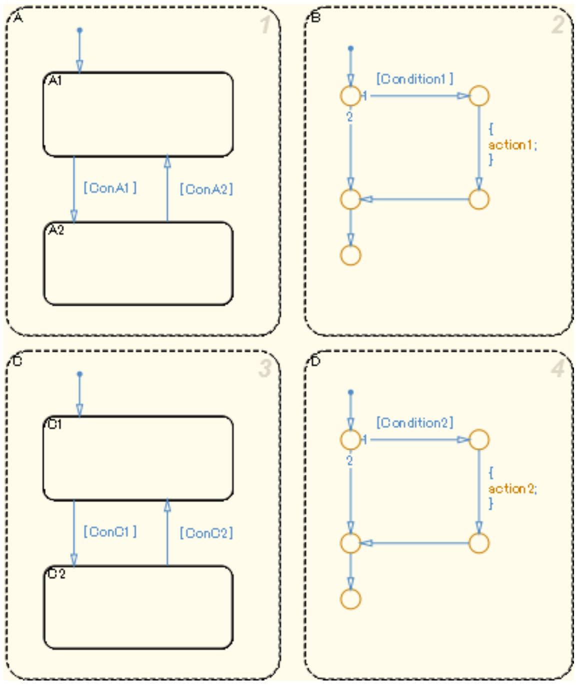
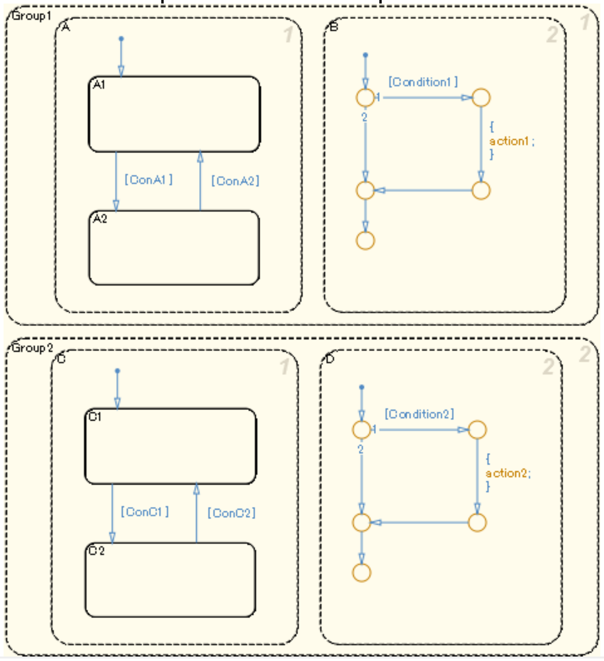
Rationale	
Sub ID	Description
ab	・Unconnected transitions can result in adverse effects, such as misinterpretation of simulation results or failure to generate code.

db_0137: States in state machines

Rule ID: Title	db_0137: States in state machines	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	Description	Custom Parameter
a	When the {Decomposition} of the [Chart] or State is set to “OR (Exclusive)”, there shall be at least two states in the hierarchy.	-
	【Incorrect】 {Decomposition} of the [Chart] and State A is set to “OR(exclusive)”, but the hierarchy contains only one state. 	

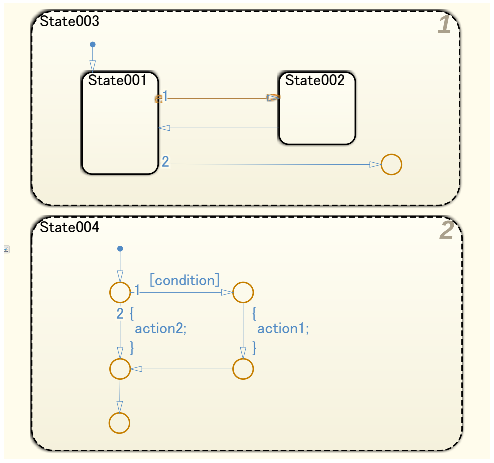
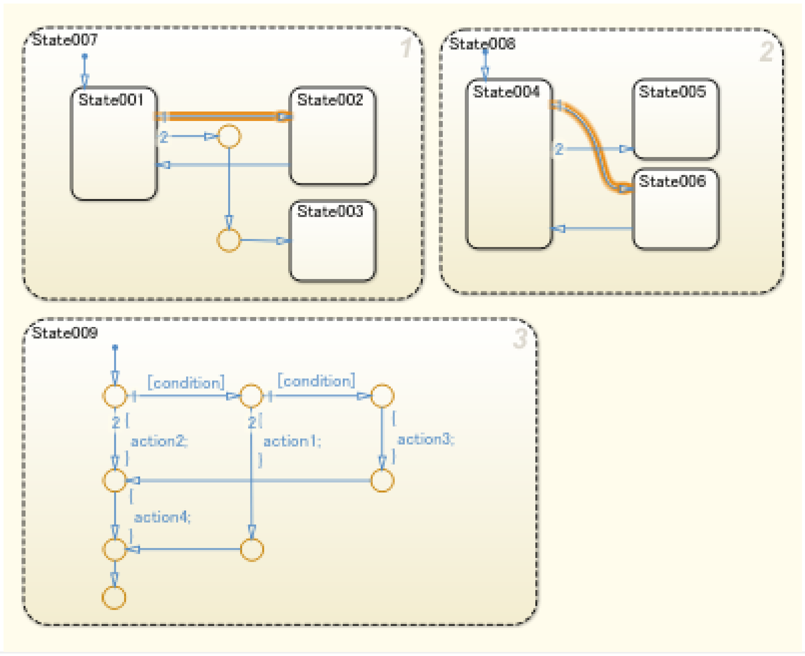
Rationale	
Sub ID	Description
a	・Redundant descriptions impair readability. ・Generated code includes unnecessary state variables.

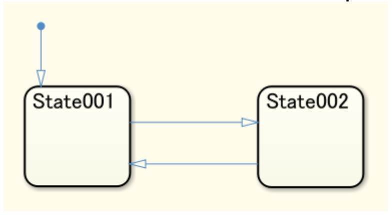
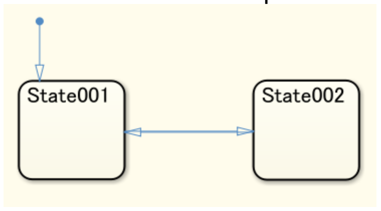
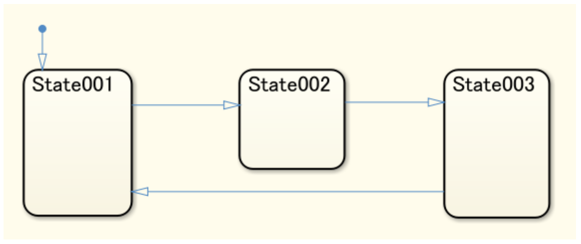
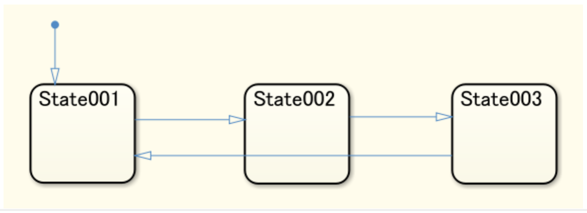
jc_0721: Usage of parallel states

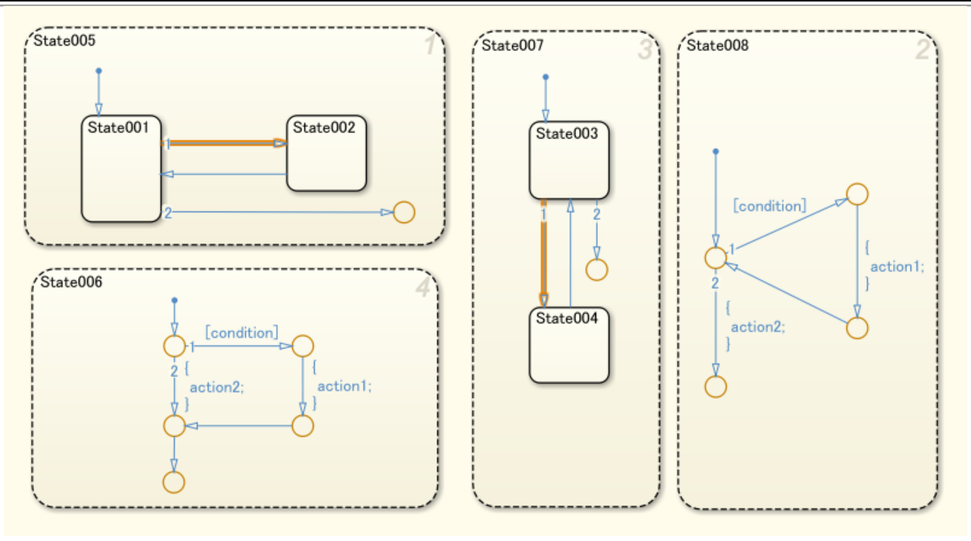
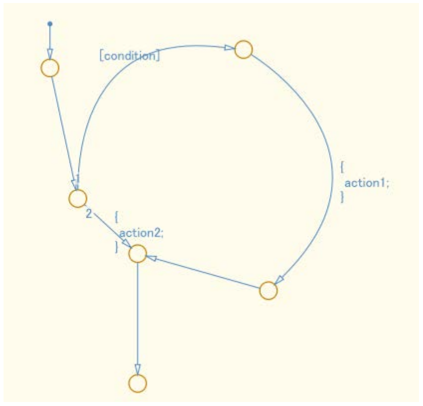
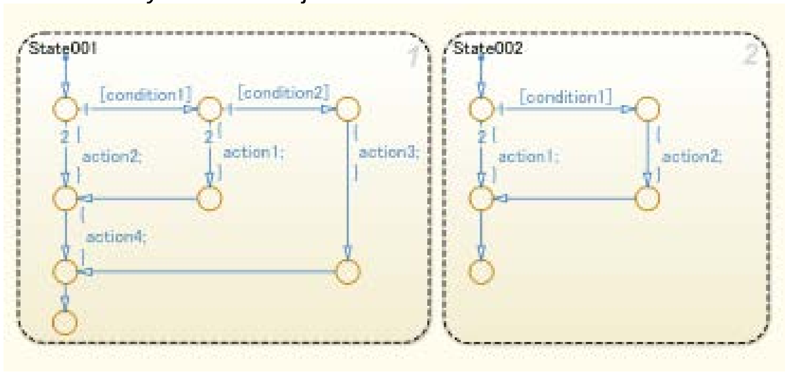
Rule ID: Title	**jc_0721: Usage of parallel states**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Substates of parallel states shall not be parallel states.	-
	【Correct】  【Incorrect】 Substates of parallel states are parallel states. 	

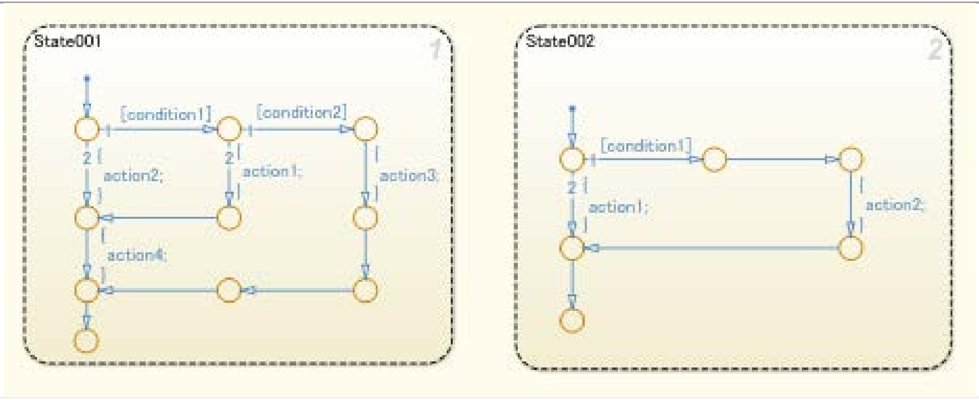
Rationale	
Sub ID	**Description**
a	・Behavior is not affected by nesting parallel states in a parent superstate. ・Hierarchization of the parallel state decreases readability.

db_0129: Stateflow transition appearance

Rule ID: Title	**db_0129: Stateflow transition appearance**	
Sub ID Recommendations	NA-MAAB: a, b, c, d, e JMAAB: a, b, c, d, e	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Transition lines shall not cross over one another.	-
	【Correct】 Transition lines do not cross.  【Incorrect】 Transition lines cross. 	

b	Transition lines shall not overlap other transition lines.	-
	【Correct】 Transition lines do not overlap other transition lines.  【Incorrect】 Transition lines overlap. 	
c	Transition lines shall not cross over states.	-
	【Correct】 Transition lines do not cross over states.  【Incorrect】 Transition lines cross over states. 	
d	Transition lines shall be drawn vertically or horizontally. Diagonal lines can be used for flow charts.	-
	【Correct】 Transition lines are drawn vertically or horizontally. Diagonal lines are used for flow charts.	

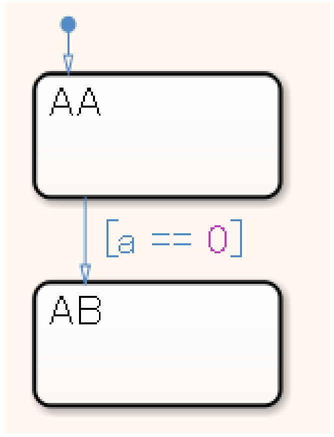
	 State005 State001 State002 State006 [condition] action1; action2; State007 State003 State004 State008 [condition] action1; action2; 【Incorrect】 Transition lines are not drawn vertically or horizontally. 	
e	Unnecessary connective junctions shall not be used.	-
	【Correct】 Unnecessary connective junctions are not used.  State001 [condition1] [condition2] action2; action1; action3; action4; State002 [condition1] action1; action2; 【Incorrect】 Unnecessary connective junctions are used.	

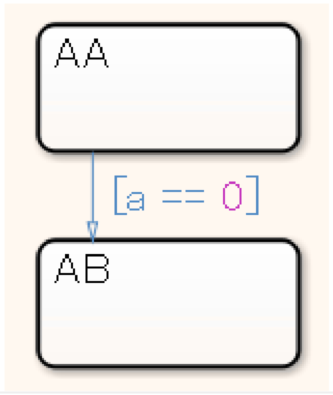
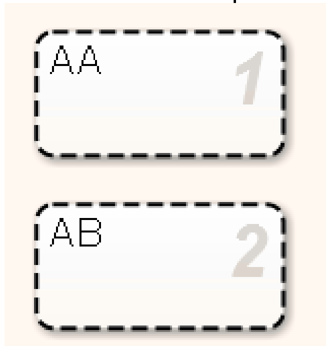
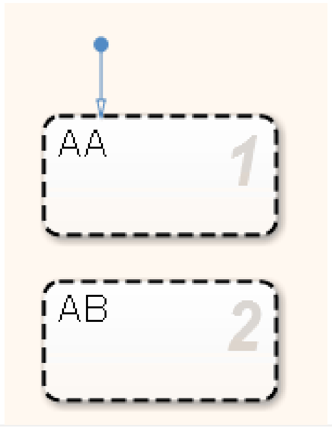
	

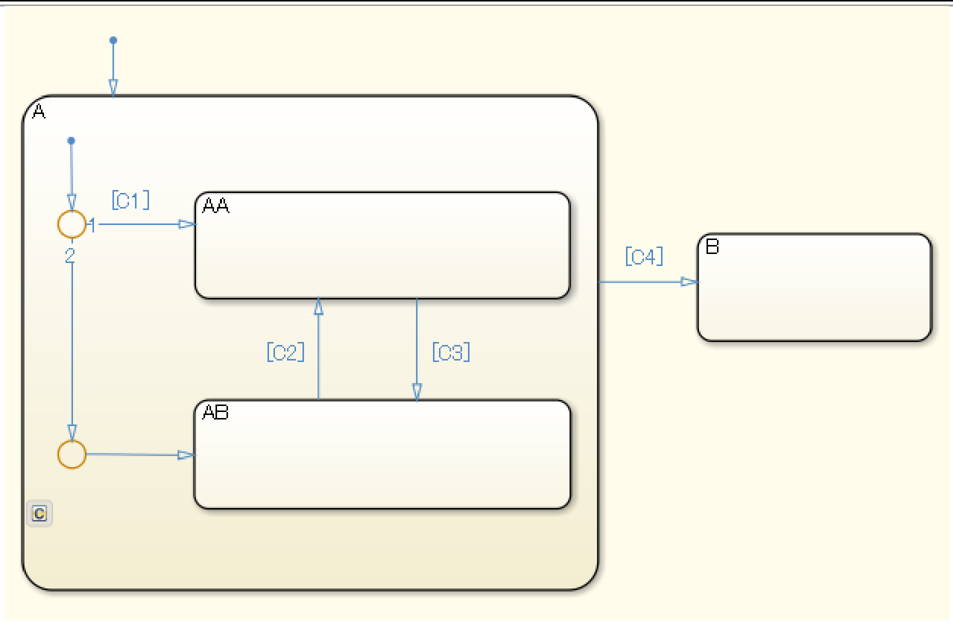
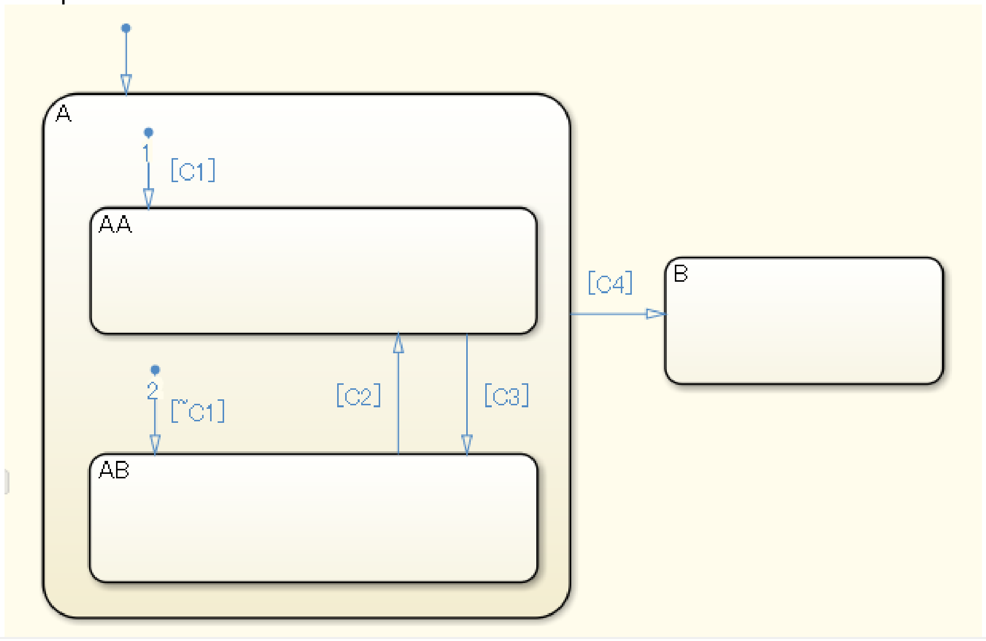
Rationale	
Sub ID	**Description**
a	・Difficult to understand the relationship between states when transition lines cross.
b	・Difficult to understand the relationship between states when transition lines overlap.
c	・Difficult to understand the relationship between states when transition lines cross over states.
d	・Consistent application of transition lines improves readability.
e	・Transitions can be difficult to understand when unnecessary connective junctions are used.

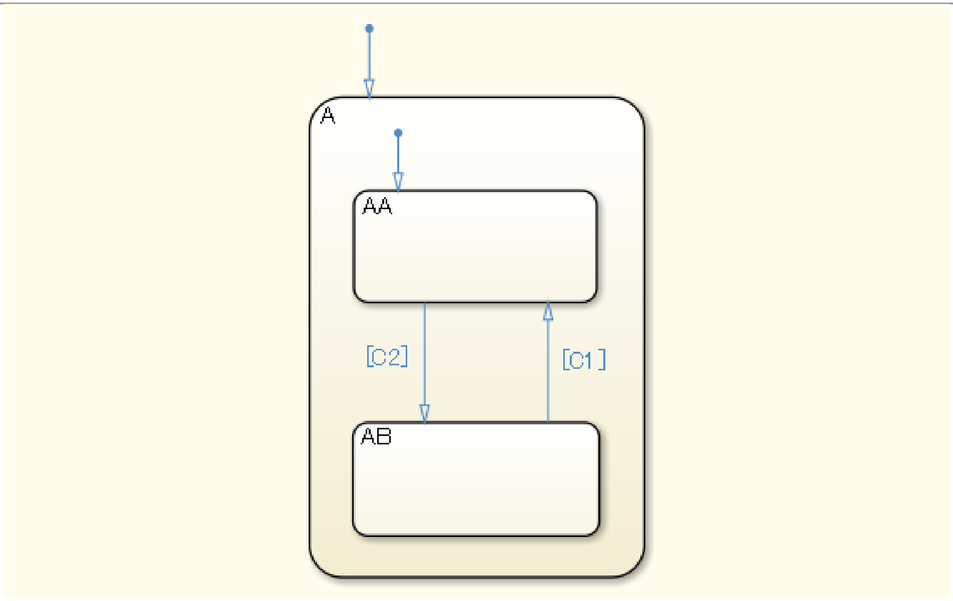
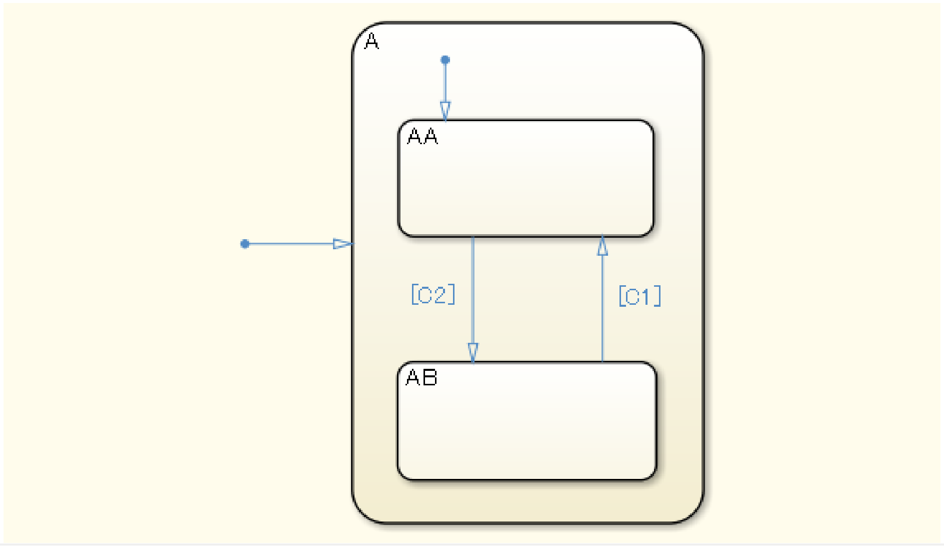
jc_0531: Default transition

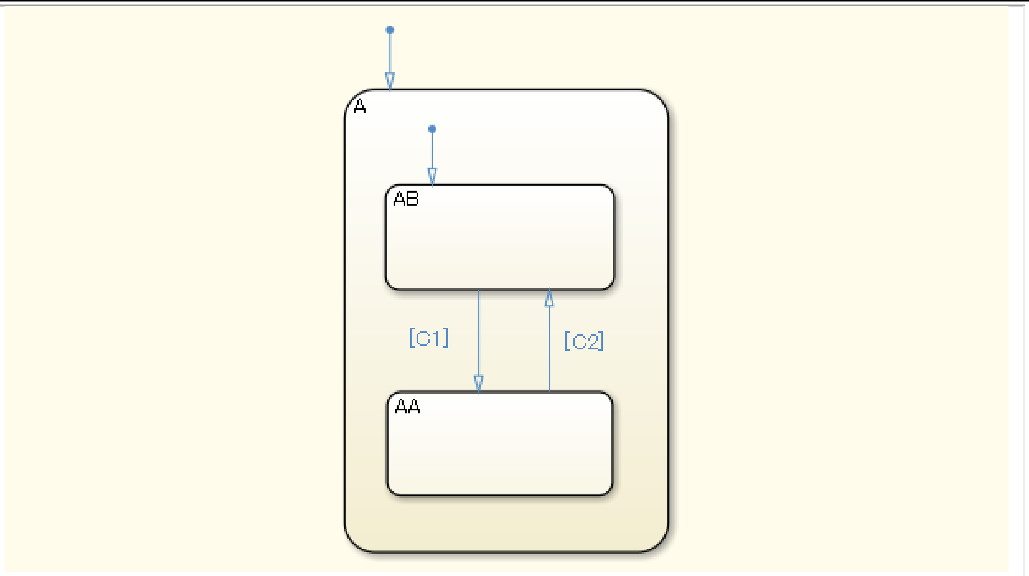
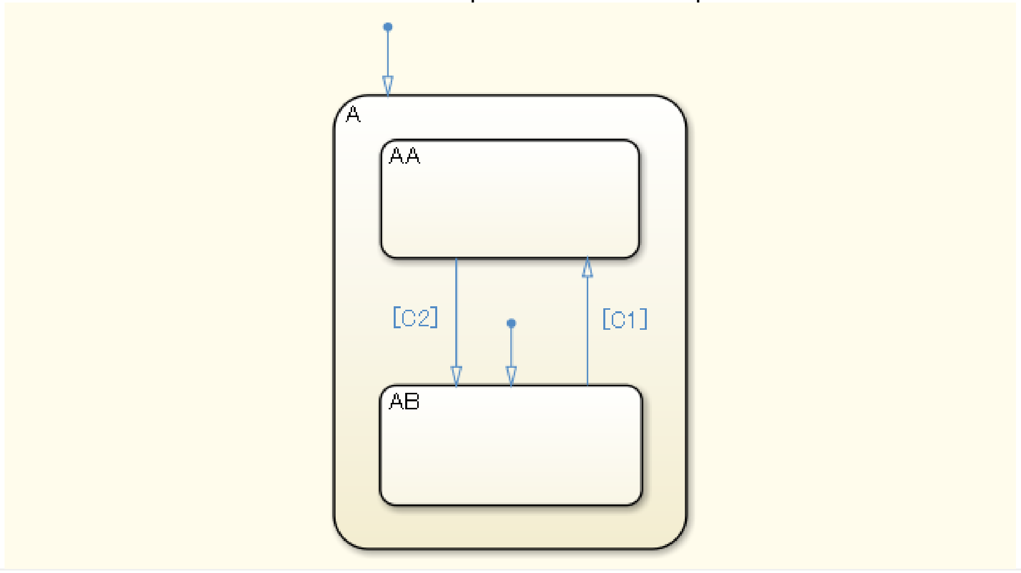
Rule ID: Title	**jc_0531: Default transition**
Sub ID Recommendations	NA-MAAB: a, b, c, d, e, f, g JMAAB: a, b, c, d, e, f, g
MATLAB® Version	All

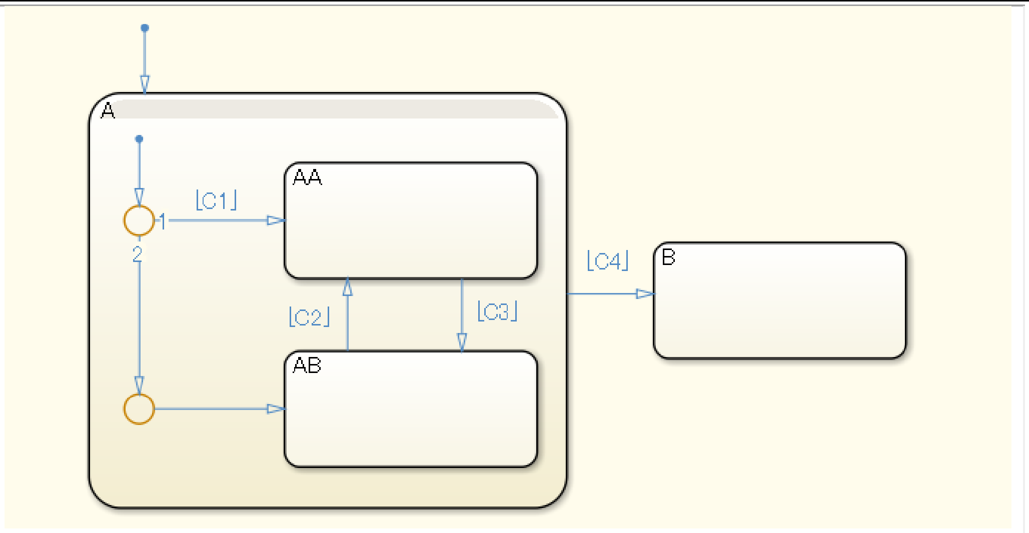
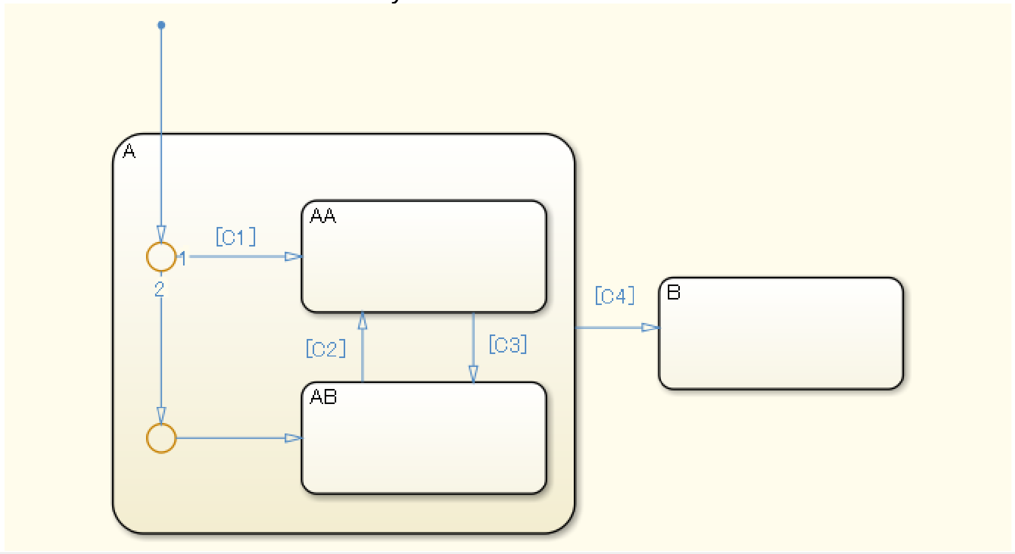
Rule		
Sub ID	**Description**	**Custom Parameter**
a	・When {Decomposition} of [Chart] is “Exclusive (OR)”, the default transition shall connect at the top of the [Chart] block. ・When {Decomposition} of the state is “Exclusive (OR)”, the default transition shall connect immediately beneath the state.	-
	【Correct】 The default transition line is connected at the top. 	

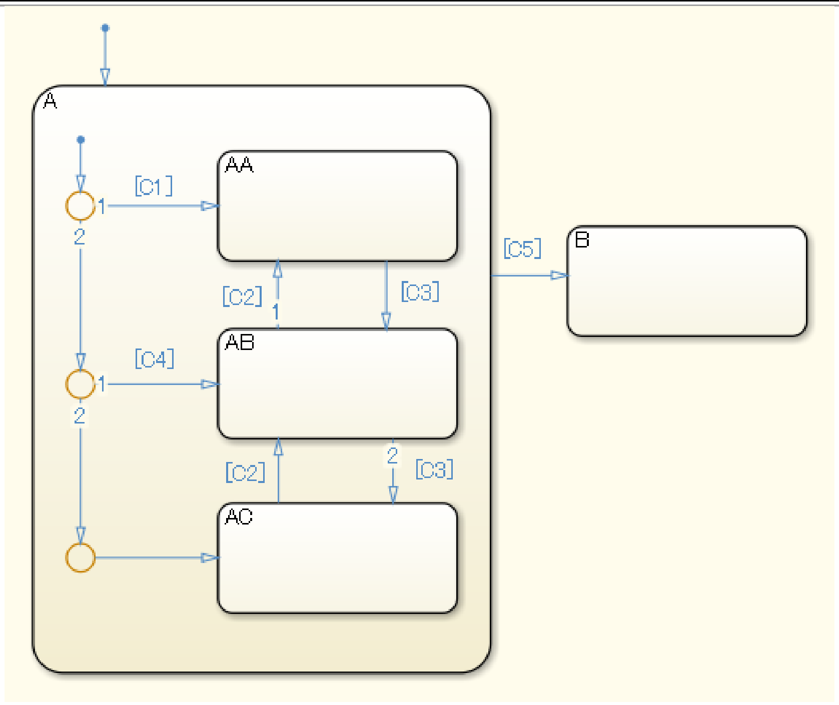
	【Incorrect】 The default transition line is not connected. 	
b	When {Decomposition} is set to “Parallel (AND)”, the default transition line shall not be connected.	-
	【Correct】 {Decomposition} of the parent object for states AA and AB is set to “Parallel (AND)”, which makes states AA and AB parallel states. The default transition line is not connected for these parallel states.  【Incorrect】 A default transition line is connected for parallel state AA. 	
c	A level shall not have multiple default transitions.	-
	【Correct】 The level does not have multiple default transitions.	

	 A [C1] 1 2 AA [C2] [C3] AB [C4] B 【Incorrect】 Multiple default transitions are included in the same level of state A.  A 1 [C1] AA 2 [~C1] [C2] [C3] AB [C4] B	
d	Default transitions shall be connected directly and positioned vertically to the upper part of the state or connective junction.	-
	【Correct】 The default transition is connected vertically to the upper part of the state.	

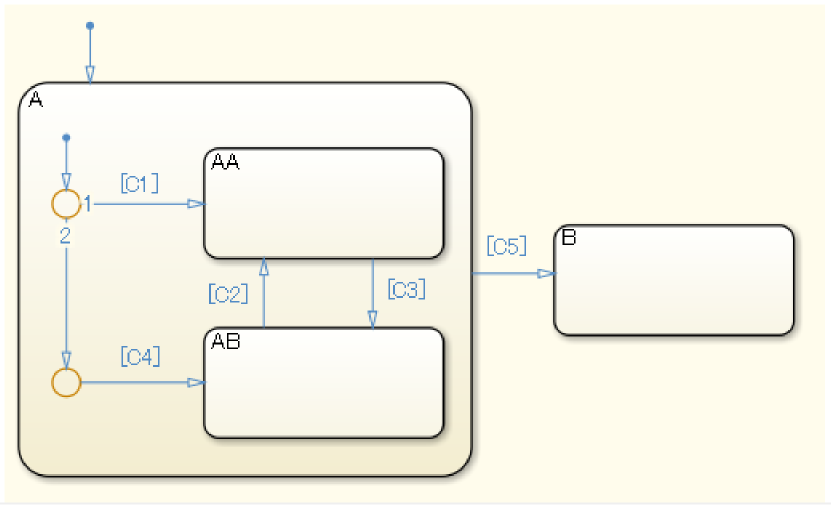
	 【Incorrect】 The default transition of state A is not connected vertically to the upper part of the state. 	
e	The destination state or destination connective junction for the default transition shall be positioned to the top left in the same level.	-
	【Correct】 The default transition is positioned to the top left in the same level.	

	 【Incorrect】 The default transition of state AB is not positioned to the top left in the same level. 	
f	Default transitions shall not extend beyond the boundaries of the state.	-
	【Correct】 The default transition is within the boundaries of the state.	

	 A [C1] 1 2 AA [C2] [C3] AB [C4] B 【Incorrect】 The default transition extends beyond the boundaries of the state.  A [C1] 1 2 AA [C2] [C3] AB [C4] B	
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.	-
	【Correct】 The path with the lowest priority in the transition path for the default transition is an unconditional transition.	



【Incorrect】
 The path with the lowest priority in the transition path for the default transition is not an unconditional transition.

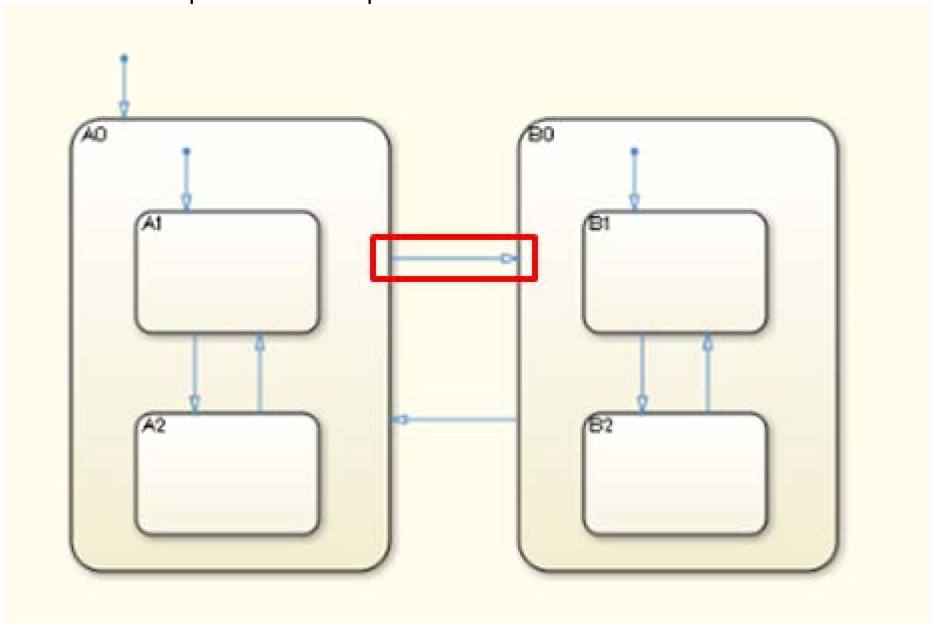


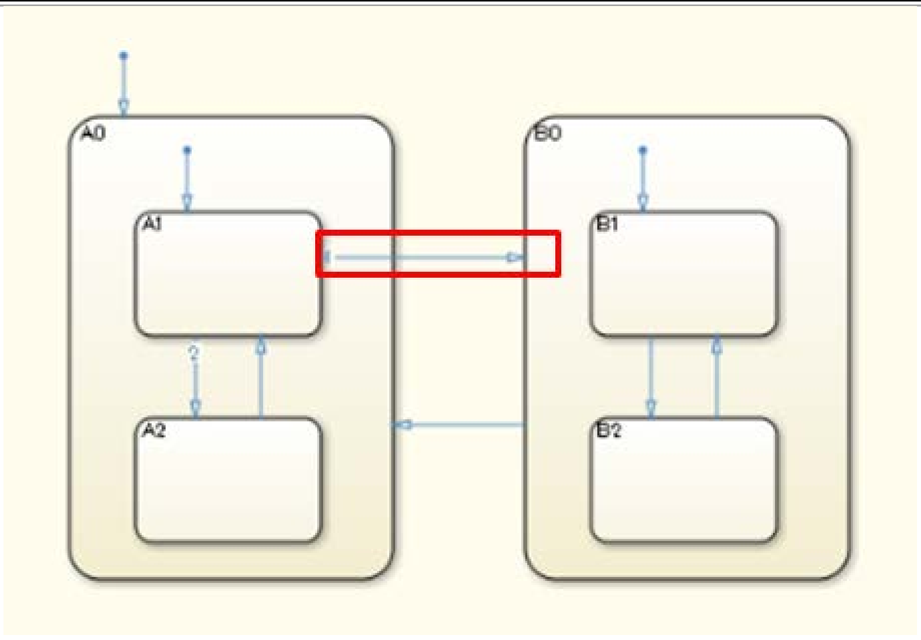
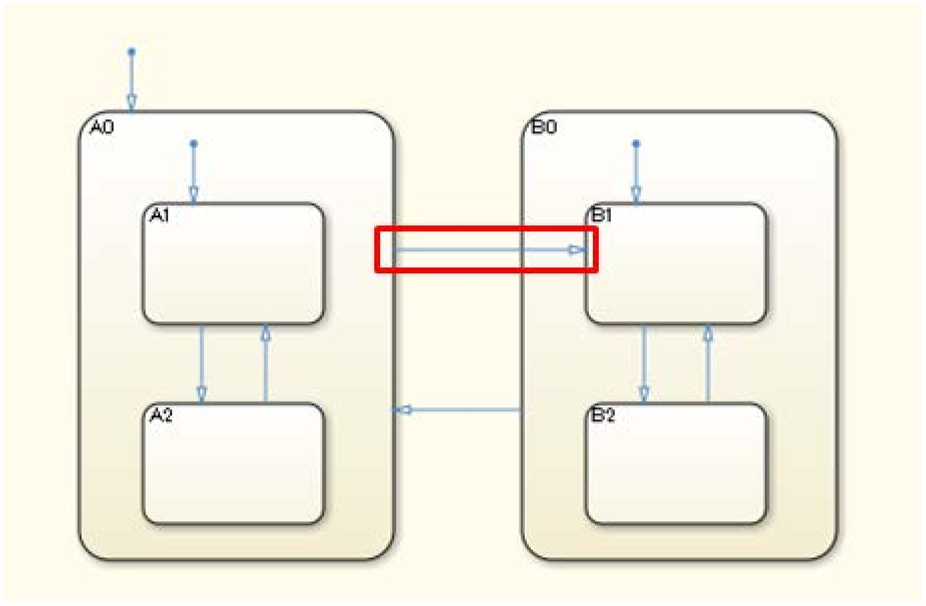
Rationale

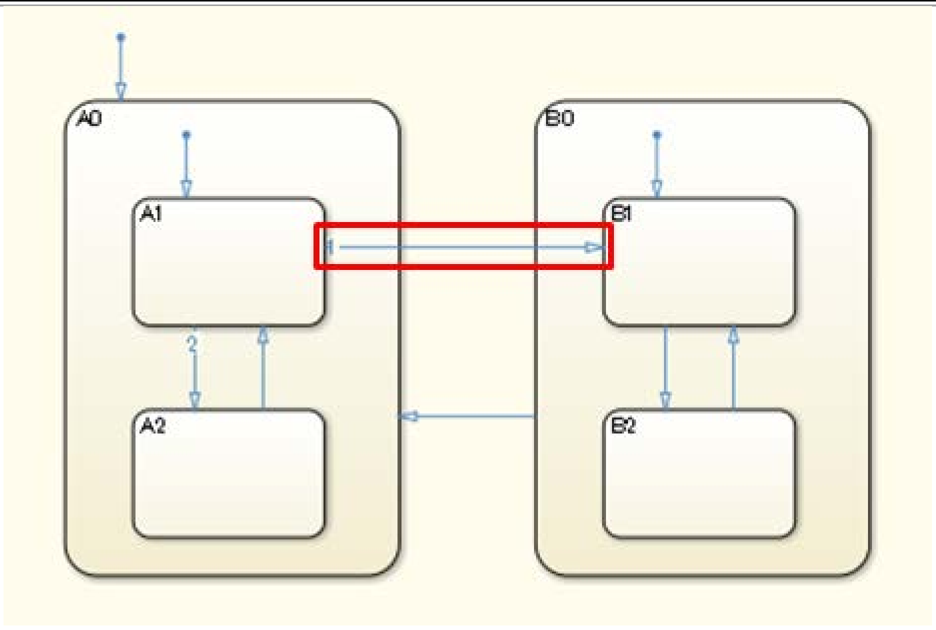
Sub ID	**Description**
a	・Simulation errors can occur when a state chart does not include default transition lines. ・When default transitions are included in a flow chart, it is impossible to determine whether this is intentional or through failure to insert them.
b	・Readability improves when there are no unnecessary default transitions.
c	・The state may not function as intended and produce a warning when multiple default transitions are included in the same level.
d	・Readability decreases when there are curves or variations in the angle or position of default transitions.
e	・Readability decreases when there are variations in the position of the transition destination state or transition destination connective junction for the default transition.

f	・Readability decreases when a default transition extends beyond the boundary of a state and intersects with state boundaries and expressions.
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.

jc_0723: Prohibited direct transition from external state to child state

Rule ID: Title	**jc_0723: Prohibited direct transition from external state to child state**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Transitions from one state directly to an external child state shall be prohibited.	-
	【Correct】 Transition from parent state to parent state.  Transition from child state to another parent state.	

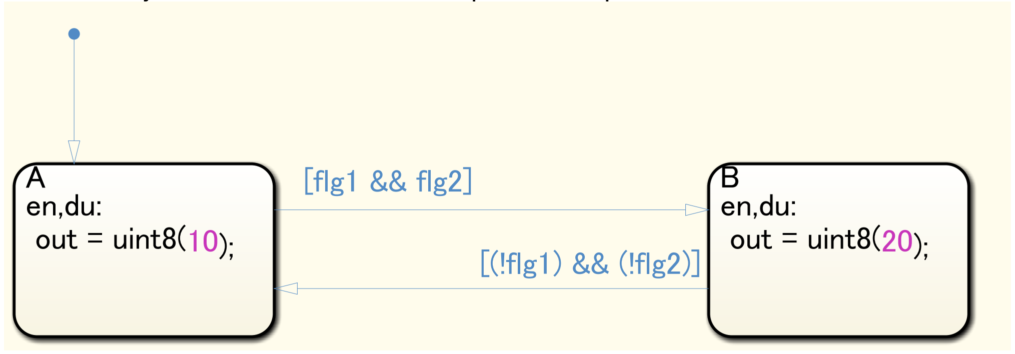
	 【Incorrect】 Direct transition from an external state to a child state in a different state.  Direct transition from an external child state to a child state in a different state.

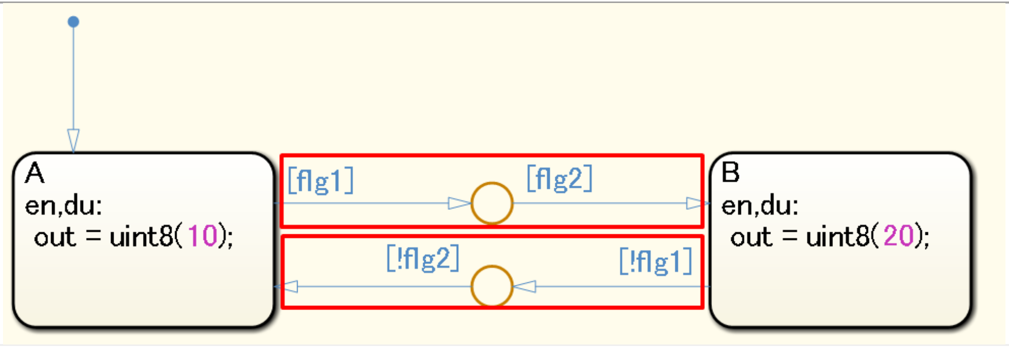
	

Rationale	
Sub ID	**Description**
a	・Direct transitions between child states can complicate the states and decrease readability.

jc_0751: Backtracking prevention in state transition

Rule ID: Title	**jc_0751:** Backtracking prevention in state transition	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Connective junctions shall not be used to separate complex conditions.	-

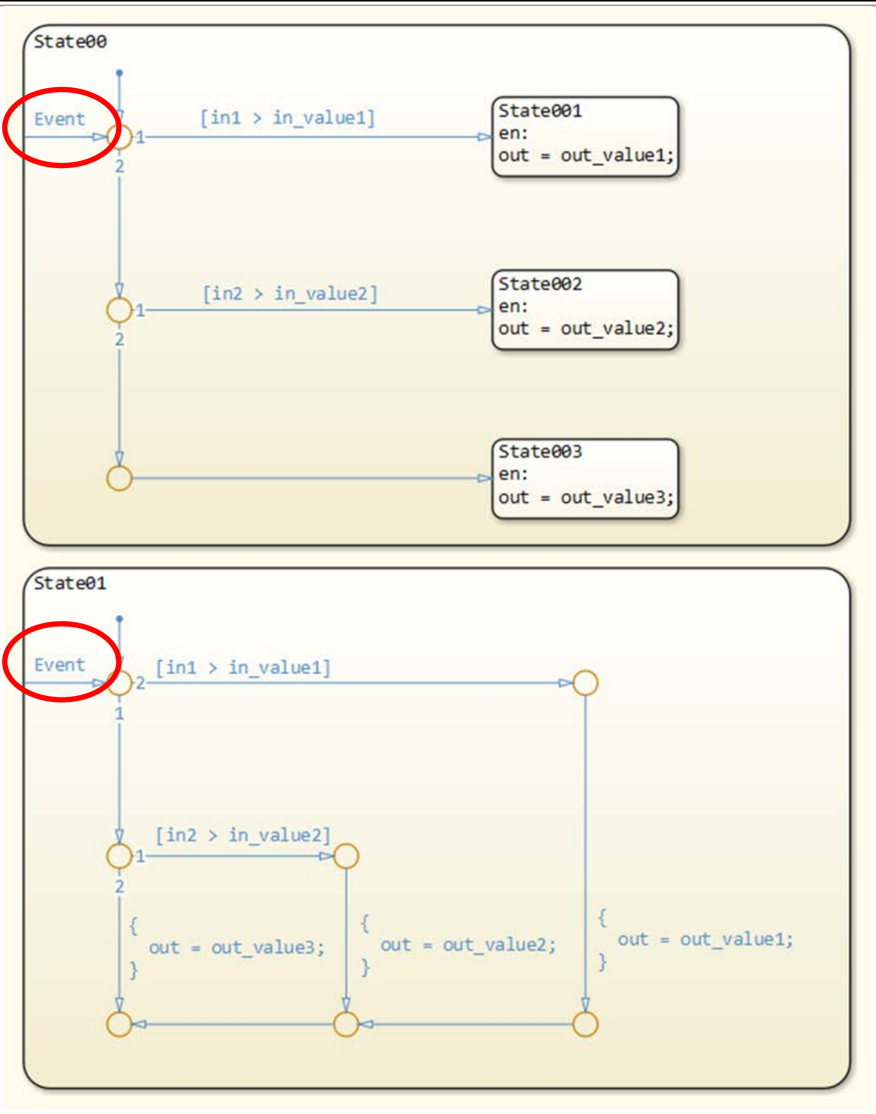
	【Correct】
	Connective junctions are not used to separate complex conditions.
	 A
	en,du:
	out = uint8(10);
	[flg1 && flg2]
	[(!flg1) && (!flg2)]
	B
	en,du:
	out = uint8(20);
	【Incorrect】
	Connective junctions are used to separate complex conditions.	

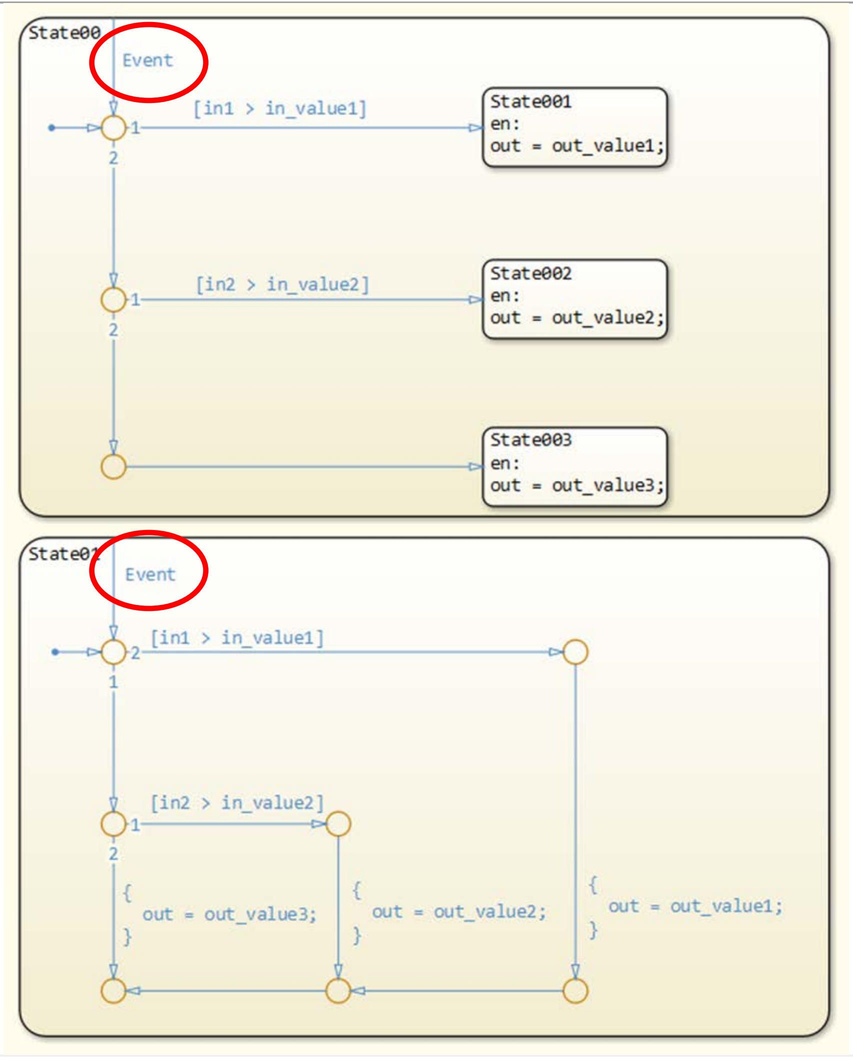
	

Rationale	
Sub ID	**Description**
a	・Deviation from the rule can cause backtracking, which results in unintended behavior.

jc_0760: Starting point of internal transition

Rule ID: Title	**jc_0760: Starting point of internal transition**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Internal transition lines shall start from the left edge of the state.	-
	【Correct】 The internal transition line begins at the left edge of the state.	

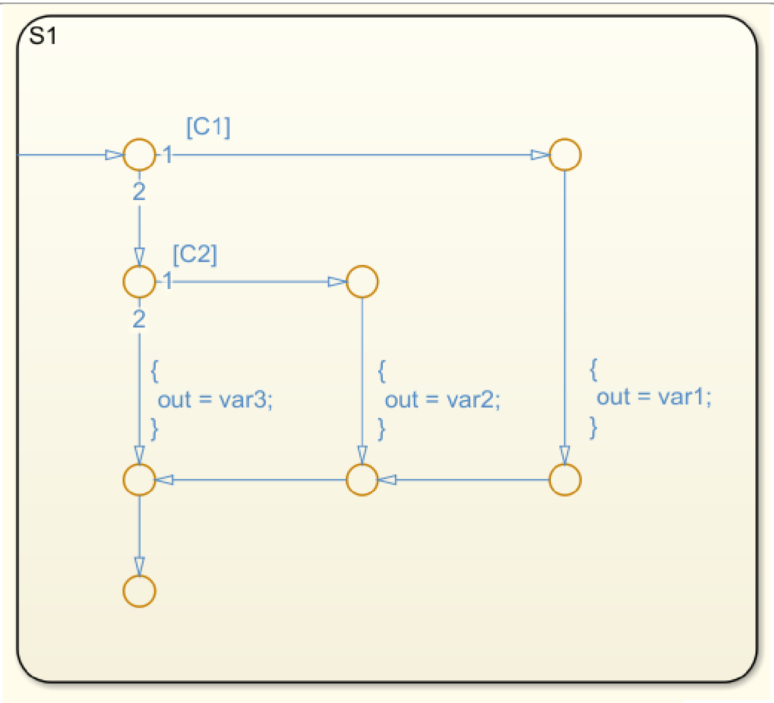
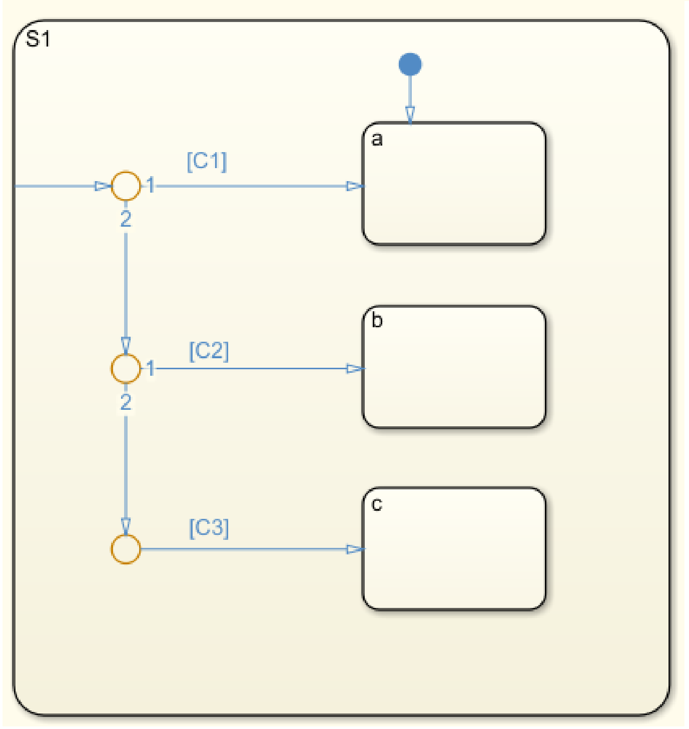
	 State00
	Event
	[in1 > in_value1]
	1
	2
	State001
	en:
	out = out_value1;
	[in2 > in_value2]
	1
	2
	State002
	en:
	out = out_value2;
	State003
	en:
	out = out_value3;
	State01
	Event
	[in1 > in_value1]
	2
	1
	[in2 > in_value2]
	1
	2
	{ out = out_value3; }
	{ out = out_value2; }
	{ out = out_value1; }
	【Incorrect】
	The internal transition line does not begin at the left edge of the state.

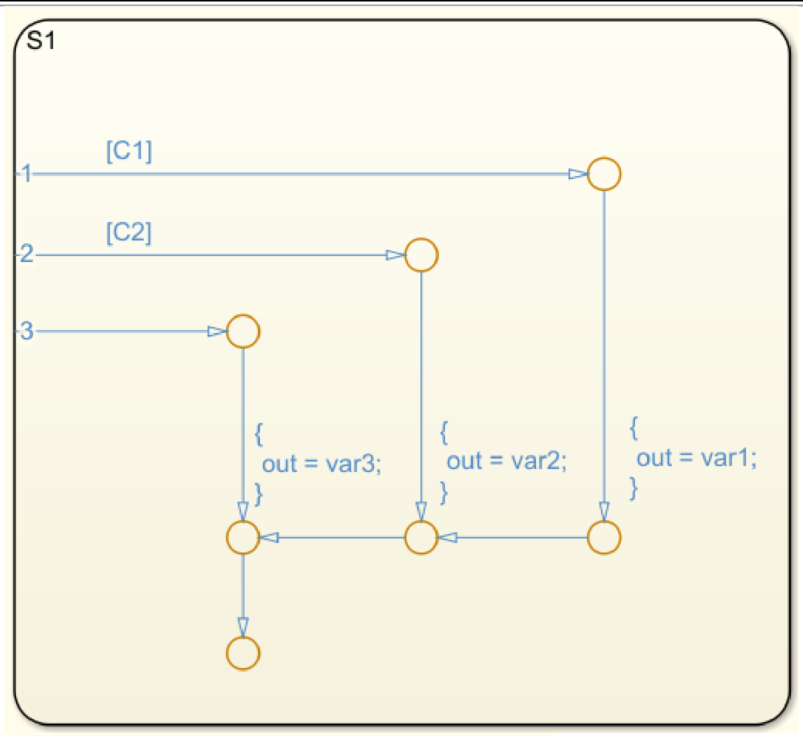
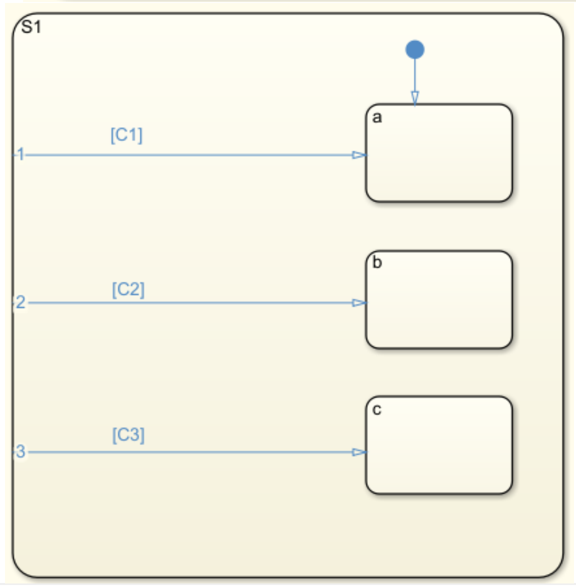
	 State00
	Event
	[in1 > in_value1]
	1
	2
	State001
	en:
	out = out_value1;
	[in2 > in_value2]
	1
	2
	State002
	en:
	out = out_value2;
	State003
	en:
	out = out_value3;
	State01
	Event
	[in1 > in_value1]
	2
	1
	[in2 > in_value2]
	1
	2
	{ out = out_value3; }
	{ out = out_value2; }
	{ out = out_value1; }

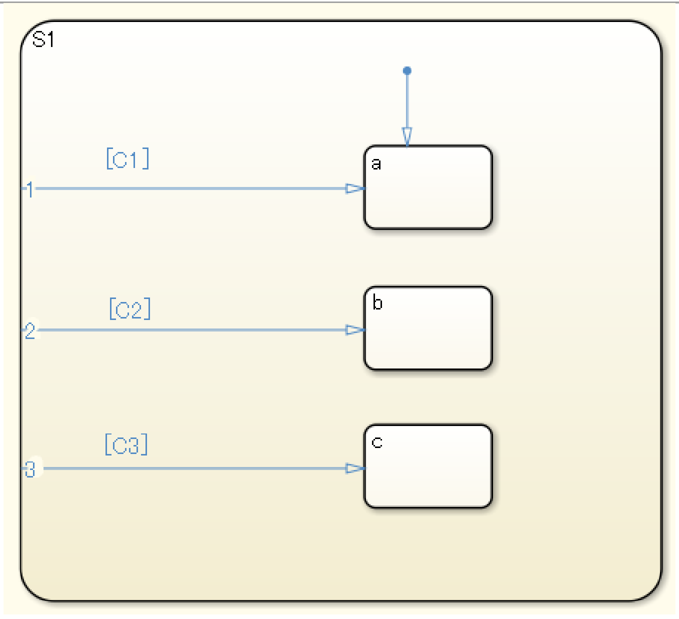
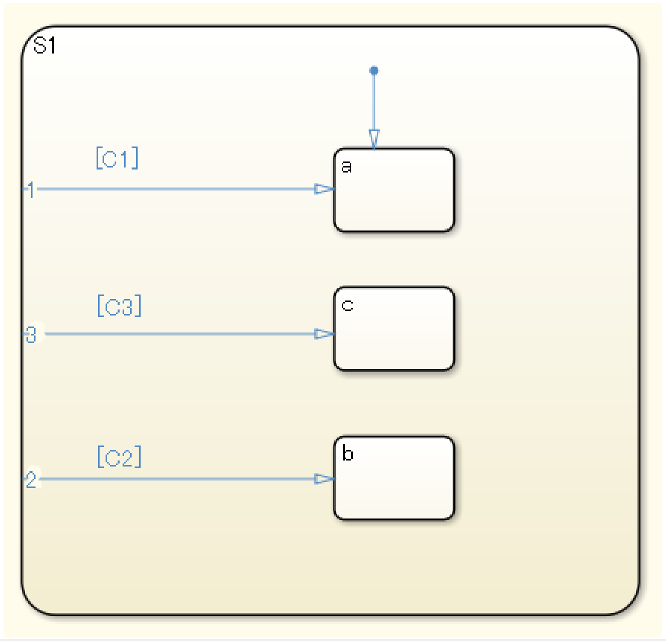
Rationale	
Sub ID	**Description**
a	・Adherence to the rule improves readability.

jc_0763: Usage of multiple internal transitions

Rule ID: Title	**jc_0763: Usage of multiple internal transitions**	
Sub ID Recommendations	NA-MAAB: a1/a2 JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	Multiple internal transitions shall not be used in a single state.	-
	【Correct】	

	 S1
	[C1]
	1
	2
	[C2]
	1
	2
	{
	out = var3;
	}
	{
	out = var2;
	}
	{
	out = var1;
	} S1
	a
	b
	c
	[C1]
	1
	2
	[C2]
	1
	2
	[C3]
	【Incorrect】

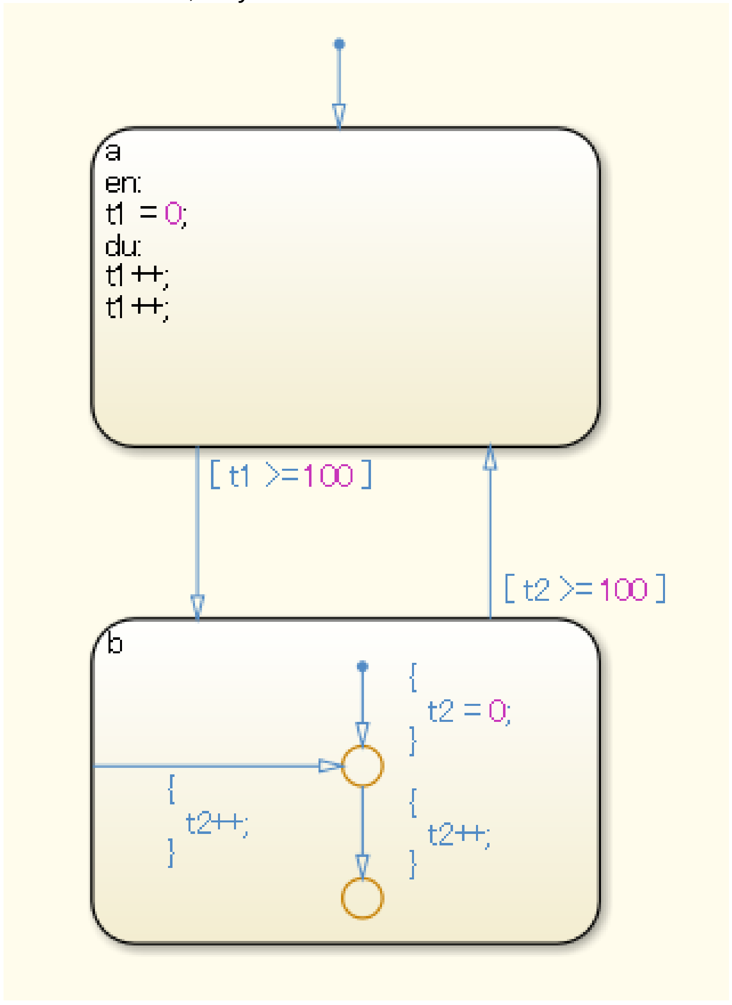
	 S1 [C1] 1 [C2] 2 3 { out = var3; } { out = var2; } { out = var1; } 	
a2	When multiple internal transitions are used in a single state, they shall be listed from top to bottom in the order of execution.	-
	【Correct】	

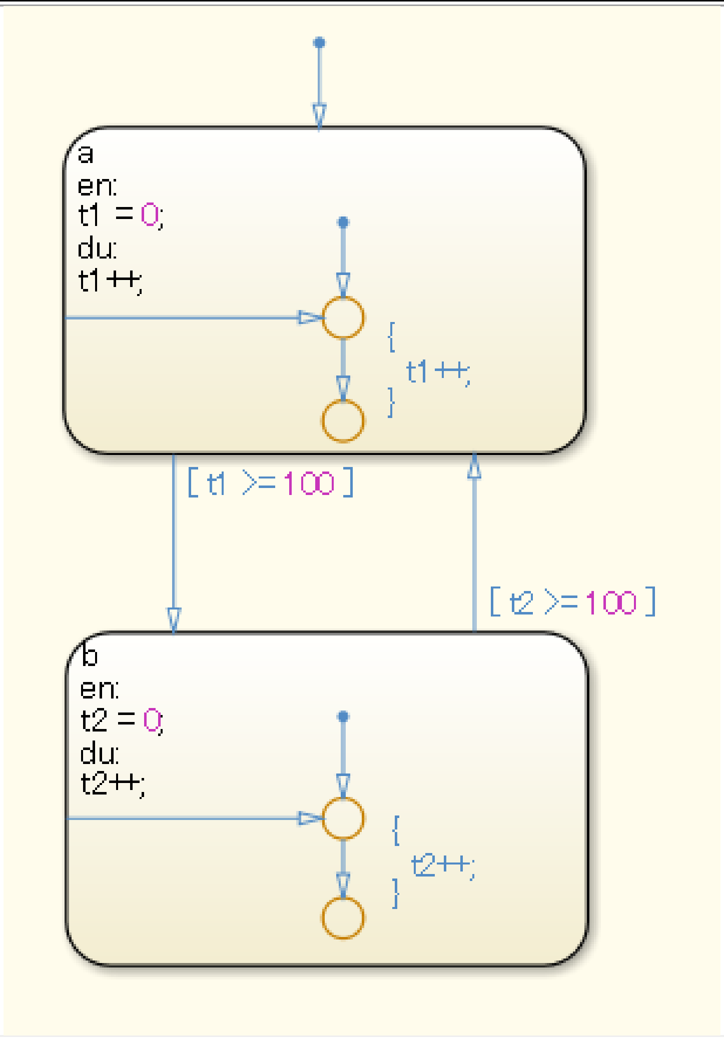
	 【Incorrect】 
Rationale	
Sub ID	**Description**
a1	・The number of transition conditions is unclear when multiple internal transitions are used. By limiting the use of internal transitions to a single use, transitions are clearer and readability improves.
a2	・Using multiple internal transitions can prevent transition lines from crossing and simplifies state transitions. ・Arranging internal transitions in execution order improves readability.

jc_0762: Prohibition of state action and flow chart combination

Rule ID: Title	**jc_0762: Prohibition of state action and flow chart combination**

Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	A state shall not include state actions (entry, during, or exit) and flow charts.	-

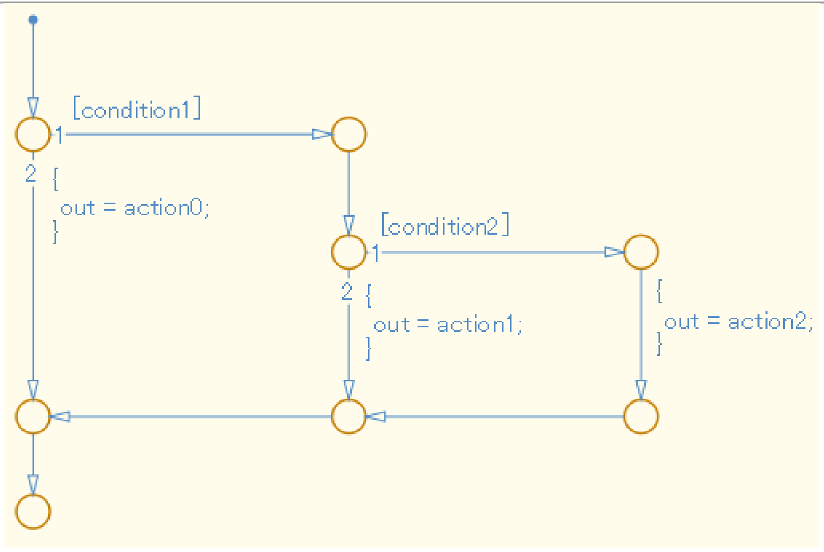
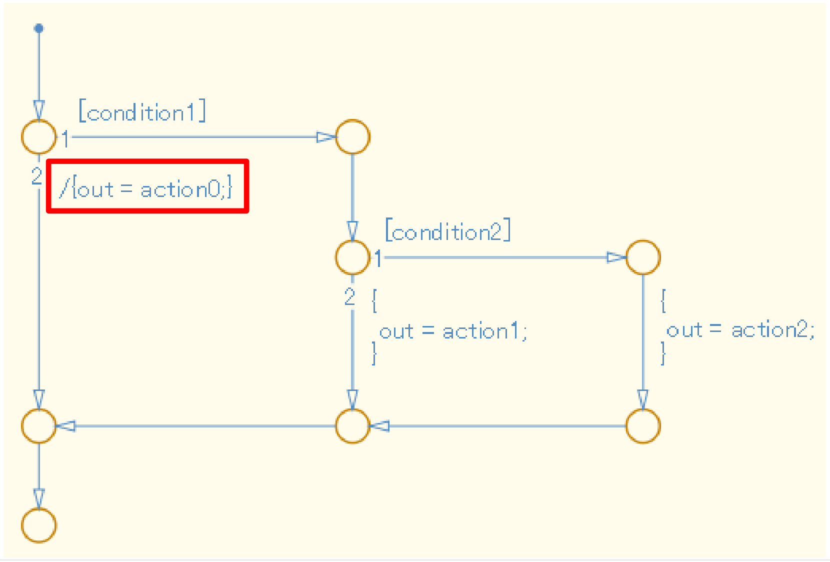
	【Correct】
	Within the state, only one of either a State action or a flow chart is described.
	 a
	en:
	t1 = 0;
	du:
	t1 ++;
	t1 ++;
	[t1 >=100]
	[t2 >=100]
	b
	{
	t2 = 0;
	}
	{
	t2++;
	}
	{
	t2++;
	}
	【Incorrect】
	The state has both a state action and a flow chart.	

	 a
	en:
	t1 = 0;
	du:
	t1++;
	{
	t1++;
	}
	[t1 >= 100]
	[t2 >= 100]
	b
	en:
	t2 = 0;
	du:
	t2++;
	{
	t2++;
	}

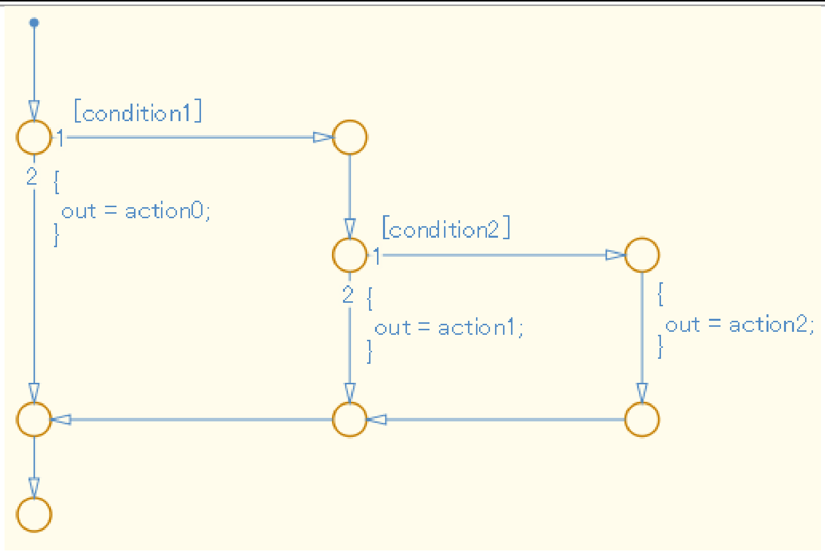
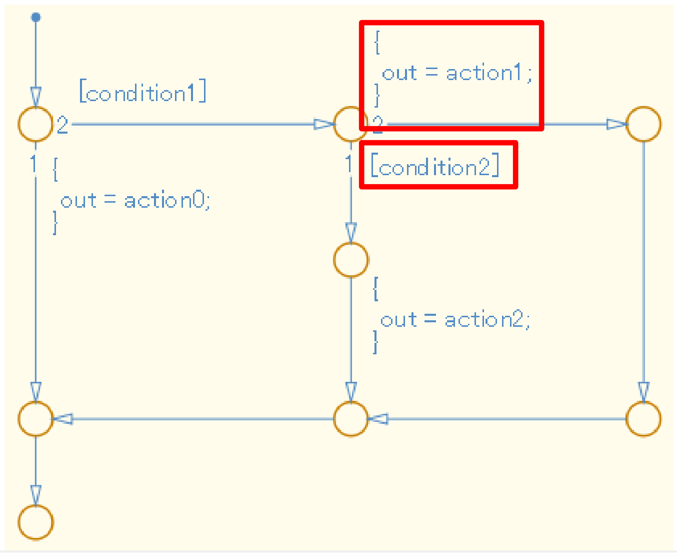
Rationale	
Sub ID	**Description**
a	· The execution order becomes difficult to understand, which decreases readability.

db_0132: Transitions in flow charts

Rule ID: Title	**db_0132: Transitions in flow charts**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Transition actions shall not be used in flow charts.	-
	【Correct】 Transition actions are not used.	

	 [condition1]
	1
	2
	{
	out = action0;
	}
	[condition2]
	1
	2
	{
	out = action1;
	}
	{
	out = action2;
	}
	【Incorrect】
	Transition actions are used.
	 [condition1]
	1
	2
	/{out = action0;}
	[condition2]
	1
	2
	{
	out = action1;
	}
	{
	out = action2;
	}	

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.	-
	【Correct】 The condition is positioned on a horizontal transition line and the condition action is on a vertical transition line.	

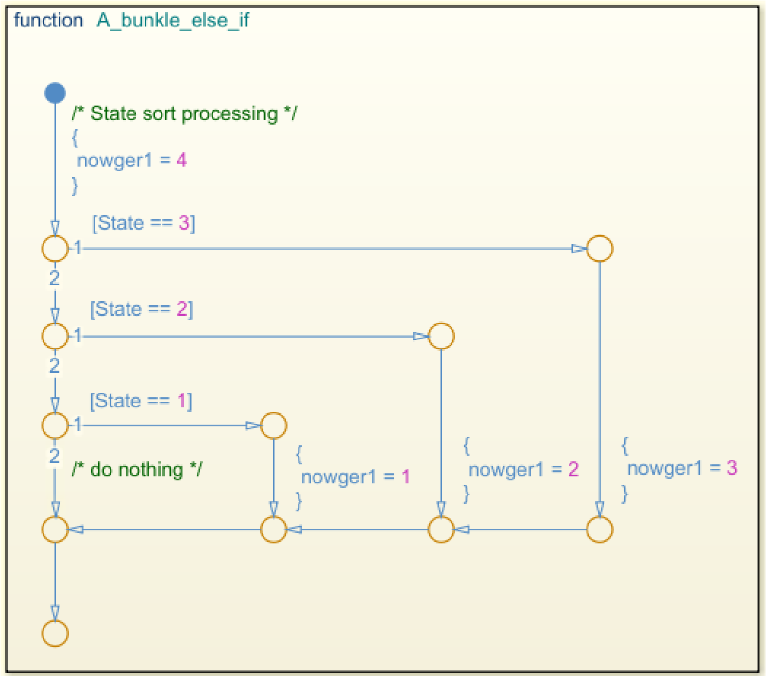
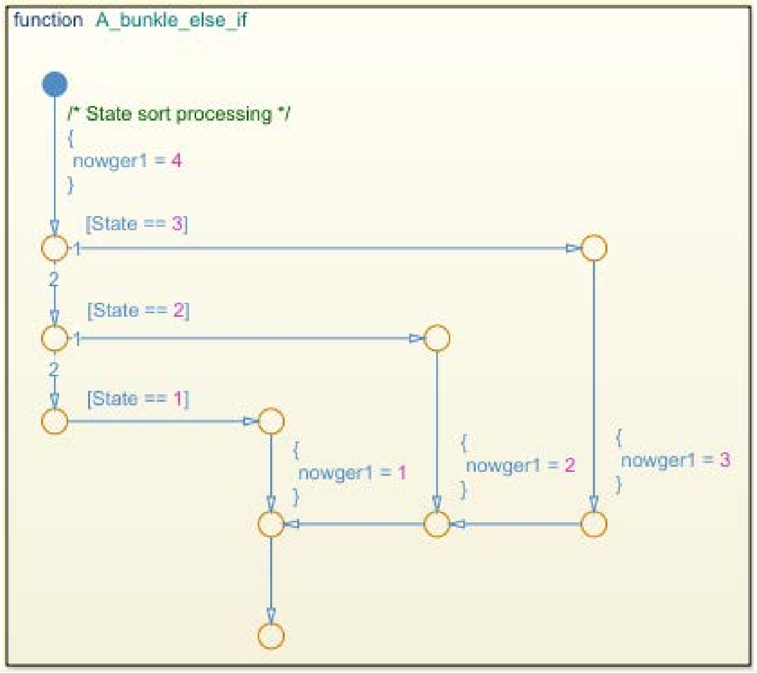
	 [condition1]
	1
	2
	{
	out = action0;
	}
	[condition2]
	1
	2
	{
	out = action1;
	}
	{
	out = action2;
	}
	【Incorrect】
	The condition is positioned on a vertical transition line and the condition action is on a horizontal transition line.
	 [condition1]
	2
	1
	{
	out = action0;
	}
	{
	out = action1;
	}
	2
	1
	[condition2]
	{
	out = action2;
	}

Rationale	
Sub ID	**Description**
a	・The transition action in a flow chart is not executed.
b	・Consistent positioning of conditions and condition actions improves readability.

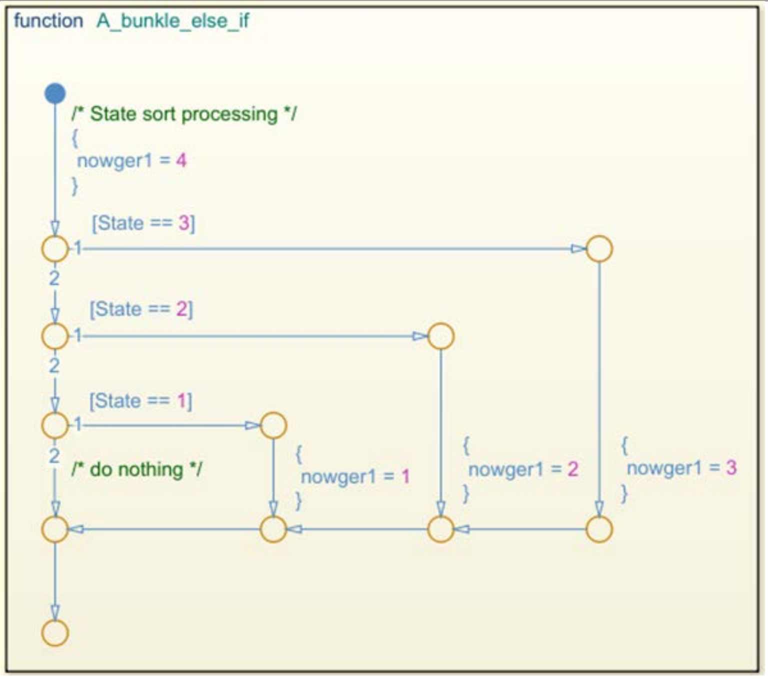
jc_0773: Unconditional transition of a flow chart

Rule ID: Title	**jc_0773: Unconditional transition of a flow chart**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

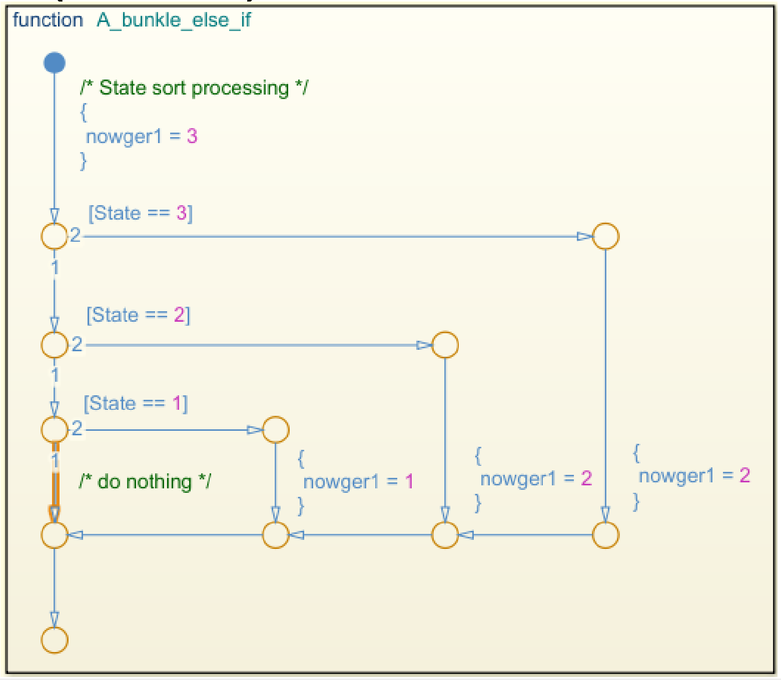
a	When a transition line with a transition condition originates from a connective junction, t unconditional transition line shall also begin from that junction.	-

	【Correct】
	 function A_bunkle_else_if
	/* State sort processing */
	{
	nowger1 = 4
	}
	[State == 3]
	[State == 2]
	[State == 1]
	/* do nothing */
	{
	nowger1 = 1
	}
	{
	nowger1 = 2
	}
	{
	nowger1 = 3
	}
	1
	2
	【Incorrect】
	0762
	 function A_bunkle_else_if
	/* State sort processing */
	{
	nowger1 = 4
	}
	[State == 3]
	[State == 2]
	[State == 1]
	{
	nowger1 = 1
	}
	{
	nowger1 = 2
	}
	{
	nowger1 = 3
	}
	1
	2	

b	The {execution order} for unconditional transitions shall be set to the last value.	-
	【Correct】	



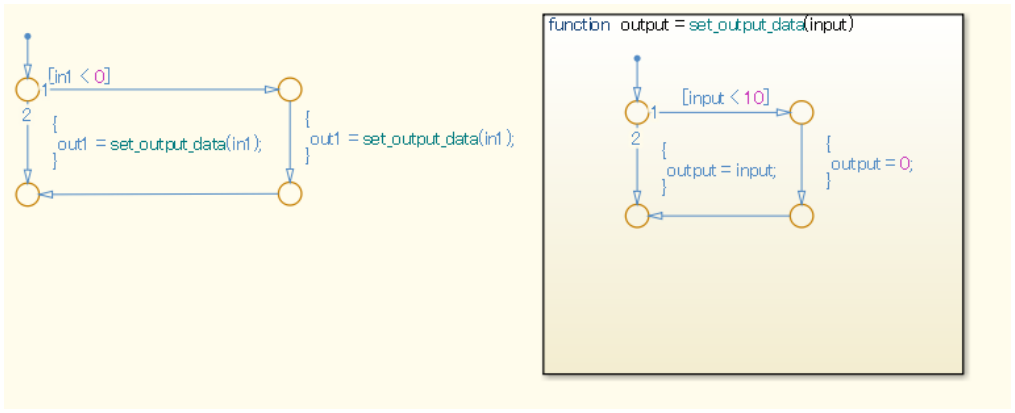
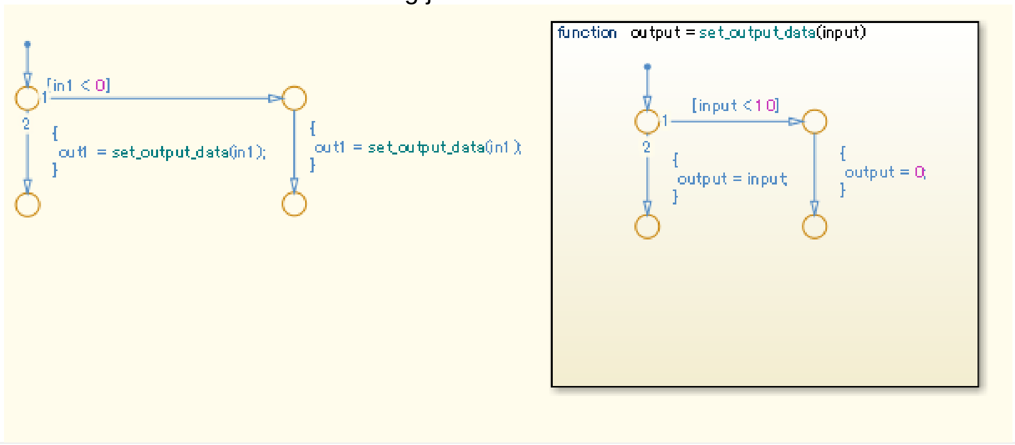
【Incorrect】
The {execution order} for the unconditional transition line is not the last value.

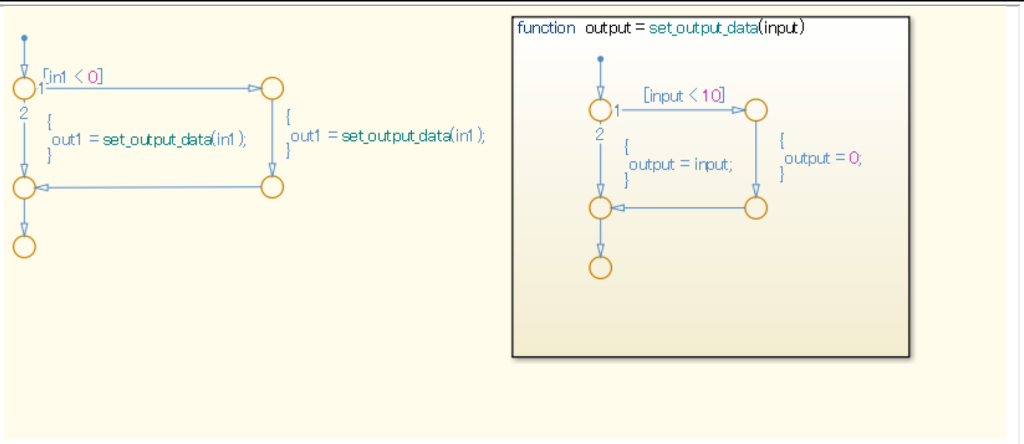


Rationale

Sub ID	Description
a	・Prevents unintended behavior that results from backtracking. Setting an unconditional transition explicitly defines the behavior for when the condition is not met.
b	・Setting the unconditional transition to take precedence can prevent unintended behavior.

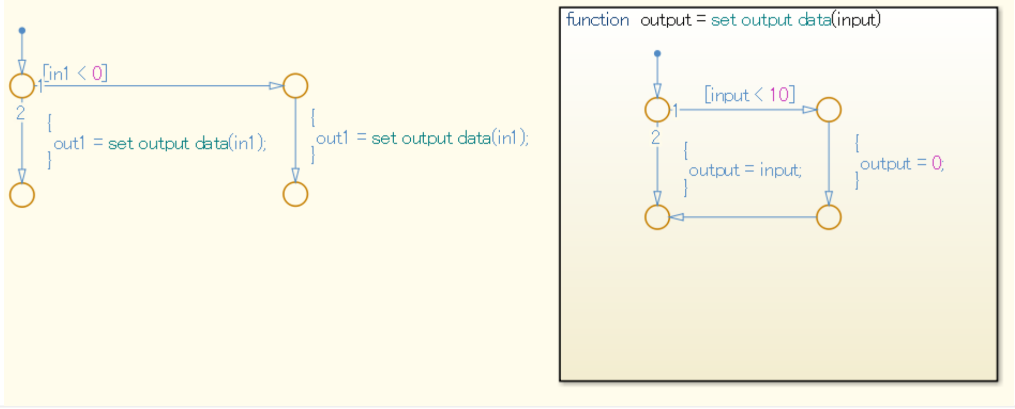
jc_0775: Terminating junctions in flow charts

Rule ID: Title	**jc_0775: Terminating junctions in flow charts**	
Sub ID Recommendations	NA-MAAB: a1/a2 JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	Only one terminating junction shall be used.	-
	【Correct】  [in1 < 0] 1 2 { out1 = set_output_data(in1); } { out1 = set_output_data(in1); } function output = set_output_data(input) [input < 10] { output = input; } { output = 0; } 【Incorrect】 There is more than one terminating junction.  [in1 < 0] 1 2 { out1 = set_output_data(in1); } { out1 = set_output_data(in1); } function output = set_output_data(input) [input < 10] { output = input; } { output = 0; }	
a2	One terminating junction with a single unconditional transition as the input shall be used.	-
	【Correct】	



[in1 < 0]
{ out1 = set_output_data(in1); }
{ out1 = set_output_data(in1); }
function output = set_output_data(input)
[input < 10]
{ output = input; }
{ output = 0; }

【Incorrect】
There is more than one terminating junction and input.

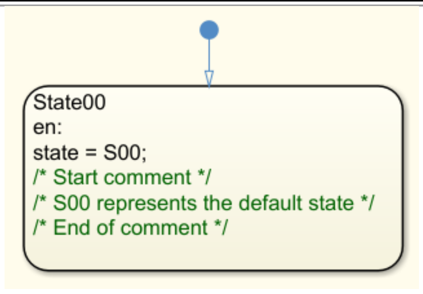
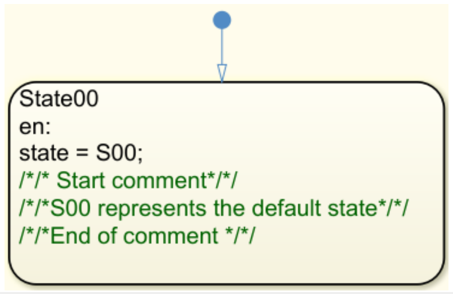
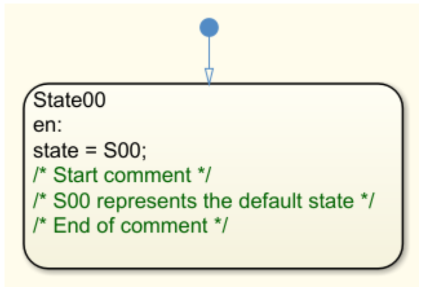
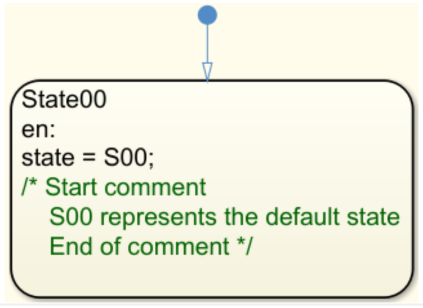


[in1 < 0]
{ out1 = set output data(in1); }
{ out1 = set output data(in1); }
function output = set output data(input)
[input < 10]
{ output = input; }
{ output = 0; }

Rationale	
Sub ID	**Description**
a1a2	・One terminating junction improves understanding of the logic end point. ・Using a consistent style for terminating junction improves readability.

jc_0738: Usage of Stateflow comments

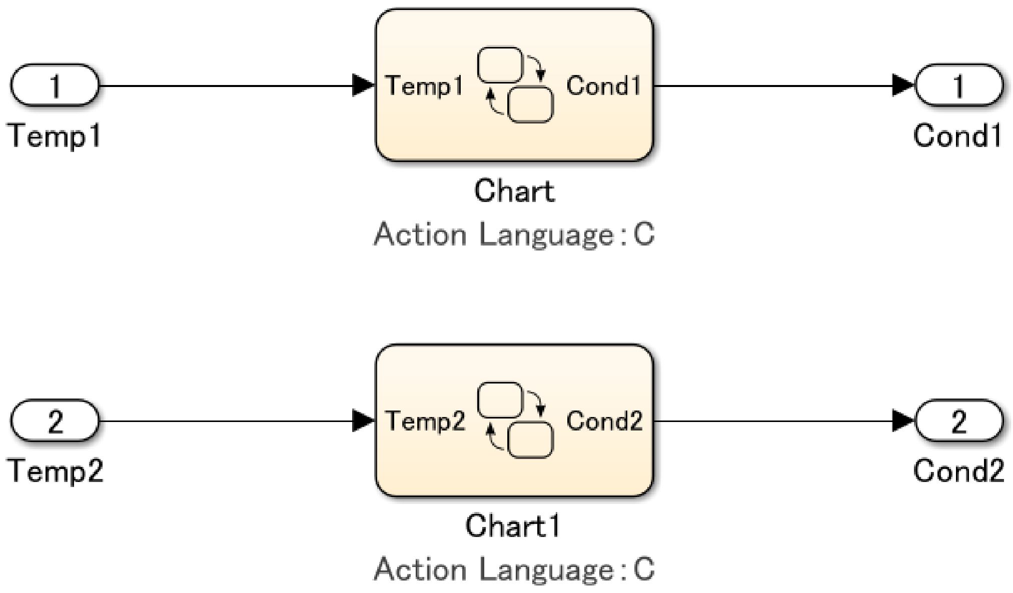
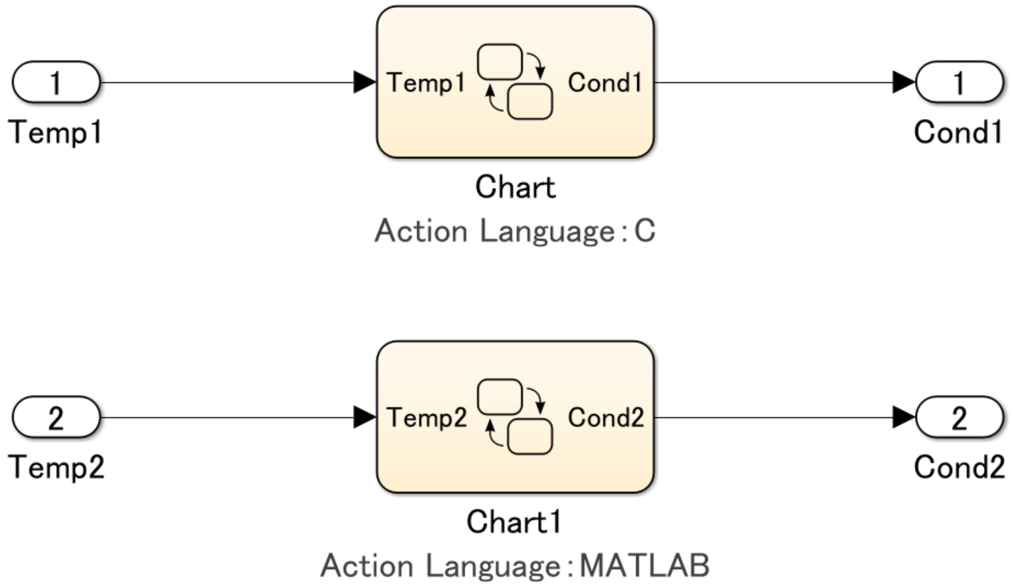
Rule ID: Title	**jc_0738: Usage of Stateflow comments**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When [Chart] parameter {Action Language} is set to "C", /*...*/ comment nesting shall not be used.	-
	【Correct】	

	 State00 en: state = S00; /* Start comment */ /* S00 represents the default state */ /* End of comment */ 【Incorrect】  State00 en: state = S00; /*/* Start comment*/*/ /*/*S00 represents the default state*/*/ /*/*End of comment */*/	
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.	-
	【Correct】  State00 en: state = S00; /* Start comment */ /* S00 represents the default state */ /* End of comment */ 【Incorrect】  State00 en: state = S00; /* Start comment S00 represents the default state End of comment */	

Rationale	
Sub ID	**Description**
a	The compiler can misinterpret the comments as a program.
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] property {Action Language} is set to “MATLAB”, comments must use %.

4.3. Conditional transition / Action

jc_0790: Action language of Chart block

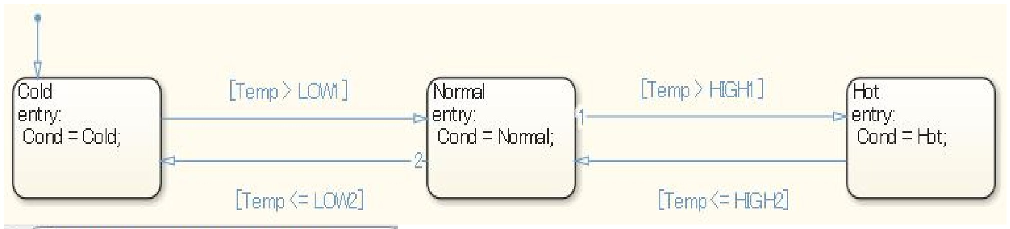
Rule ID: Title	**jc_0790: Action language of Chart block**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Chart] property {Action Language} shall be set to “C”.	-
	【Correct】 {Action Language} is set to “C”.  1 Temp1 Temp1 Cond1 Chart Action Language : C 1 Cond1 2 Temp2 Temp2 Cond2 Chart1 Action Language : C 2 Cond2 【Incorrect】 {Action Language} is set to “MATLAB”.  1 Temp1 Temp1 Cond1 Chart Action Language : C 1 Cond1 2 Temp2 Temp2 Cond2 Chart1 Action Language : MATLAB 2 Cond2	
Rationale		
Sub ID	**Description**	

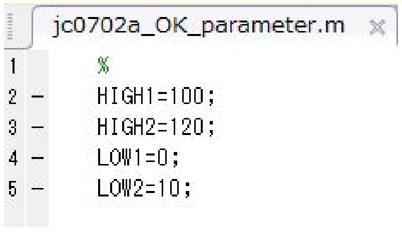
a	・Using a consistent action language improves readability because there is not a difference in syntax. ・Easier to maintain consistency between the model and the generated code when using C as the action language as compared to MATLAB. ・Easier to understand the model for users who are familiar with the C programming language.

jc_0702: Use of named Stateflow parameters/constants

Rule ID: Title	**jc_0702: Use of named Stateflow parameters/constants**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Stateflow] shall not use numeric literal. Exceptions: ・Initial value is 0 ・Increment, decrement 1	-

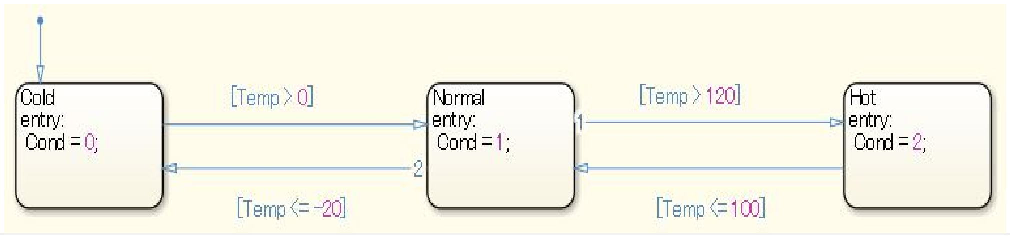
【Correct】
Numeric literals are not used.





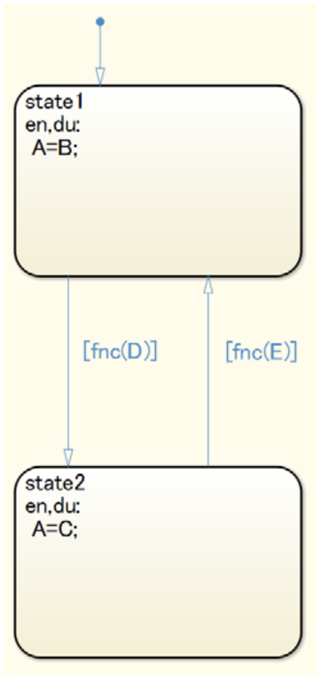
```
%
HIGH1=100;
HIGH2=120;
LOW1=0;
LOW2=10;
```

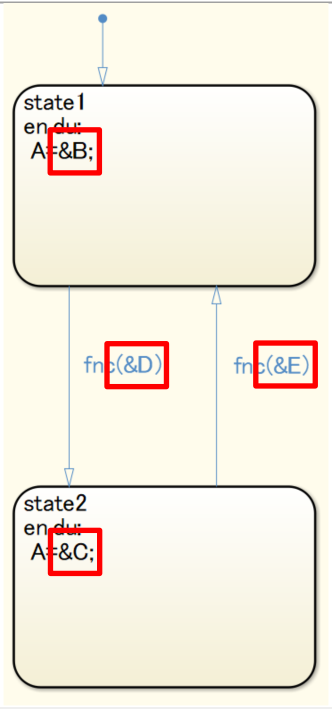
【Incorrect】
0711



Rationale	
Sub ID	**Description**
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.

jm_0011: Pointers in Stateflow

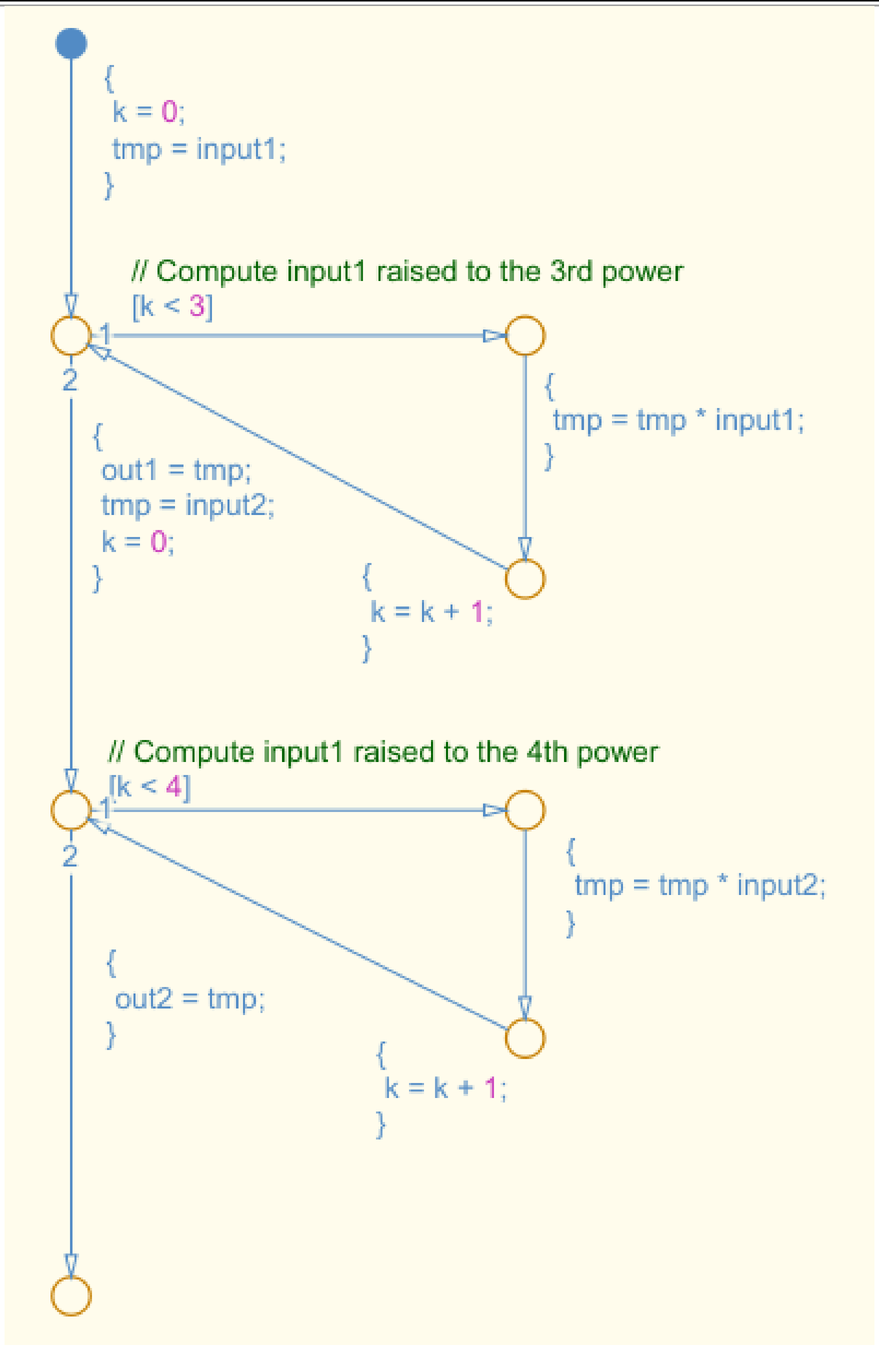
Rule ID: Title	**jm_0011: Pointers in Stateflow**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[Stateflow] shall not use pointer variables.	-
	【Correct】 Pointer variables are not used.  state1 en,du: A=B; [fnc(D)] [fnc(E)] state2 en,du: A=C; 【Incorrect】 Pointer variables are used.	

	 state1
	en,du:
	A=&B;
	fnc(&D)
	fnc(&E)
	state2
	en,du:
	A=&C;

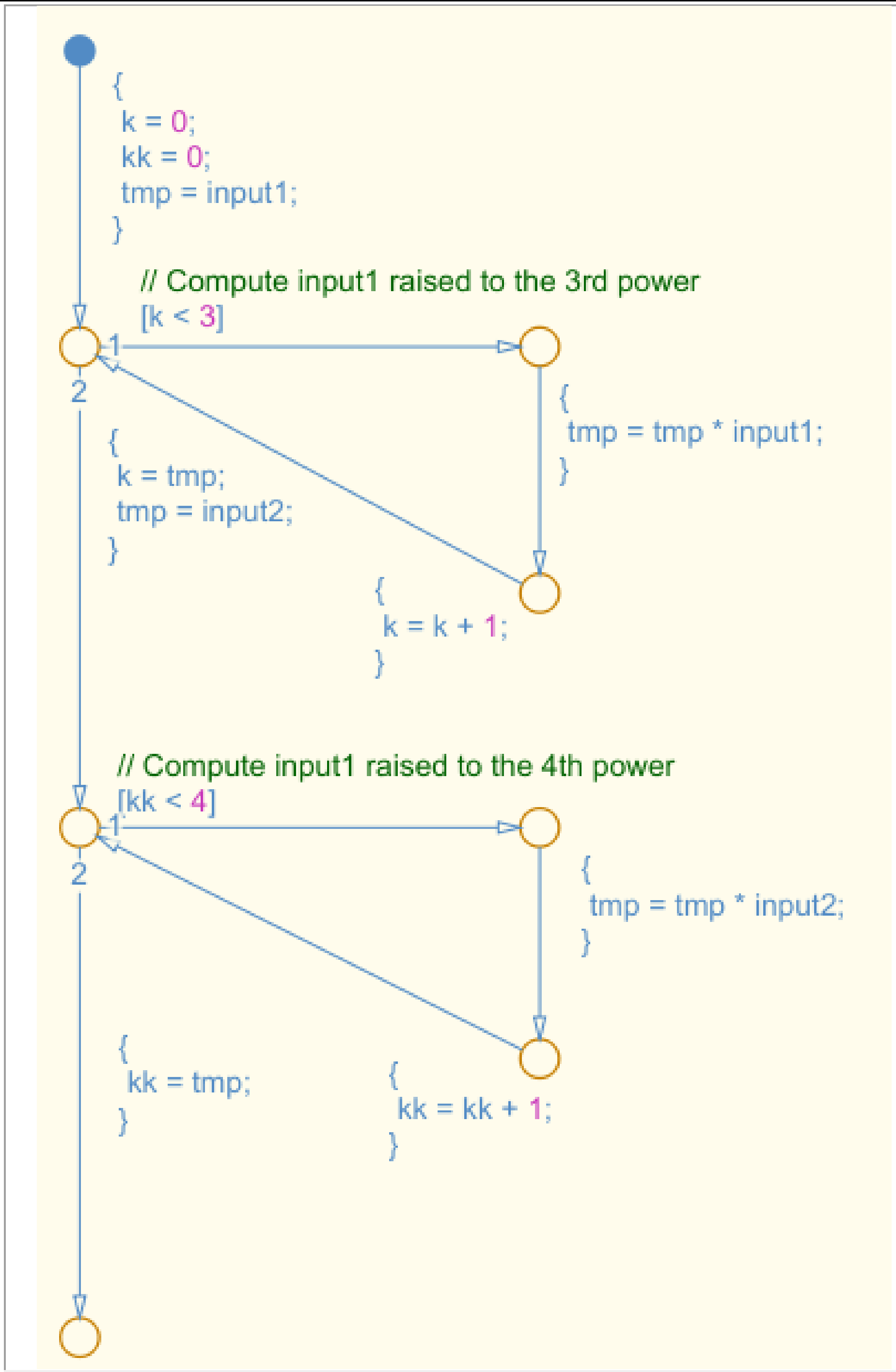
Rationale	
Sub ID	**Description**
a	・Readability is impaired when pointer variables are used. ・Code generation may not be possible.

jc_0491: Reuse of Stateflow data

Rule ID: Title	**jc_0491: Reuse of Stateflow data**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	A variable shall not have multiple meanings (usages) in a single [Chart].	-
	【Correct】 A variable does not have multiple meanings (usages).	



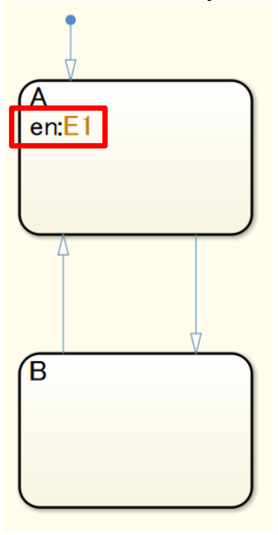
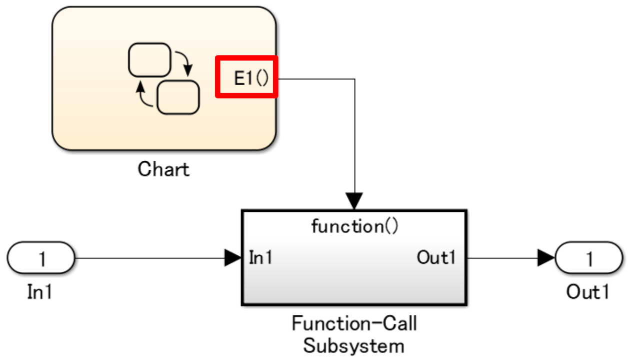
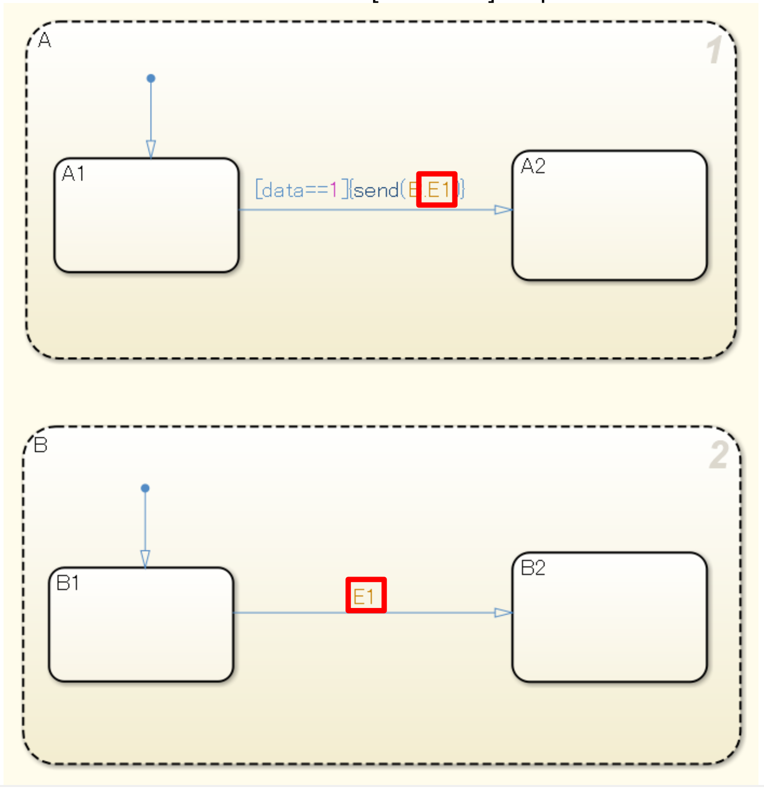
【Incorrect】
Variables k and kk have multiple meanings (usages) in a single [Chart].

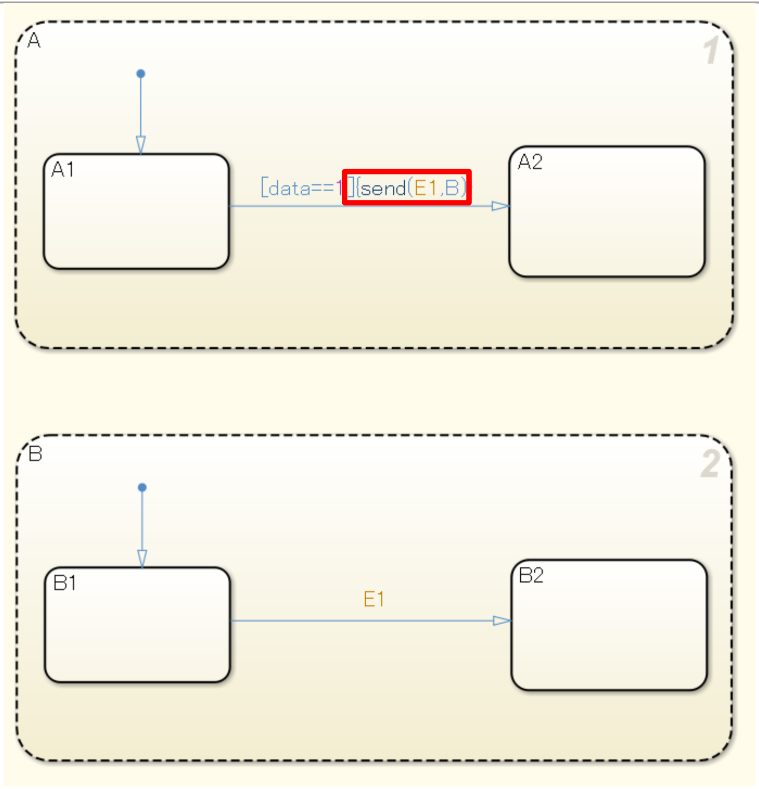
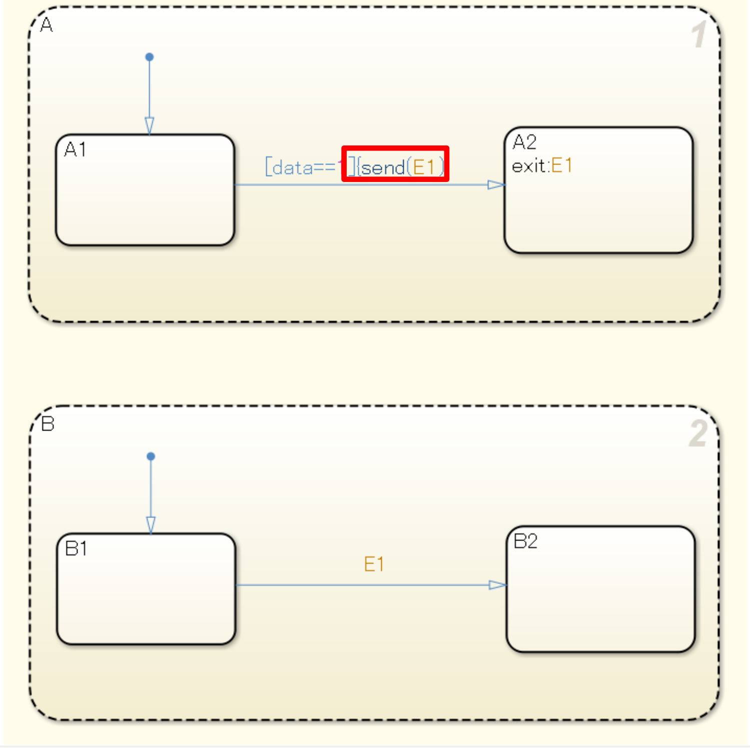
	 {
	k = 0;
	kk = 0;
	tmp = input1;
	}
	// Compute input1 raised to the 3rd power
	[k < 3]
	1
	2
	{
	tmp = tmp * input1;
	}
	{
	k = tmp;
	tmp = input2;
	}
	{
	k = k + 1;
	}
	// Compute input1 raised to the 4th power
	[kk < 4]
	1
	2
	{
	tmp = tmp * input2;
	}
	{
	kk = tmp;
	}
	{
	kk = kk + 1;
	}

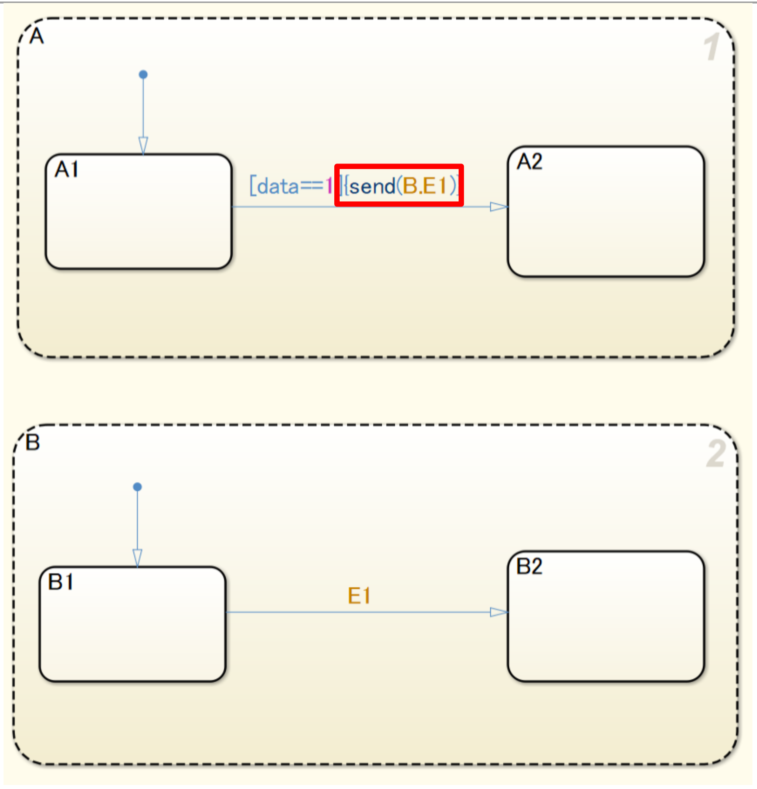
Rationale	
Sub ID	**Description**
a	· Variables can be misinterpreted when the variable name is different than the meaning of the numerical value that is assigned to the variable.

jm_0012: Usage restrictions of events and broadcasting events

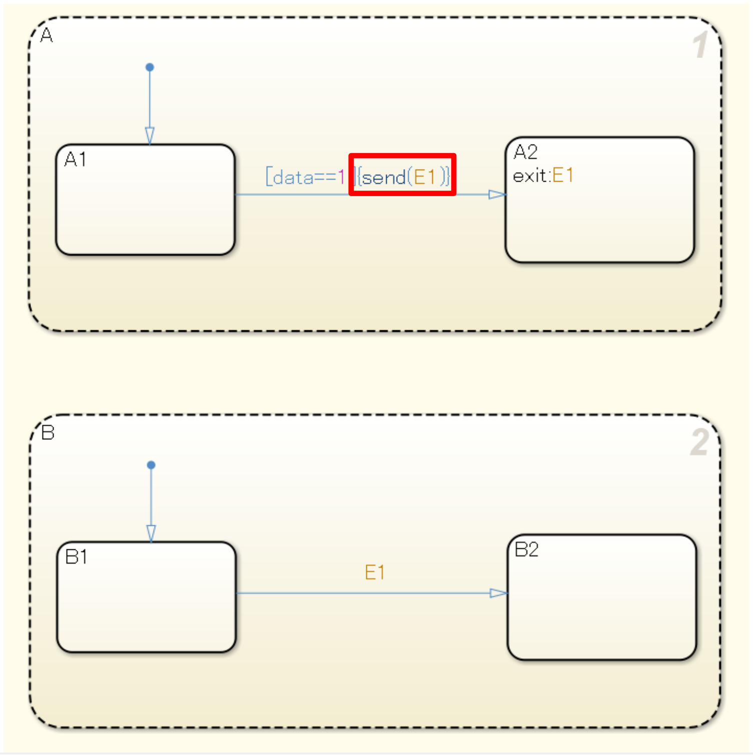
Rule ID: Title	**jm_0012: Usage restrictions of events and broadcasting events**
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a1/a2/a3

MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	Stateflow events shall be used only in [Stateflow] output.	-
	【Correct】 Event is used only in the [Stateflow] output.   E1() Chart function() 1 In1 In1 Out1 1 Out1 Function-Call Subsystem 【Incorrect】 Event is used other than in the [Stateflow] output.  A 1 A1 [data==1]{send(E1)} A2 B 2 B1 E1 B2	
a2	Send syntax `send(event_name, state_name)`shall be used to broadcast Stateflow events.	-
	【Correct】 Event is broadcast using the send syntax.	

	 A 1 A1 [data==1][send(E1,B)] A2 B 2 B1 E1 B2 【Incorrect】 The state that receives the broadcast has not been defined in the send syntax.  A 1 A1 [data==1][send(E1)] A2 exit:E1 B 2 B1 E1 B2	
a3	Send syntax `send(state_name.event_name)`with the qualified event name shall be used to broadcast Stateflow events.	-
	【Correct】 The qualified event name is used in the event being broadcast.	



【Incorrect】
The state that receives the broadcast has not been described in the send syntax.

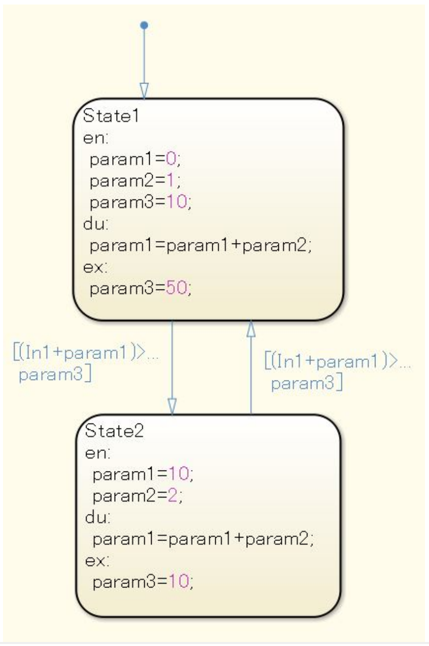
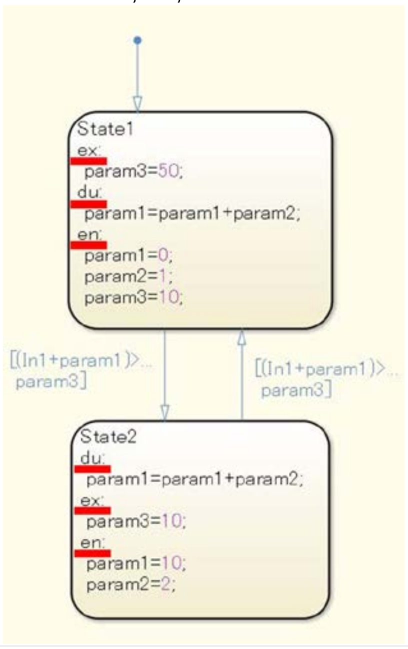


Rationale	
Sub ID	**Description**
a1	・Recursive processing in a chart is prevented by using Stateflow events in [Stateflow] output only.

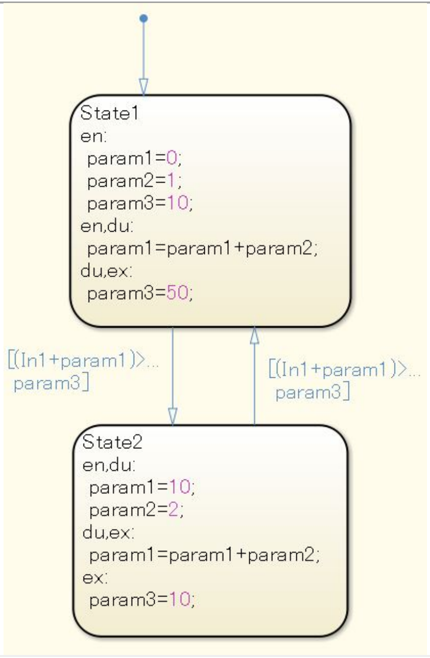
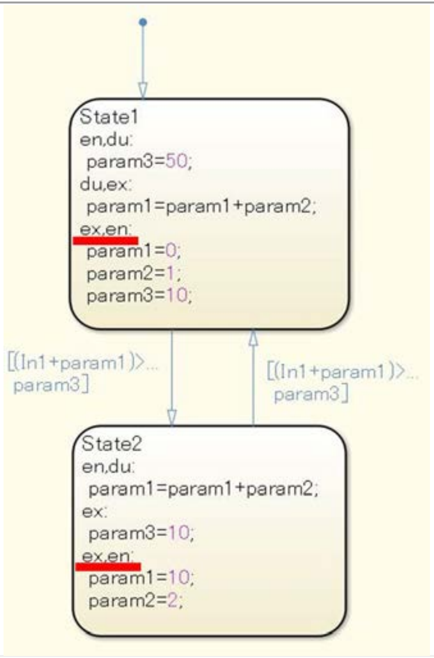
a2a3	・Improves readability because transitions that are triggered by events are clearly identified.

jc_0733: Order of state action types

Rule ID: Title	**jc_0733: Order of state action types**	
Sub ID Recommendations	NA-MAAB: a, b JMAAB: a, b	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Basic state action types shall be stated in this order: entry (en) during (du) exit (ex)	-

	【Correct】
	Order is en, du, ex.
	 State1
	en:
	param1=0;
	param2=1;
	param3=10;
	du:
	param1=param1+param2;
	ex:
	param3=50;
	[(In1+param1)>... param3]
	[(In1+param1)>... param3]
	State2
	en:
	param1=10;
	param2=2;
	du:
	param1=param1+param2;
	ex:
	param3=10;
	【Incorrect】
	Not in en, du, ex order.
	 State1
	ex:
	param3=50;
	du:
	param1=param1+param2;
	en:
	param1=0;
	param2=1;
	param3=10;
	[(In1+param1)>... param3]
	[(In1+param1)>... param3]
	State2
	du:
	param1=param1+param2;
	ex:
	param3=10;
	en:
	param1=10;
	param2=2;	

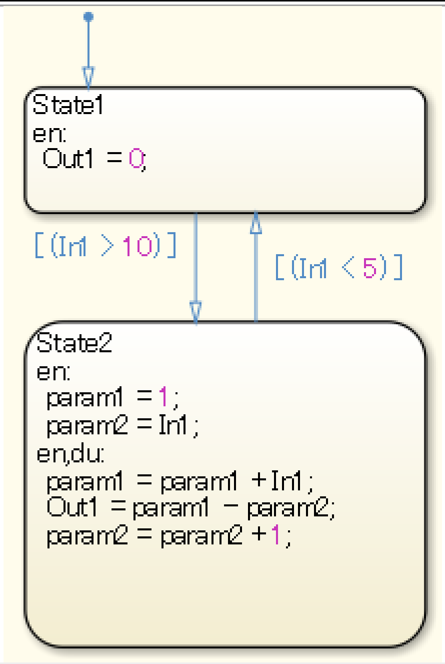
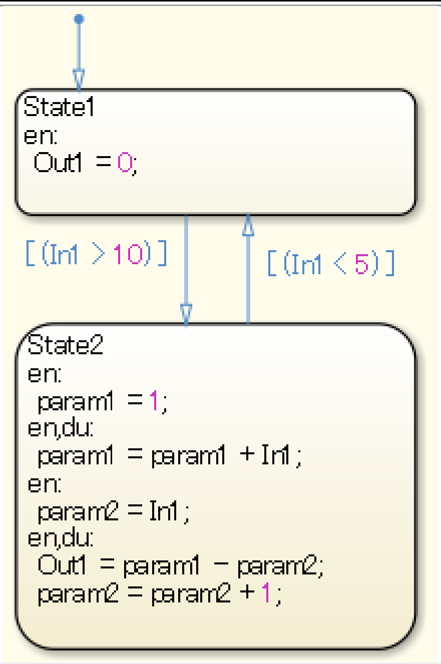
b	Combined state action types shall be stated in this order: entry (en) during (du) exit (ex)	-
	【Correct】 Order is en, du, ex. 【Incorrect】 Not in en, du, ex order.	

	 State1
	en:
	param1=0;
	param2=1;
	param3=10;
	en,du:
	param1=param1+param2;
	du,ex:
	param3=50;
	[(In1+param1)>...
	param3]
	[(In1+param1)>...
	param3]
	State2
	en,du:
	param1=10;
	param2=2;
	du,ex:
	param1=param1+param2;
	ex:
	param3=10;  State1
	en,du:
	param3=50;
	du,ex:
	param1=param1+param2;
	ex,en:
	param1=0;
	param2=1;
	param3=10;
	[(In1+param1)>...
	param3]
	[(In1+param1)>...
	param3]
	State2
	en,du:
	param1=param1+param2;
	ex:
	param3=10;
	ex,en:
	param1=10;
	param2=2;

Rationale	
Sub ID	**Description**
ab	・Consistent modelling improves readability and maintainability.

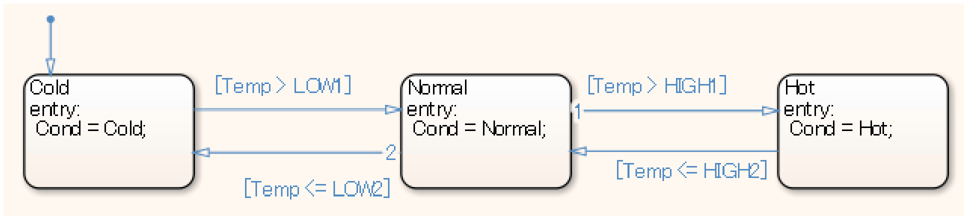
jc_0734: Number of state action types

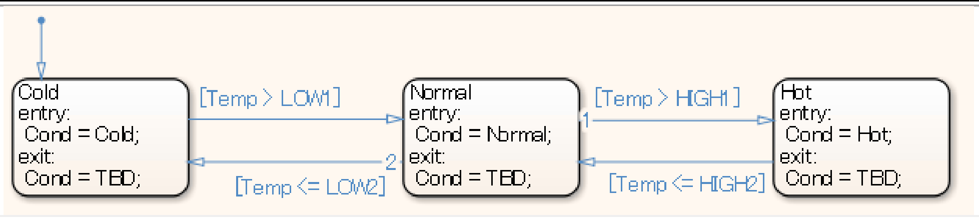
Rule ID: Title	**jc_0734: Number of state action types**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	State action types shall not describe the same thing more than twice.	-
	【Correct】 【Incorrect】	

	 State1
	en:
	Out1 = 0;
	[(In1 > 10)]
	[(In1 < 5)]
	State2
	en:
	param1 = 1;
	param2 = In1;
	en,du:
	param1 = param1 + In1;
	Out1 = param1 − param2;
	param2 = param2 + 1;  State1
	en:
	Out1 = 0;
	[(In1 > 10)]
	[(In1 < 5)]
	State2
	en:
	param1 = 1;
	en,du:
	param1 = param1 + In1;
	en:
	param2 = In1;
	en,du:
	Out1 = param1 − param2;
	param2 = param2 + 1;

Rationale	
Sub ID	**Description**
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.

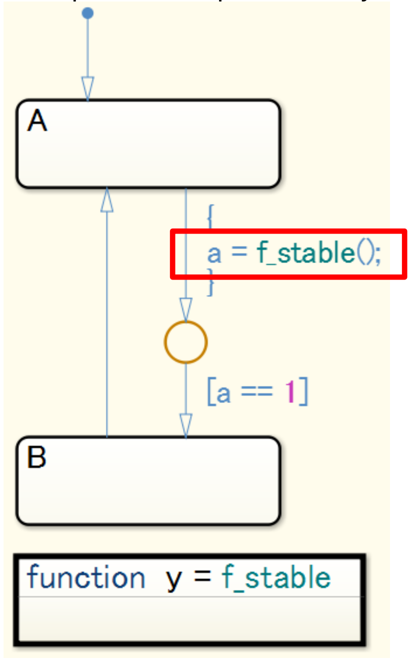
jc_0740: Limitation on use of exit state action

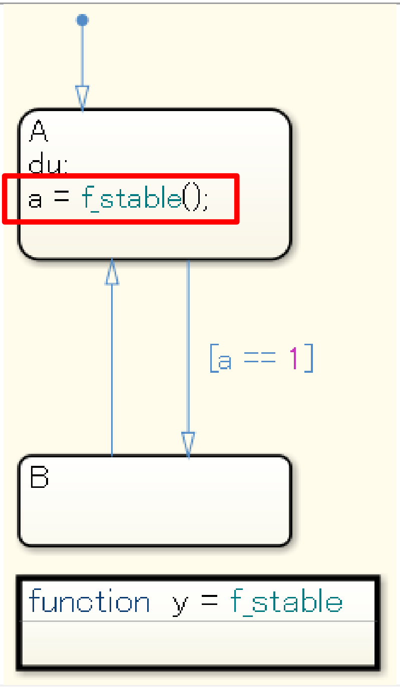
Rule ID: Title	**jc_0740: Limitation on use of exit state action**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	State action type exit(ex) shall not be used.	-
	【Correct】  Cold entry: Cond = Cold; [Temp > LOW1] [Temp <= LOW2] 2 Normal entry: Cond = Normal; 1 [Temp > HIGH1] [Temp <= HIGH2] Hot entry: Cond = Hot; 【Incorrect】 This example illustrates how the model behavior in [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 [Chart].	

	 Cold entry: Cond = Cold; exit: Cond = TBD; [Temp > LOW1] [Temp <= LOW2] 2 Normal entry: Cond = Normal; exit: Cond = TBD; [Temp > HIGH1] 1 [Temp <= HIGH2] Hot entry: Cond = Hot; exit: Cond = TBD;
Rationale	
Sub ID	**Description**
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.

jc_0741: Timing to update data used in state chart transition conditions

Rule ID: Title	**jc_0741: Timing to update data used in state chart transition conditions**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Variables that are used in a state transition condition shall not use “during” to perform an update.	-

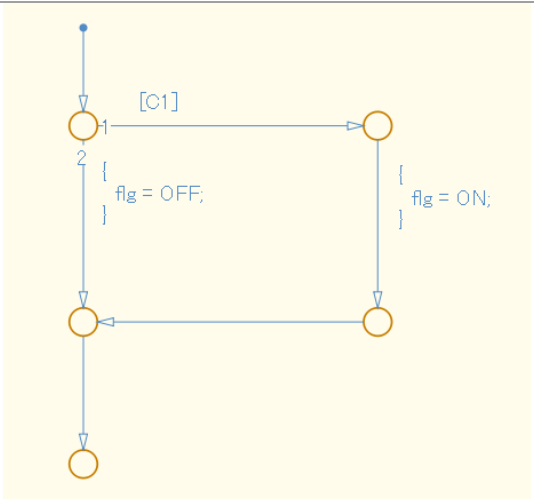
	【Correct】
	The update is not performed by using “during”.
	 A
	{
	a = f_stable();
	}
	[a == 1]
	B
	function y = f_stable
	【Incorrect】
	The update is performed by using “during”.	

	 A du: a = f_stable(); [a == 1] B function y = f_stable

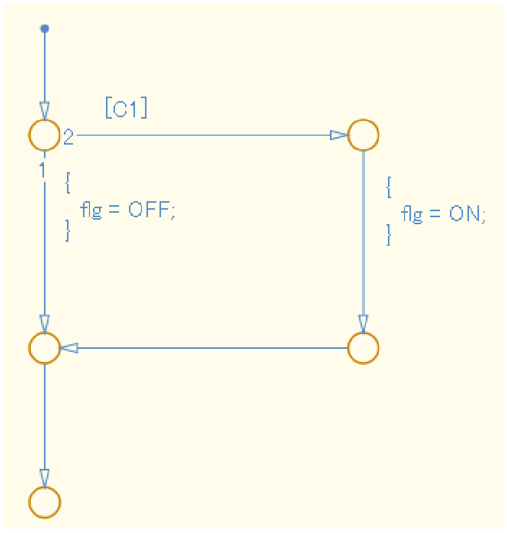
Rationale	
Sub ID	**Description**
a	· The execution order of the transition condition and implement of “during” can be difficult to understand, which increases the risk of errors.

jc_0772: Execution order and transition conditions of transition lines

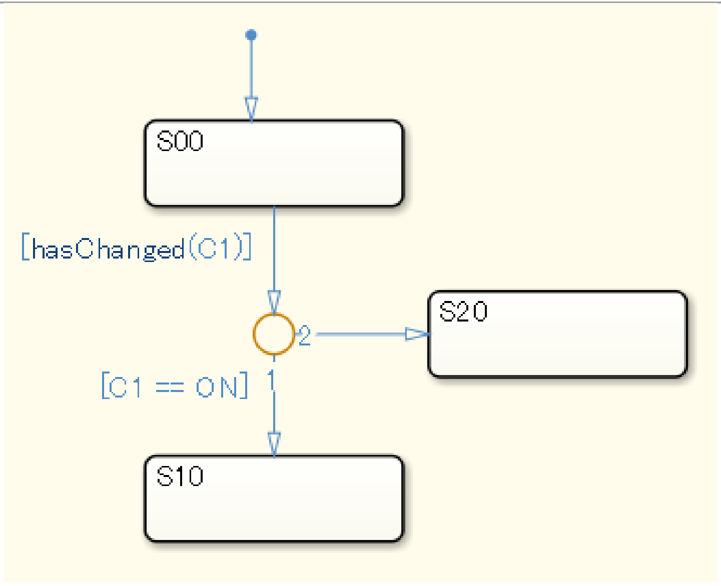
Rule ID: Title	**jc_0772: Execution order and transition conditions of transition lines**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	All transition paths shall be executable. (R2011b to R2016a) Configuration parameter {Transition shadowing} shall be set to “error”. (R2016b and later) Configuration parameter {Unreachable execution path] shall be set to “error”.	-
	【Correct】	



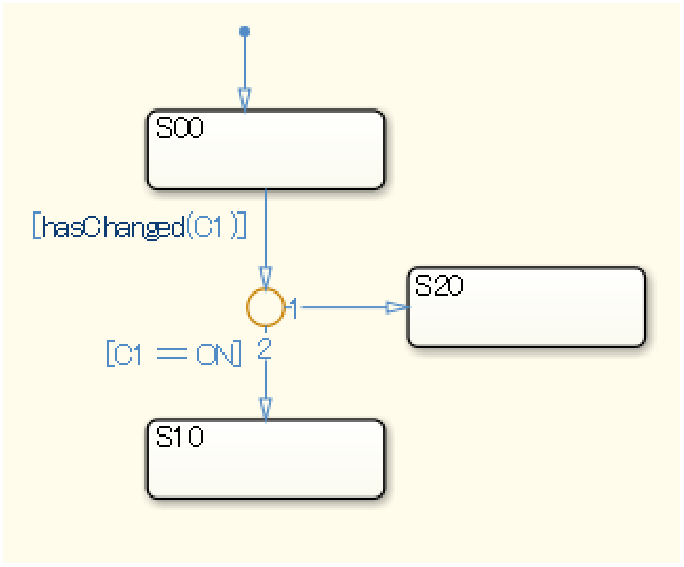
【Incorrect】
  Execution order 1 is an unconditional transition and conditional expression [C1] is described in execution condition 2.



【Correct】
Includes a state transition.



【Incorrect】
Includes state transition. The unconditional transition line is higher in the execution order than the conditional transition line.

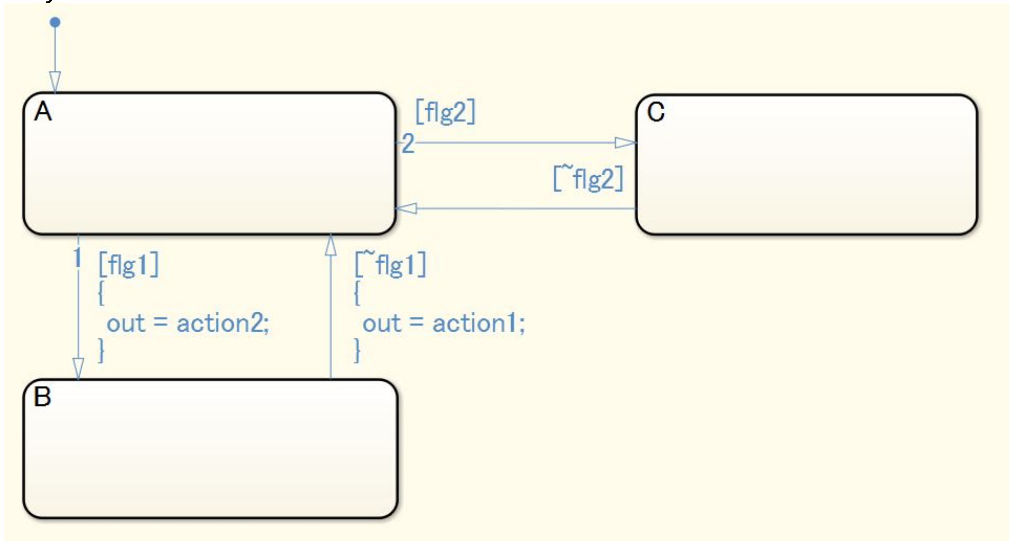
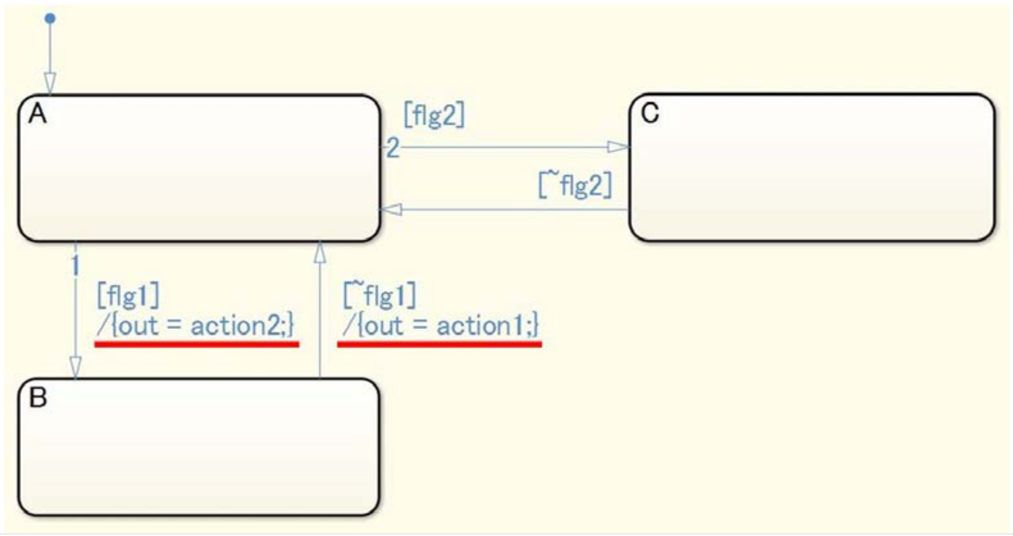


Rationale	
Sub ID	**Description**
a	・An unconditional transition that is in any position other than the last in the execution order causes the subsequent transition to be a dead path, which results in unintended simulation behavior.

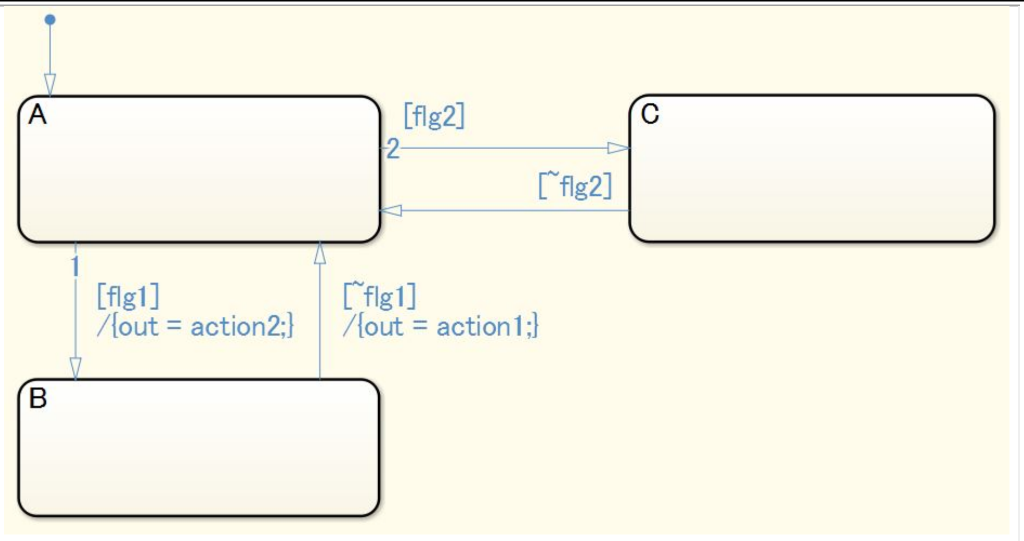
jc_0753: Condition actions and transition actions in Stateflow

Rule ID: Title	**jc_0753: Condition actions and transition actions in Stateflow**
Sub ID Recommendations	NA-MAAB: a1/a2 JMAAB: a1/a2
MATLAB® Version	All
Rule	

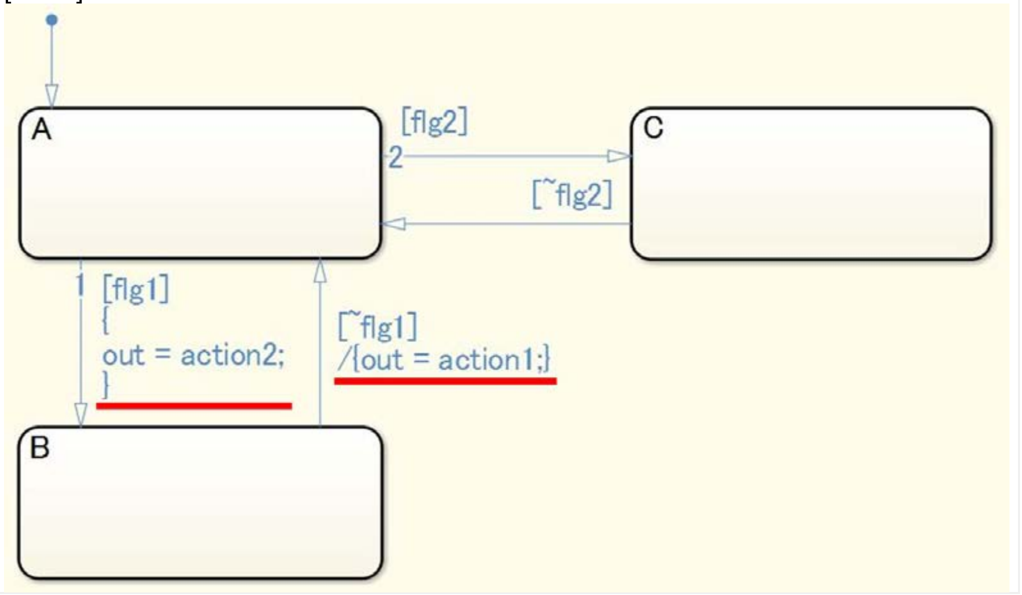
Sub ID	Description	Custom Parameter
a1	Transition actions shall not be used in a state chart.	-

	【Correct】
	Only a condition action is used in the state chart.
	 A
	B
	C
	[flg2]
	2
	[~flg2]
	1
	[flg1]
	{
	out = action2;
	}
	[~flg1]
	{
	out = action1;
	}
	【Incorrect】
	A transition action is used in the state chart.
	 A
	B
	C
	[flg2]
	2
	[~flg2]
	1
	[flg1]
	/{out = action2;}
	[~flg1]
	/{out = action1;}	

a2	Condition actions and transition actions shall not be combined in the same [Chart].	-
	【Correct】 Either a condition action or a transition action can be used in a [Chart]. (The following diagram illustrates a transition action.)	



【Incorrect】
[Chart] 0774

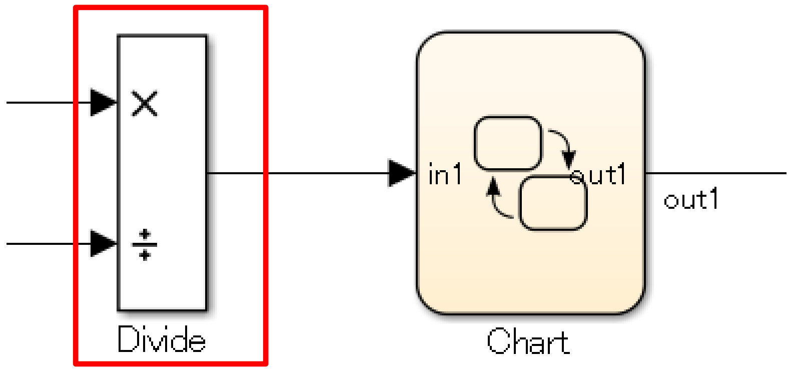
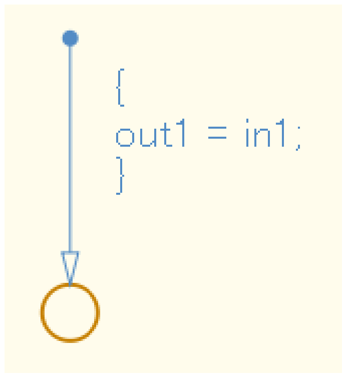
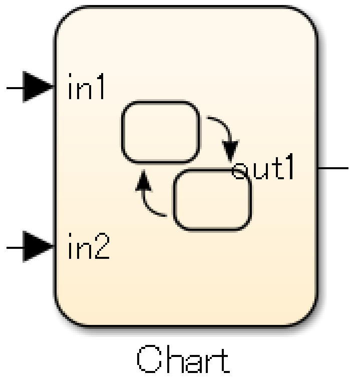


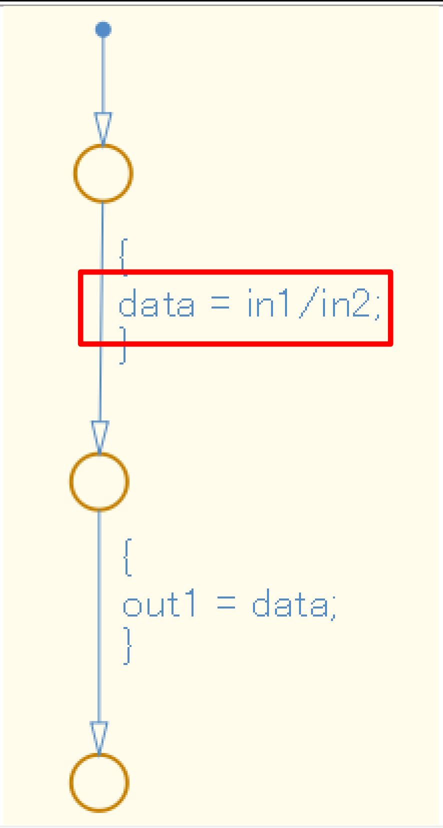
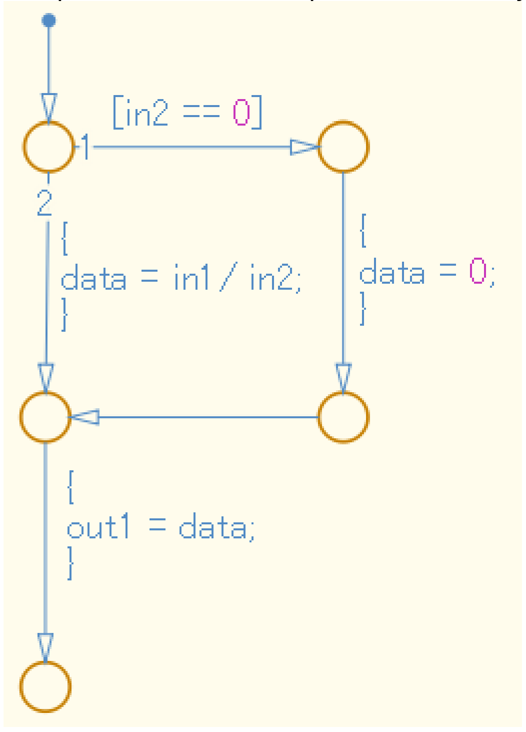
Rationale	
Sub ID	**Description**
a1	・Prevents confusion with a condition action, thus improving readability.
a2	・A condition action executes upon entering a transition. A transition action executes after determining whether it can transition to the next state. Adherence to the rule prevents confusion between a conditional action and a transition action.

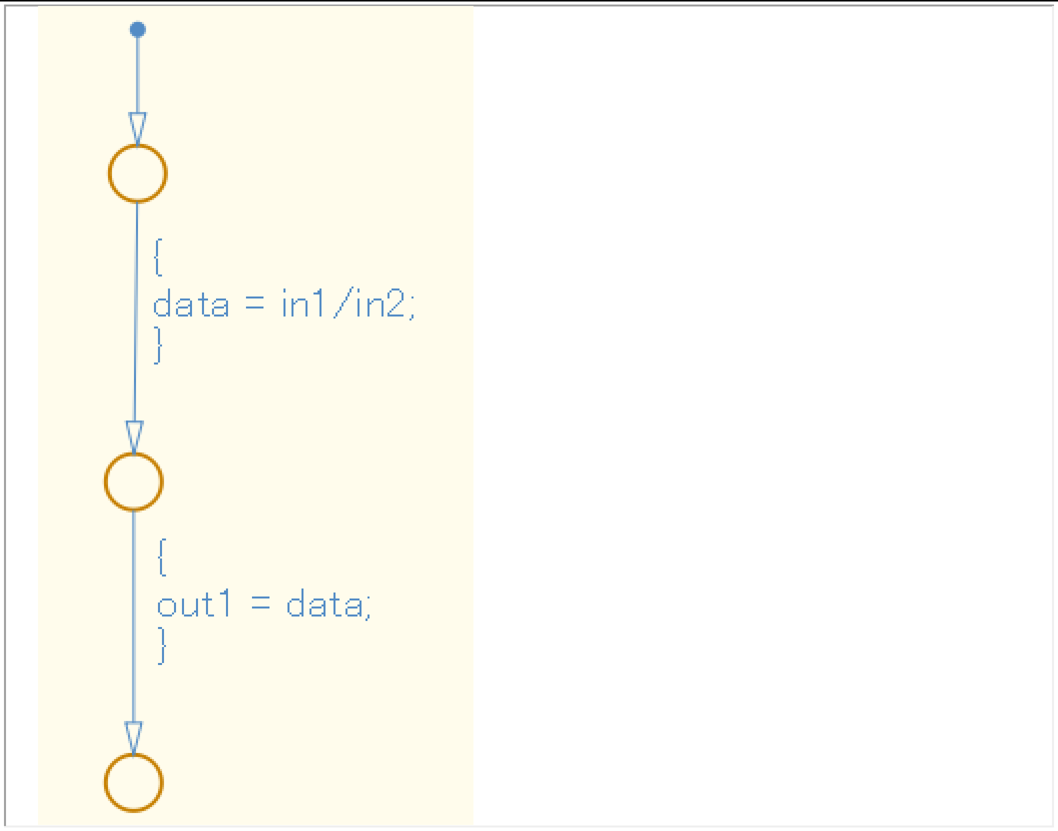
jc_0711: Division in Stateflow

Rule ID: Title	**jc_0711: Division in Stateflow**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

a1	Variables, constants, or parameters in a Stateflow block shall not be used to perform division operations.	-

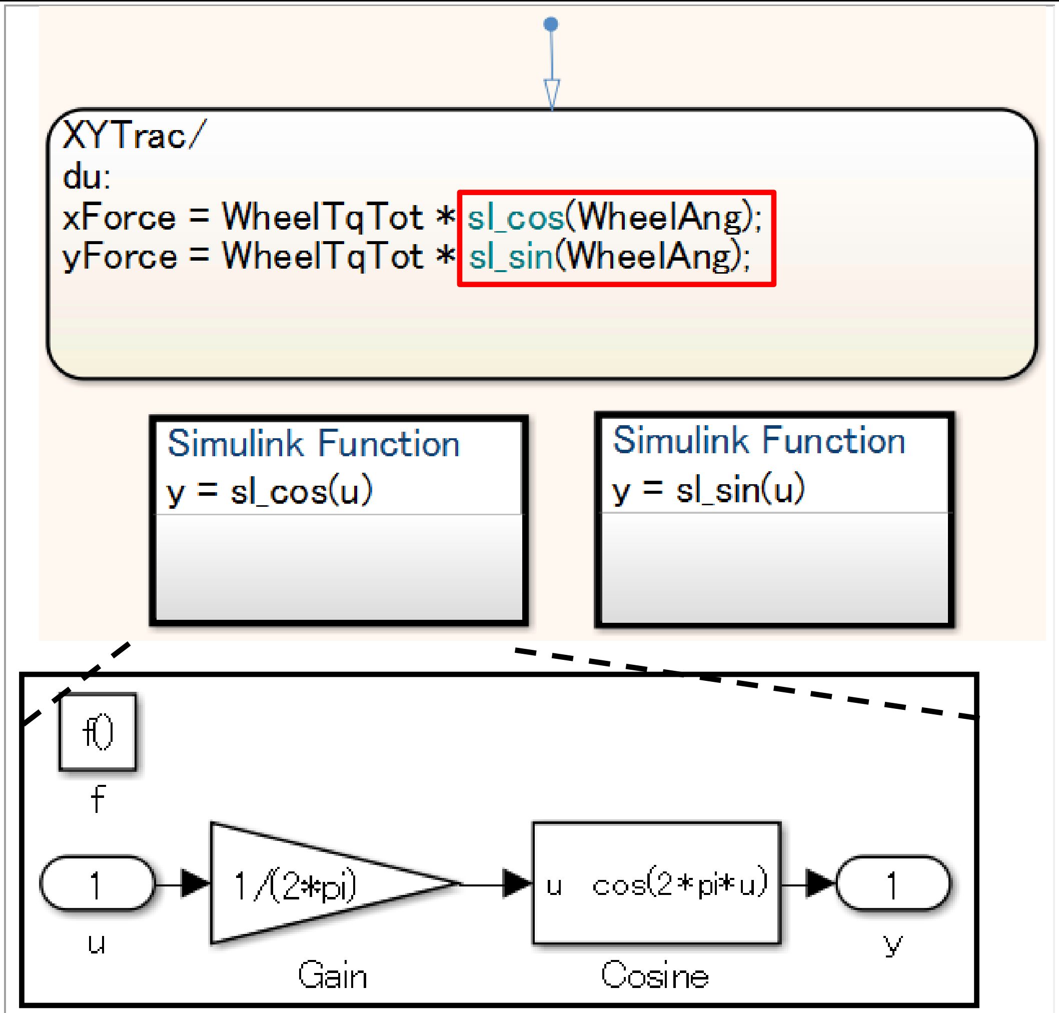
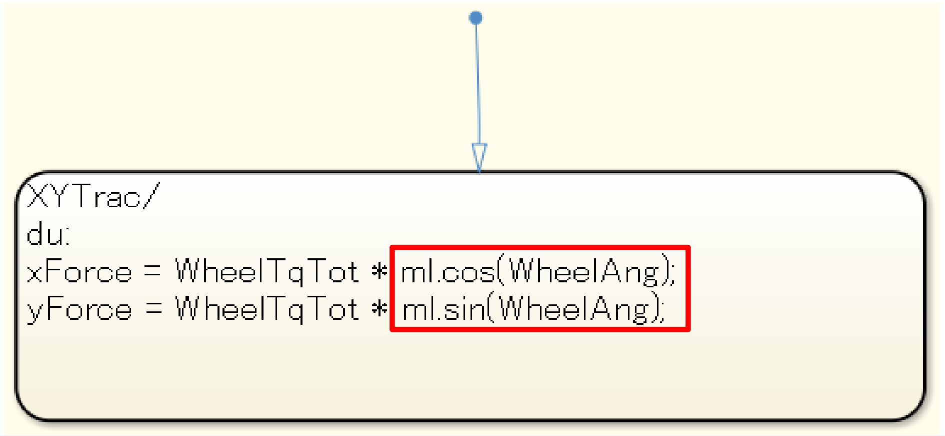
	【Correct】
	Division is performed outside of [Chart].
	
	
	{
	out1 = in1;
	}
	
	【Incorrect】
	Division occurs within [Chart].
		

		
a2	When division occurs in a Stateflow block, the process shall prevent division by zero.	-
	【Correct】 The process is defined to prevent division by zero.  [in2 == 0] 1 2 { data = in1 / in2; } { data = 0; } { out1 = data; } 【Incorrect】 The process does not prevent division by zero.	

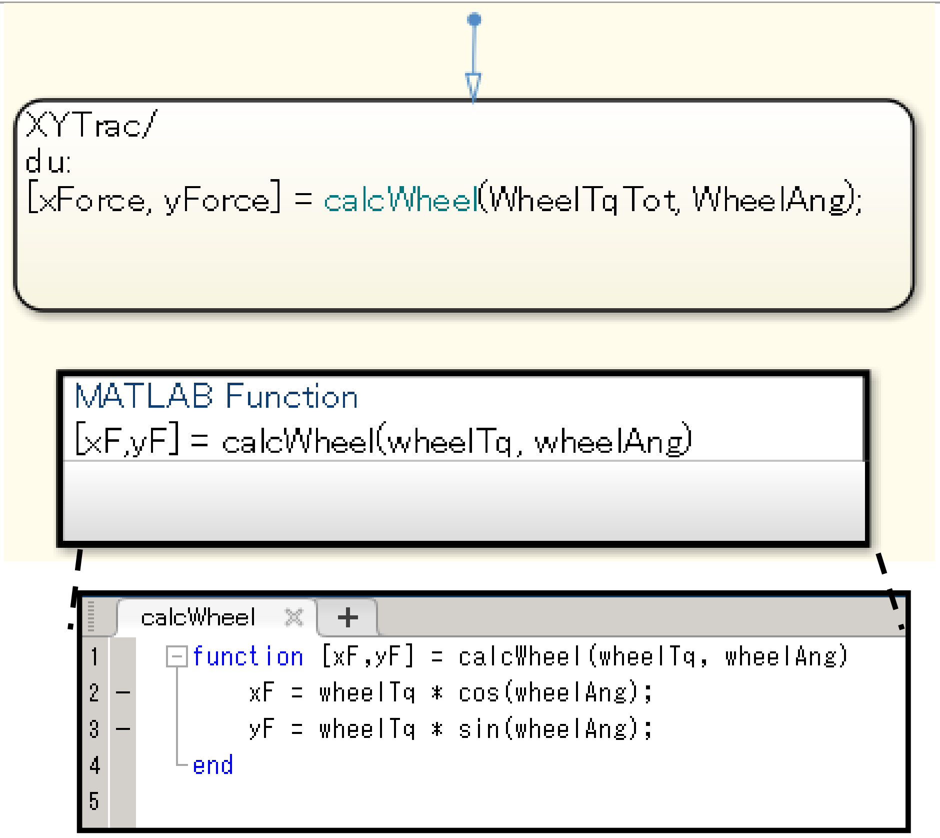
	
Rationale	
Sub ID	**Description**
a1a2	・Deviation from the rule can cause unintended operation and code generation results.

db_0127: Limitation on MATLAB commands in Stateflow blocks

Rule ID: Title	**db_0127: Limitation on MATLAB commands in Stateflow blocks**	
Sub ID Recommendations	NA-MAAB: a1/a2 JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	MATLAB commands shall not be used in Stateflow blocks.	-
	【Correct】 MATLAB commands are not used in Stateflow blocks.	

	 XYTrac/
	du:
	xForce = WheelTqTot * sl_cos(WheelAng);
	yForce = WheelTqTot * sl_sin(WheelAng);
	 Simulink Function
	y = sl_cos(u)
	Simulink Function
	y = sl_sin(u)
	f()
	f
	1
	u
	1/(2*pi)
	Gain
	u cos(2*pi*u)
	Cosine
	1
	y
	【Incorrect】
	A MATLAB command is used in Stateflow blocks.
	 XYTrac/
	du:
	xForce = WheelTqTot * ml.cos(WheelAng);
	yForce = WheelTqTot * ml.sin(WheelAng);	

a2	When a MATLAB command is used in Stateflow blocks, it shall be accessed only by using [MATLAB Function].	-
	【Correct】 The MATLAB command is accessed by using [MATLAB Function].	

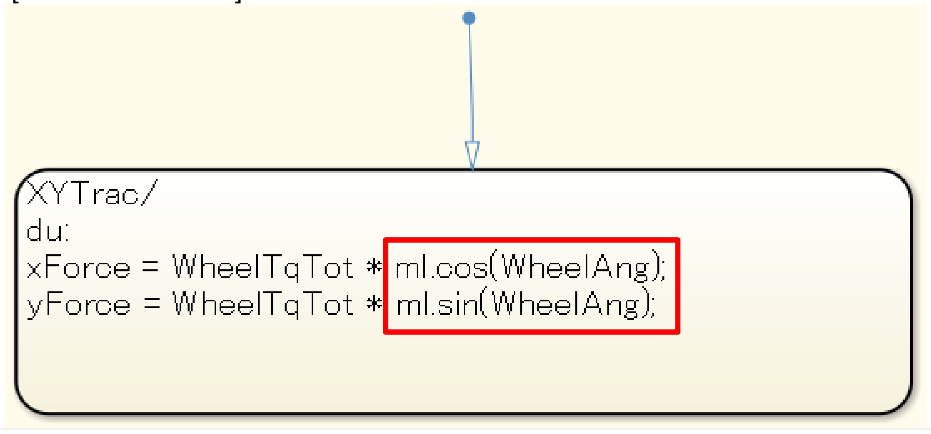


```
XYTrac/
du:
[xForce, yForce] = calcWheel(WheelTqTot, WheelAng);
```

```
MATLAB Function
[xF,yF] = calcWheel(wheelTq, wheelAng)
```

```
function [xF,yF] = calcWheel(wheelTq, wheelAng)
    xF = wheelTq * cos(wheelAng);
    yF = wheelTq * sin(wheelAng);
end
```

【Incorrect】
[MATLAB Function] is not used for a MATLAB command.



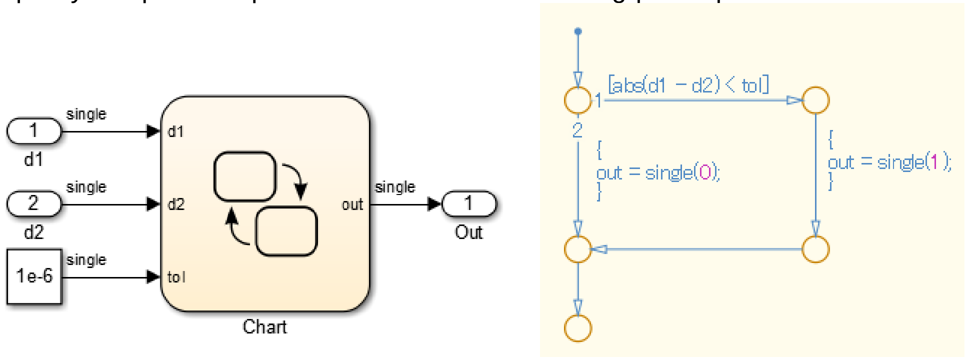
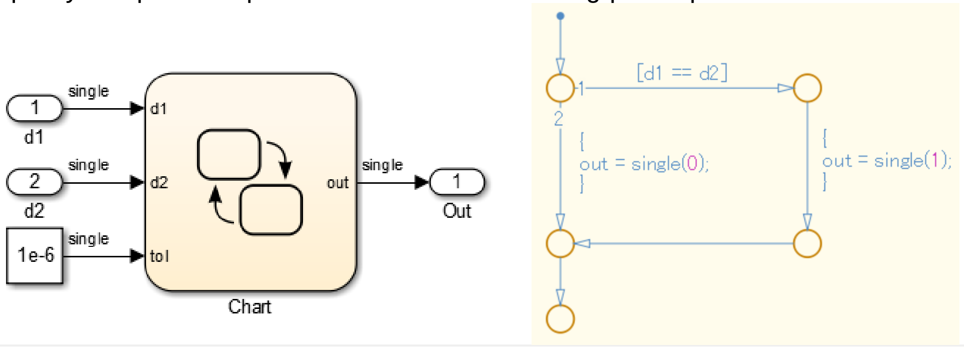
```
XYTrac/
du:
xForce = WheelTqTot * ml.cos(WheelAng);
yForce = WheelTqTot * ml.sin(WheelAng);
```

Rationale	
Sub ID	**Description**
a1	・Not all MATLAB commands are supported for code generation. As a result, code may not be generated for these unsupported MATLAB commands.
a2	・Not all MATLAB commands are supported for code generation. As a result, code may not be generated for these unsupported MATLAB commands. ・Readability improves when C and MATLAB action languages are described separately.

jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow

Rule ID: Title	**jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow**

Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	These equality comparison operators shall not be used in floating-point operands: • "==" • "!=" • "~="	-

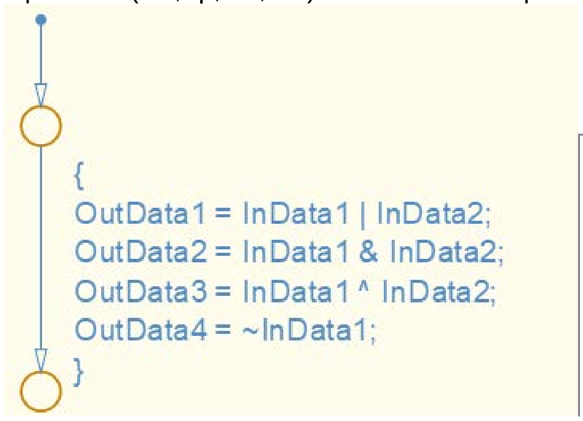
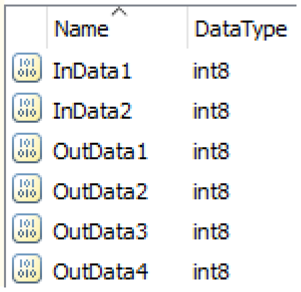
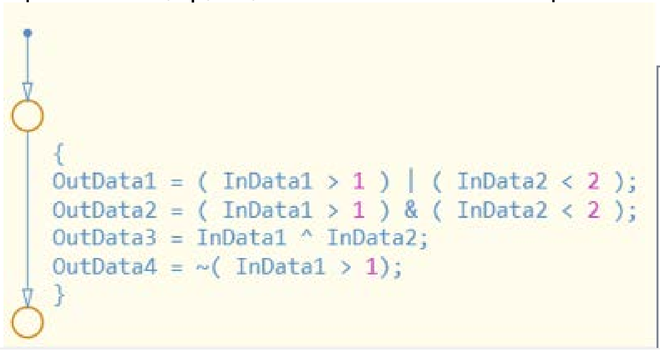
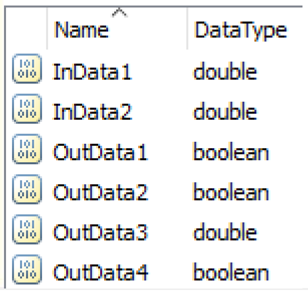
	【Correct】
	Equality comparison operators are not used in floating-point operands.
	 single
	1
	d1
	single
	2
	d2
	single
	1e-6
	d1
	d2
	tol
	out
	Chart
	single
	1
	Out
	[abs(d1 − d2) < tol]
	1
	2
	{
	out = single(0);
	}
	{
	out = single(1);
	}
	【Incorrect】
	Equality comparison operator "==" is used in floating-point operands.
	 single
	1
	d1
	single
	2
	d2
	single
	1e-6
	d1
	d2
	tol
	out
	Chart
	single
	1
	Out
	[d1 == d2]
	1
	2
	{
	out = single(0);
	}
	{
	out = single(1);
	}	

Rationale		
Sub ID	**Description**	
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.	

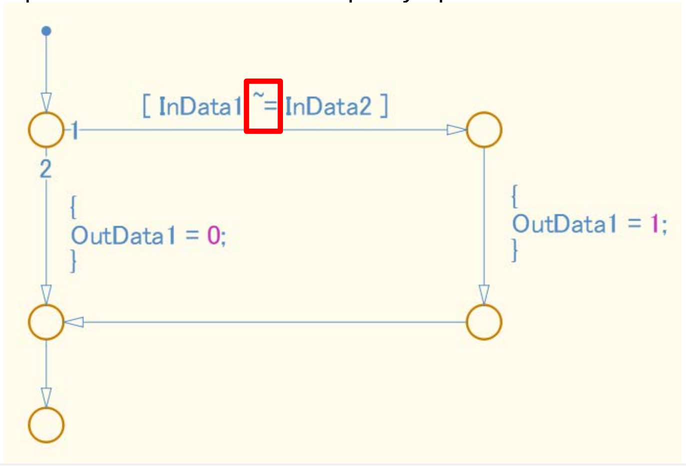
na_0001: Standard usage of Stateflow operators

Rule ID: Title	**na_0001: Standard usage of Stateflow operators**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b1/b2/b3, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

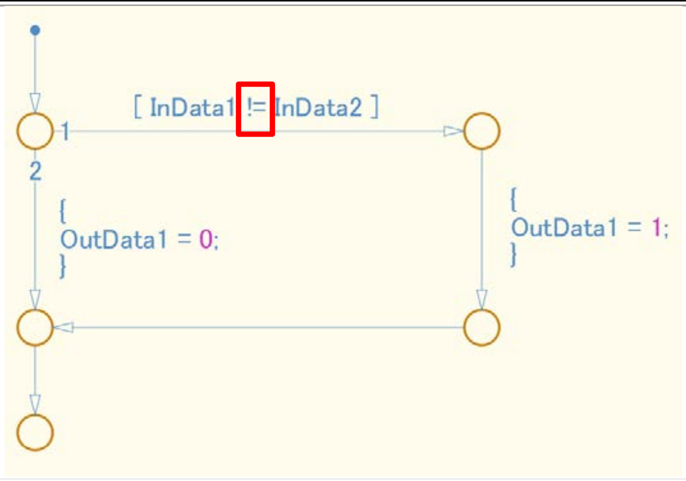
a	When [Chart] property {Action Language} is set to “C”, operators (“&”, “|”, “^”, “~”) shall be used only for bit operations.	-

	【Correct】
	Operators (“&”, “|”, “^”, “~”) are used for bit operations.
	
	`{`
	`OutData1 = InData1 | InData2;`
	`OutData2 = InData1 & InData2;`
	`OutData3 = InData1 ^ InData2;`
	`OutData4 = ~InData1;`
	`}`
	
	Name / DataType
	InData1 / int8
	InData2 / int8
	OutData1 / int8
	OutData2 / int8
	OutData3 / int8
	OutData4 / int8
	
	【Incorrect】
	Operators “&”, “|”, “^”, “~” are not used for bit operations.
	
	`{`
	`OutData1 = ( InData1 > 1 ) | ( InData2 < 2 );`
	`OutData2 = ( InData1 > 1 ) & ( InData2 < 2 );`
	`OutData3 = InData1 ^ InData2;`
	`OutData4 = ~( InData1 > 1);`
	`}`
	
	Name / DataType
	InData1 / double
	InData2 / double
	OutData1 / boolean
	OutData2 / boolean
	OutData3 / double
	OutData4 / boolean	

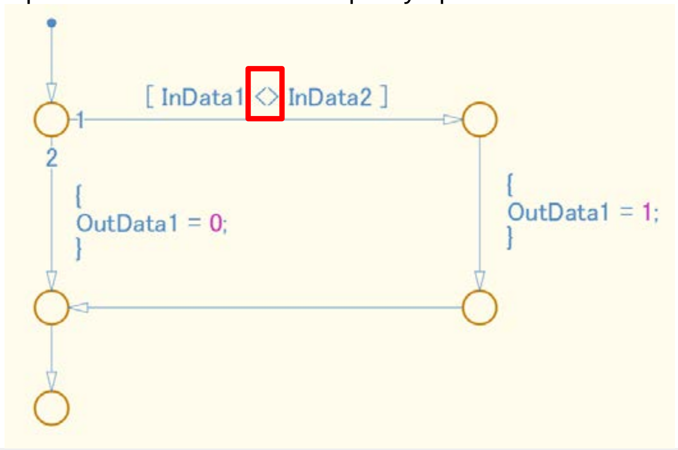
b1	When [Chart] property {Action Language} is set to “C”, operator “~=” shall be used for inequality operations.	-

	【Correct】
	Operator “~=” is used for inequality operations.
	
	`[ InData1 ~= InData2 ]`
	`1`
	`2`
	`{ OutData1 = 0; }`
	`{ OutData1 = 1; }`	

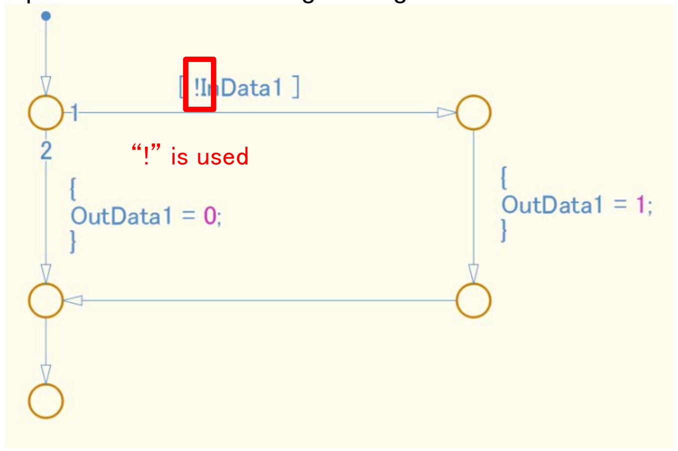
b2	When [Chart] property {Action Language} is set to “C”., operator “!=” shall be used for inequality operations.	-
	【Correct】 Operator “!=” is used for inequality operations.	

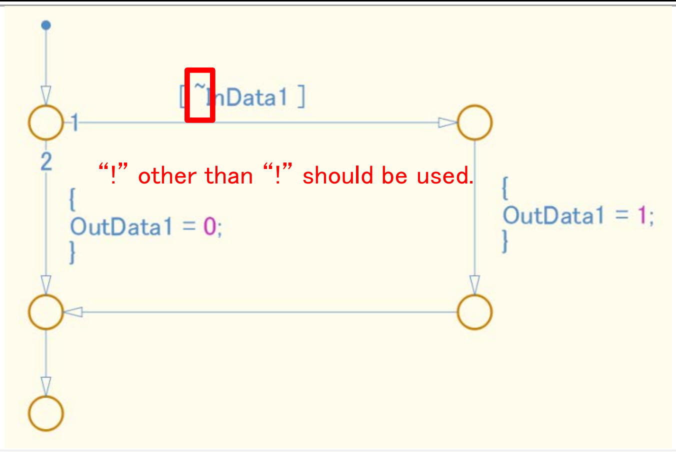
	 [InData1 != InData2]
	1
	2
	{
	OutData1 = 0;
	}
	{
	OutData1 = 1;
	}	

b3	When [Chart] property {Action Language} is set to “C”, operator “<>” shall be used for inequality operations.	-

	【Correct】
	Operator “<>” is used for inequality operations.
	 [InData1 <> InData2]
	1
	2
	{
	OutData1 = 0;
	}
	{
	OutData1 = 1;
	}	

c	When [Chart] property {Action Language} is set to “C”, operation “!” shall be used for logical negation.	-

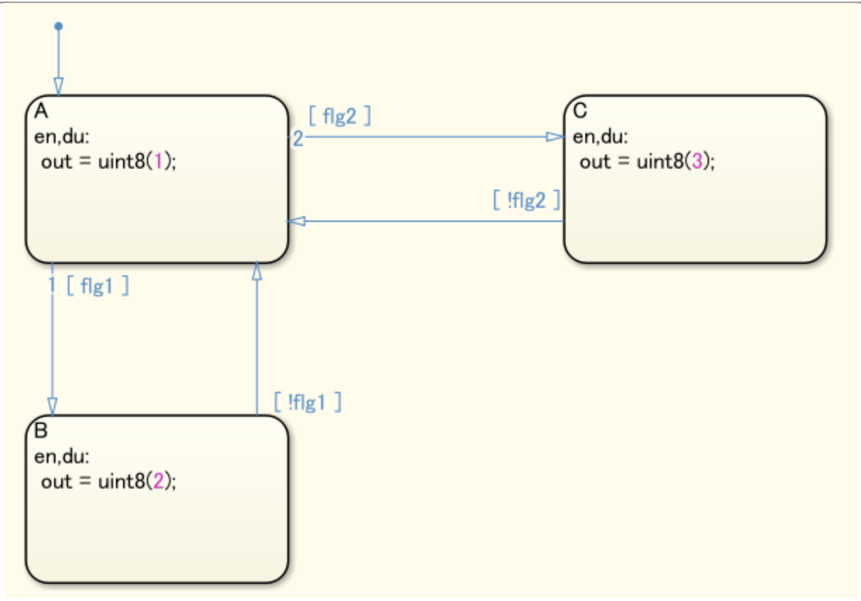
	【Correct】
	Operator “!” is used for logical negation.
	 [!InData1]
	1
	2
	“!” is used
	{
	OutData1 = 0;
	}
	{
	OutData1 = 1;
	}
	【Incorrect】
	An operator other than “!” should be used for logical negation.	

	 [~InData1]
	1
	2
	"!" other than "!" should be used.
	{
	OutData1 = 0;
	}
	{
	OutData1 = 1;
	}

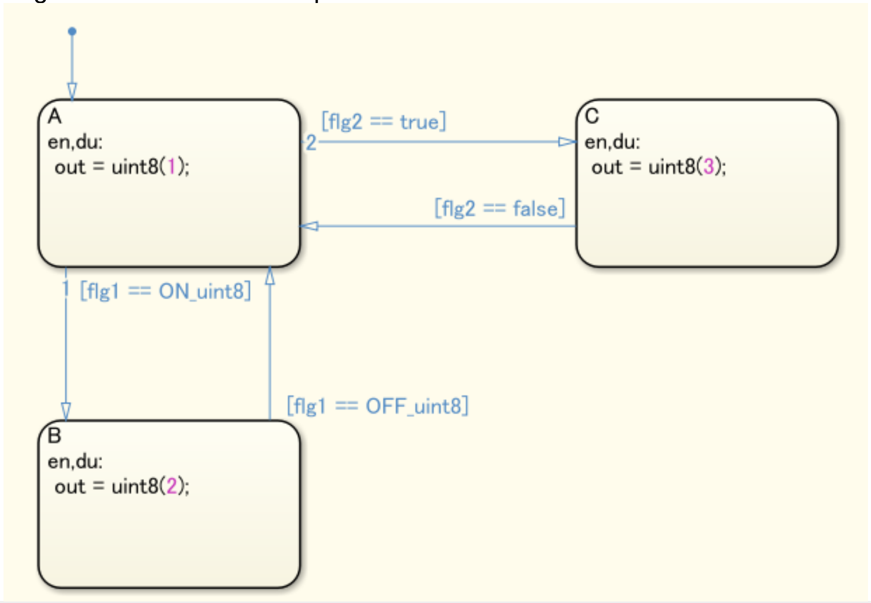
Rationale	
Sub ID	**Description**
a	When either of these [Chart] properties are set as follows: • {Action Language} is set to "MATLAB" • {Action Language} is set to "C" and {Enable C-bit operations} is selected, "&&" and "&", "||" and "|", have the same calculation function. However, when "&&" and "&" 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.
b1b2b3	· Consistent use of equality operators improves readability.
c	· Consistent use of logical negation operators improves readability. · When [Chart] property {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.

jc_0655: Prohibition of logical value comparison in Stateflow

Rule ID: Title	**jc_0655: Prohibition of logical value comparison in Stateflow**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Logical constants shall not be compared to each other.	-
	【Correct】 Logical constants are not compared to each other.	



【Incorrect】
Logical constants are compared to each other.



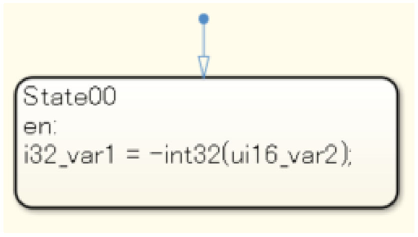
Rationale	
Sub ID	**Description**
a	・Readability improves with consistent use of “boolean-valued signal==true(boolean type constant)” or “(boolean-valued signal)” for logical signal condition expressions. ・Prevents redundancy in the model. ・Deviation from the rule can cause unexpected issues.

jc_0451: Use of unary minus on unsigned integers

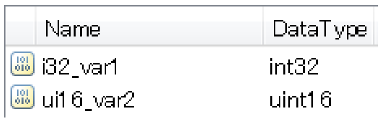
Rule ID: Title	**jc_0451: Use of unary minus on unsigned integers**
Sub ID Recommendations	NA-MAAB: a JMAAB: a
MATLAB® Version	All
Rule	

Sub ID	Description	Custom Parameter
a	Unary minus shall not be used on unsigned integers.	-

【Correct】

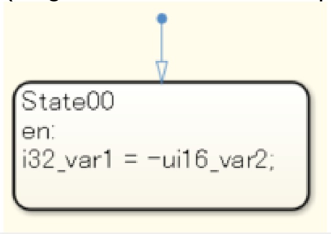


```
State00
en:
i32_var1 = -int32(ui16_var2);
```

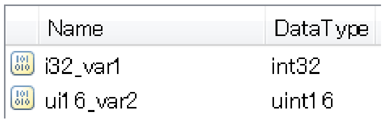


Name	DataType
i32_var1	int32
ui16_var2	uint16

【Incorrect】
Negative values cannot be input into 16-bit environments.
(Negative values can be input into 32-bit environments)



```
State00
en:
i32_var1 = -ui16_var2;
```



Name	DataType
i32_var1	int32
ui16_var2	uint16

Rationale

Sub ID	Description
a	・As the results are depend on the execution environment, unintended results can occur.

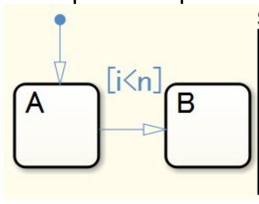
jc_0802: Prohibited use of implicit type casting in Stateflow

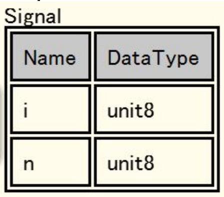
Rule ID: Title	**jc_0802: Prohibited use of implicit type casting in Stateflow**
Sub ID Recommendations	NA-MAAB: a JMAAB: a
MATLAB® Version	All

Rule

Sub ID	Description	Custom Parameter
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.	-

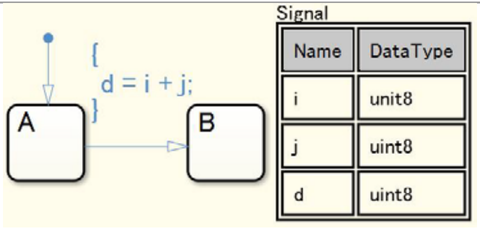
【Correct】
・Variables use the same data type for calculations.
Example: Comparison operation




Signal

Name	DataType
i	unit8
n	unit8

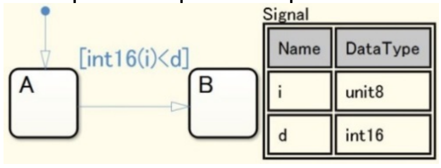
Example: Arithmetic operations and assignment operations (compound expressions)



```
{
 d = i + j;
}
```

Name	DataType
i	unit8
j	uint8
d	uint8

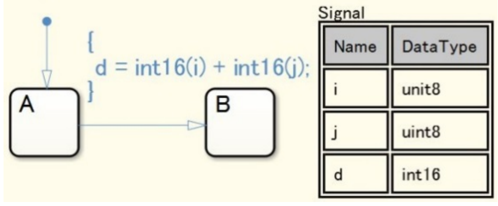
・Variables have different data types but are explicitly typecast before calculation.
Example: Comparison operation



```
[int16(i)<d]
```

Name	DataType
i	unit8
d	int16

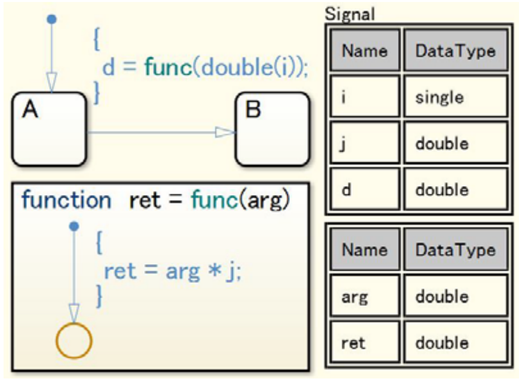
Example: Arithmetic operations and assignment operations (compound expressions)



```
{
 d = int16(i) + int16(j);
}
```

Name	DataType
i	unit8
j	uint8
d	int16

・The data type of actual arguments and formal arguments in the function call are the same.



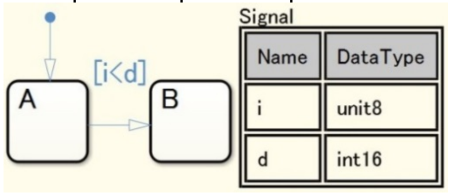
```
{
 d = func(double(i));
}
```

```
function  ret = func(arg)
 {
  ret = arg * j;
 }
```

Name	DataType
i	single
j	double
d	double

Name	DataType
arg	double
ret	double

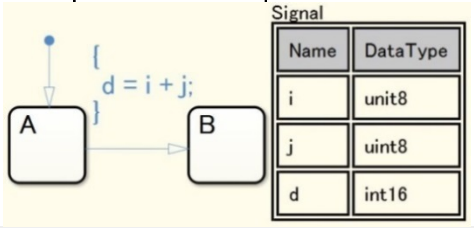
【Incorrect】
・Variables use different data types for calculations.
Example: Comparison operation



```
[i<d]
```

Name	DataType
i	unit8
d	int16

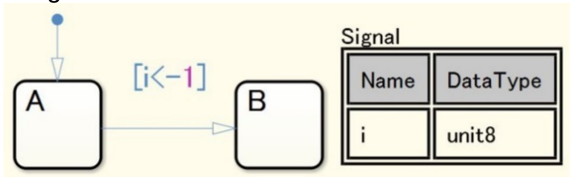
Example: Arithmetic operations and assignment operations (compound expressions)



```
{
 d = i + j;
}
```

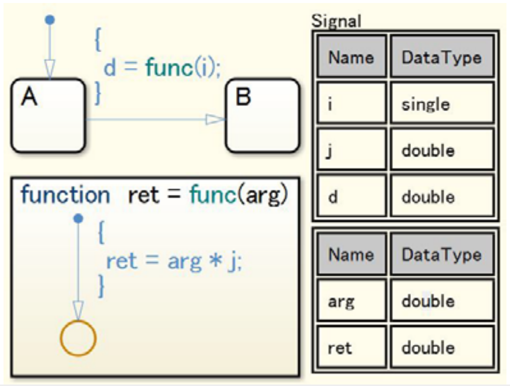
Name	DataType
i	unit8
j	uint8
d	int16

・Calculations are performed between unsigned integer type variables and signed integers.



Name	DataType
i	unit8

・The data type of actual arguments and formal arguments in the function call are different.



```
{
d = func(i);
}
```

```
function  ret = func(arg)
{
ret = arg * j;
}
```

Name	DataType
i	single
j	double
d	double

Name	DataType
arg	double
ret	double

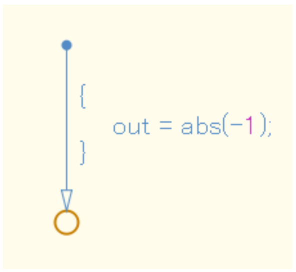
Rationale	
Sub ID	**Description**
a	・Implicit data type conversion can produce unexpected results.

jc_0803: Passing values to library functions

Rule ID: Title	**jc_0803: Passing values to library functions**
Sub ID Recommendations	NA-MAAB: 1/a2, b1/b2, c1/c2, d1/d2 JMAAB: 1/a2, b1/b2, c1/c2, d1/d2
MATLAB® Version	All

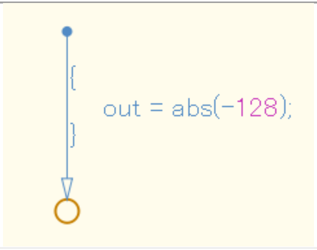
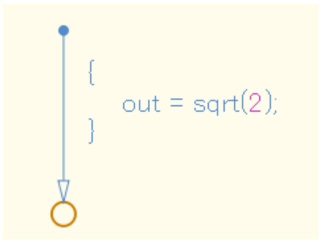
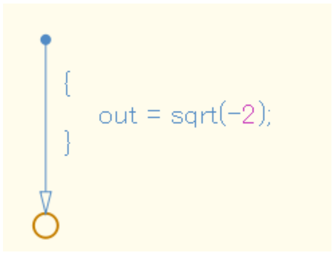
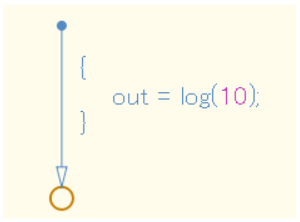
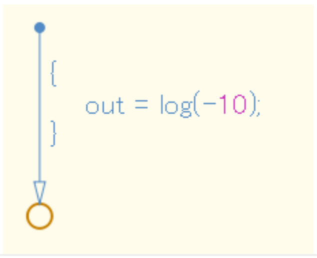
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	A minimum value for the signed integer type shall not be provided when using the abs library function.	-

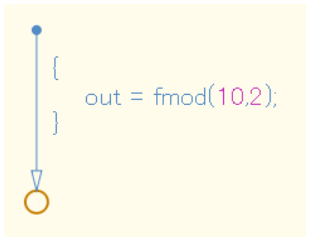
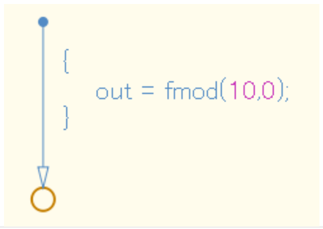
【Correct】



```
{
out = abs(-1);
}
```

【Incorrect】

	 `{ out = abs(−128); }`	
a2	The `abs` library function shall not be used.	-
b1	A negative number shall not be entered when using the `sqrt` library function.	-
	【Correct】  `{ out = sqrt(2); }` 【Incorrect】  `{ out = sqrt(−2); }`	
b2	The `sqrt` library function shall not be used.	-
c1	A negative number shall not be entered when using the `log` and `log10` library functions.	-
	【Correct】  `{ out = log(10); }` 【Incorrect】  `{ out = log(−10); }`	
c2	The `log` or `log10` library functions shall not be used.	-
d1	Zero shall not be entered for the second argument when using the `fmod` library function.	-

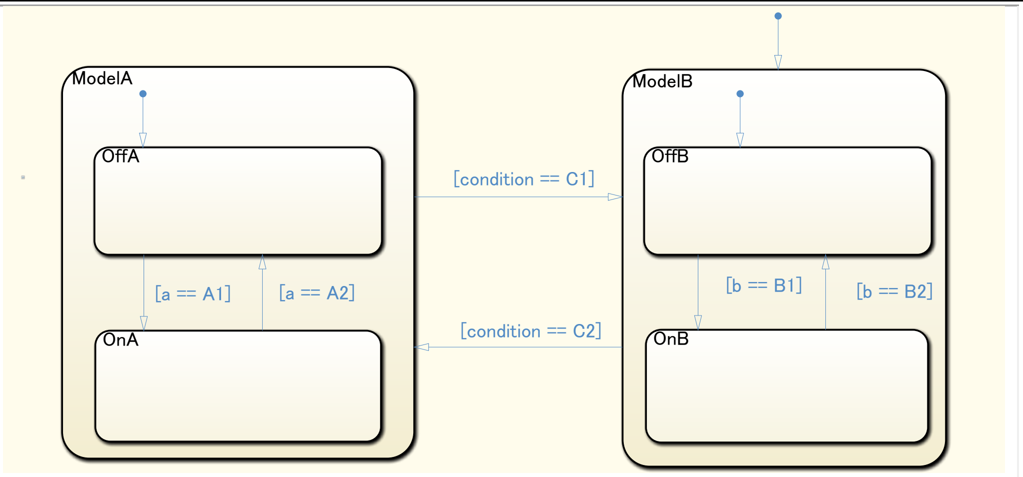
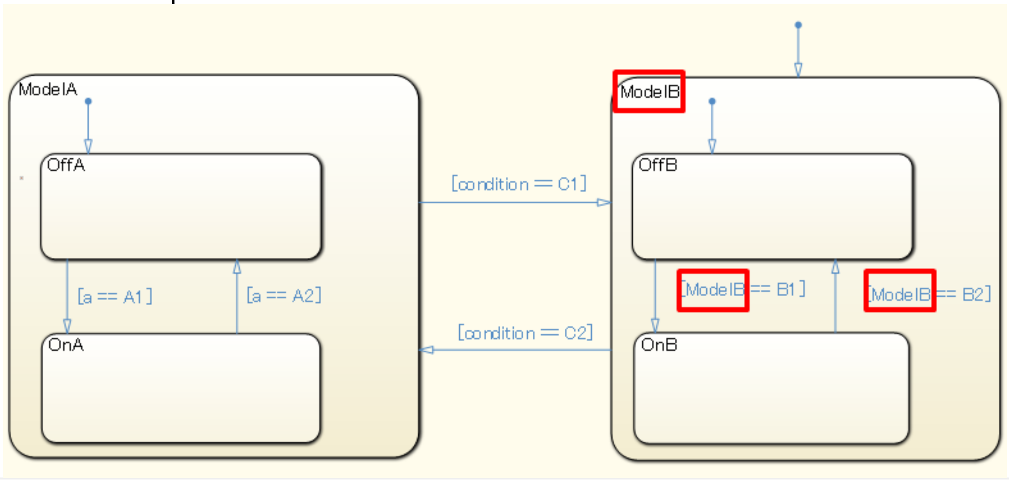
	【Correct】  【Incorrect】 	
d2	The `fmod` library function shall not be used.	-

Rationale	
Sub ID	**Description**
a1b1c1d1	・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.
a2b2 c2d2	・To avoid duplicate modelling of the same guard process in Simulink and Stateflow, use Simulink to perform arithmetic operations.

4.4. Label description

jc_0732: Distinction between state names, data names, and event names

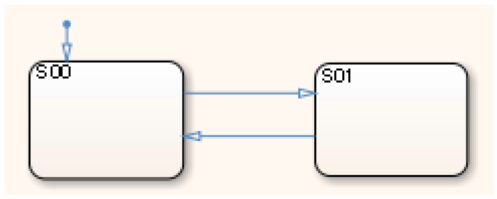
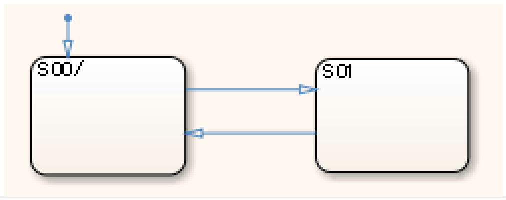
Rule ID: Title	**jc_0732: Distinction between state names, data names, and event names**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	An identical name shall not be used for state, data (inputs and outputs, local data, constants, parameters, data store memory), or event names in a single [Chart].	-
	【Correct】 Names are not duplicated.	

	 ModelA OffA [a == A1] [a == A2] OnA [condition == C1] [condition == C2] ModelB OffB [b == B1] [b == B2] OnB 【Incorrect】 Names are duplicated.  ModelA OffA [a == A1] [a == A2] OnA [condition == C1] [condition == C2] ModelB OffB [ModelB == B1] [ModelB == B2] OnB
Rationale	
Sub ID	**Description**
a	・Using unique names prevent misunderstanding.

jc_0730: Unique state name in Stateflow blocks

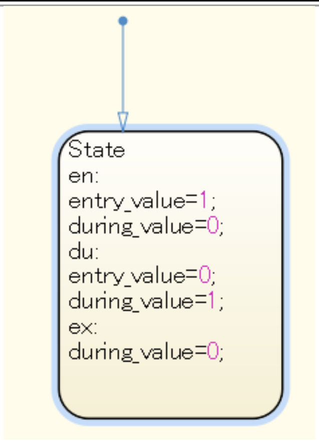
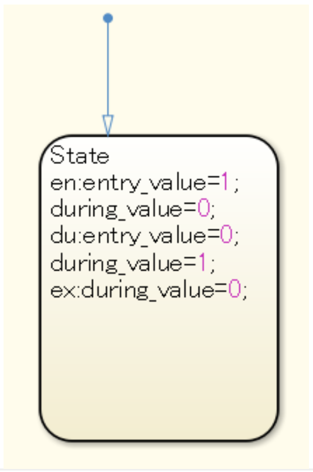
Rule ID: Title	**jc_0730: Unique state name in Stateflow blocks**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	State names in [Chart] shall be unique. The content of linked atomic sub-charts can be treated as another [Chart].	-
Rationale		
Sub ID	**Description**	
a	・Readability is impaired. ・Deviation from the rule can cause unintended code behavior.	

jc_0731: State name format

Rule ID: Title	**jc_0731: State name format**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The state name shall be followed by a new line that does not include a slash (/).	-
	【Correct】  【Incorrect】  	
Rationale		
Sub ID	**Description**	
a	・Readability improves when state names are described consistently.	

jc_0501: Line breaks in state labels

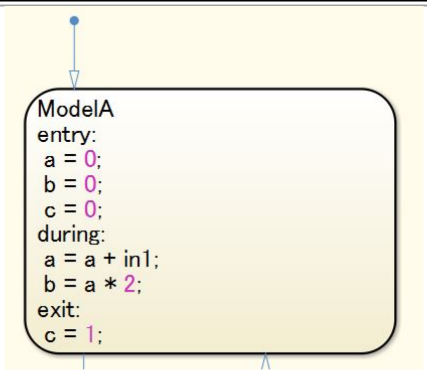
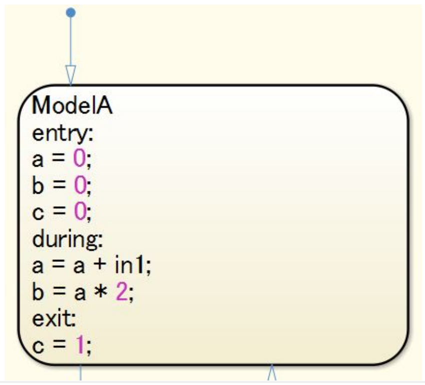
Rule ID: Title	**jc_0501: Line breaks in state labels**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	A state action statement shall not be written on the same line as a state action type.	-
	【Correct】	

	
	State
	en:
	entry_value=1;
	during_value=0;
	du:
	entry_value=0;
	during_value=1;
	ex:
	during_value=0;
	
	【Incorrect】
	
	State
	en:entry_value=1;
	during_value=0;
	du:entry_value=0;
	during_value=1;
	ex:during_value=0;

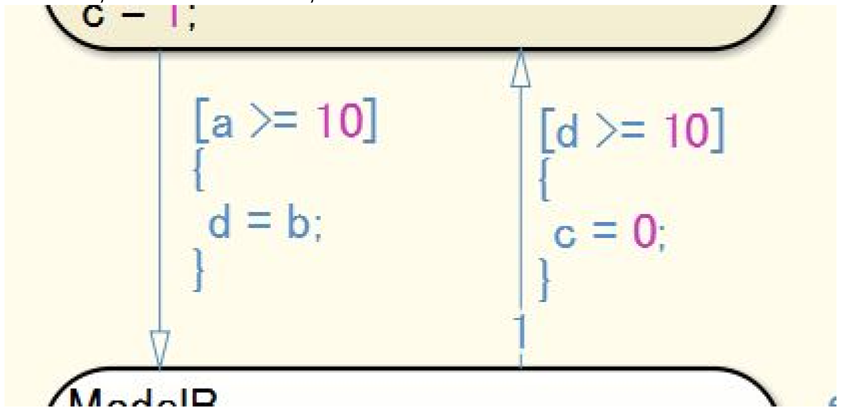
Rationale	
Sub ID	**Description**
a	· Readability is impaired.

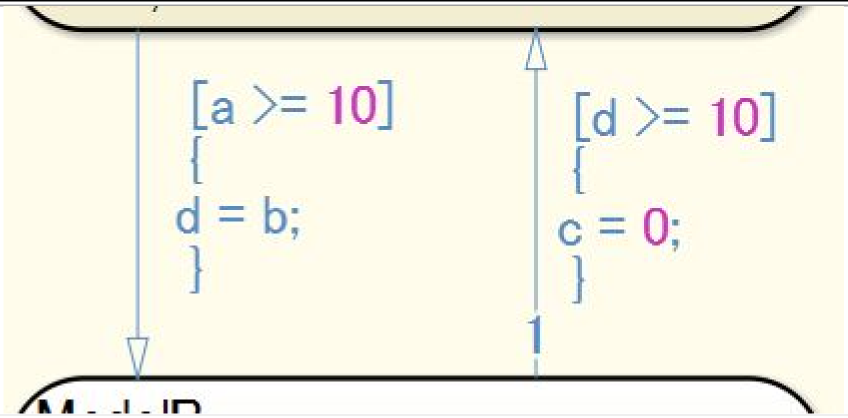
jc_0736: Uniform indentations in Stateflow blocks

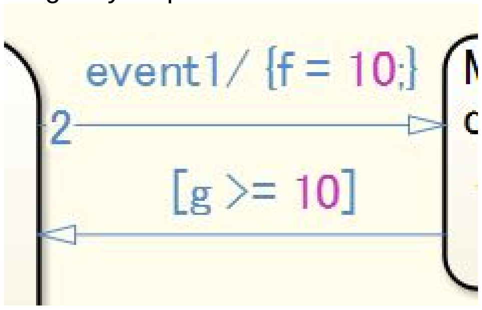
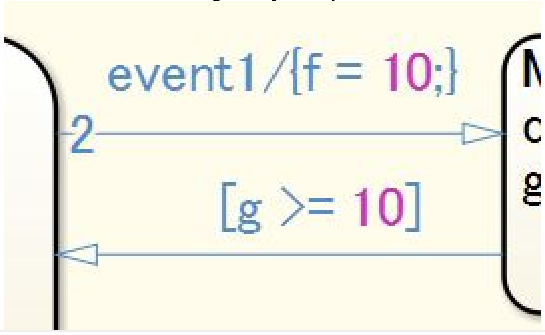
Rule ID: Title	**jc_0736: Uniform indentations in Stateflow blocks**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a, b, c	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	State action types shall not have blank spaces at the start of a line. Executable statements shall have one single-byte space at the start of the line.	Number of single-byte spaces
	【Correct】 Executable statements use one single-byte space at the start of the line.	

	
	ModelA
	entry:
	 a = 0;
	 b = 0;
	 c = 0;
	during:
	 a = a + in1;
	 b = a * 2;
	exit:
	 c = 1;
	
	【Incorrect】
	Executable statements do not have a single-byte space at the start of the line.
	
	ModelA
	entry:
	a = 0;
	b = 0;
	c = 0;
	during:
	a = a + in1;
	b = a * 2;
	exit:
	c = 1;	

b	A blank space shall not be entered before the following: • “[“ of a transition condition • “{“ of a condition action • “/” of a transition action	-

	【Correct】
	A blank space is not entered before the “[“ and “{“ of the transition label condition, condition action, and transition action.
	
	[a >= 10]
	{
	 d = b;
	}
	
	[d >= 10]
	{
	 c = 0;
	}
	
	【Incorrect】
	A blank space is entered before the “[“ and “{“ of the transition label condition, condition action, and transition action.	

	 [a >= 10]
	{
	d = b;
	}
	[d >= 10]
	{
	c = 0;
	}
	1	

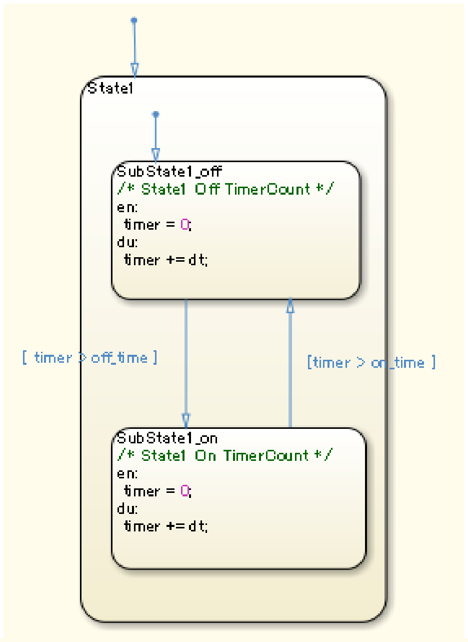
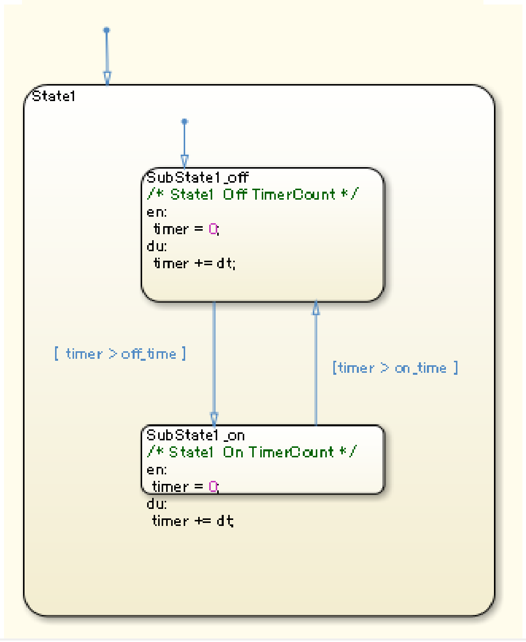
c	At least one single-byte space shall be entered after the “/” of a transition action.	Number of single-byte spaces
	【Correct】 Single-byte spaces are entered after the “/” of the transition action.  【Incorrect】 There are no single-byte spaces after the “/” of the transition action. 	

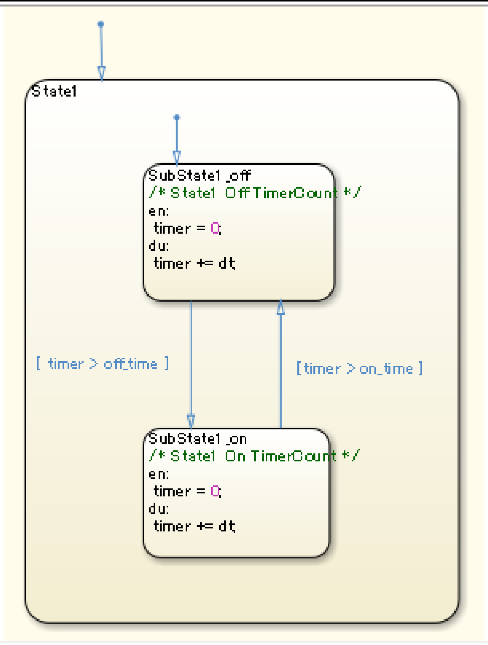
Rationale	
Sub ID	**Description**
a	・Using uniform indents before the executable statement clarifies the link between the state action type of a state label and the execution statement, improving readability.
b	・Using uniform indents for transition conditions, condition actions, and transition actions improves readability.
c	・Consistent use of blank spaces improves readability.

jc_0739: Describing text inside states

Rule ID: Title	**jc_0739: Describing text inside states**
Sub ID	NA-MAAB: a

Recommendations	JMAAB: a	
MATLAB$^{®}$ Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Text inside a state shall not extend beyond the boundaries of the state.	-

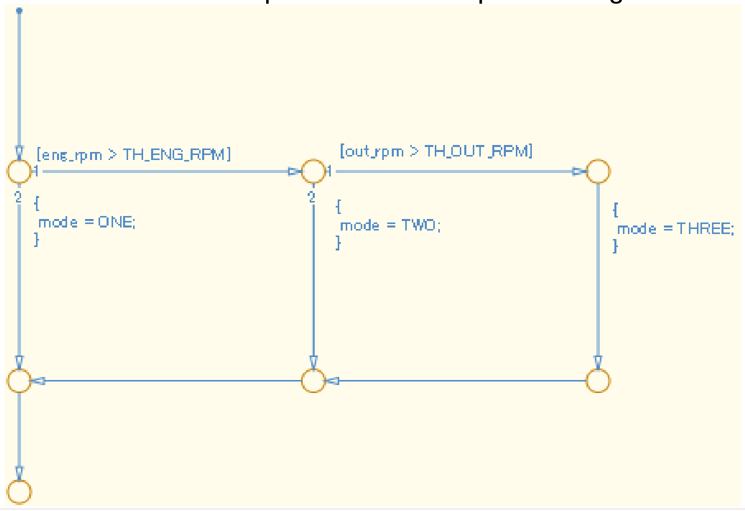
	【Incorrect】
	 State1
	SubState1_off
	/* State1 Off TimerCount */
	en:
	timer = 0;
	du:
	timer += dt;
	[timer > off_time]
	[timer > on_time]
	SubState1_on
	/* State1 On TimerCount */
	en:
	timer = 0;
	du:
	timer += dt;
	 State1
	SubState1_off
	/* State1 Off TimerCount */
	en:
	timer = 0;
	du:
	timer += dt;
	[timer > off_time]
	[timer > on_time]
	SubState1_on
	/* State1 On TimerCount */
	en:
	timer = 0;
	du:
	timer += dt;	

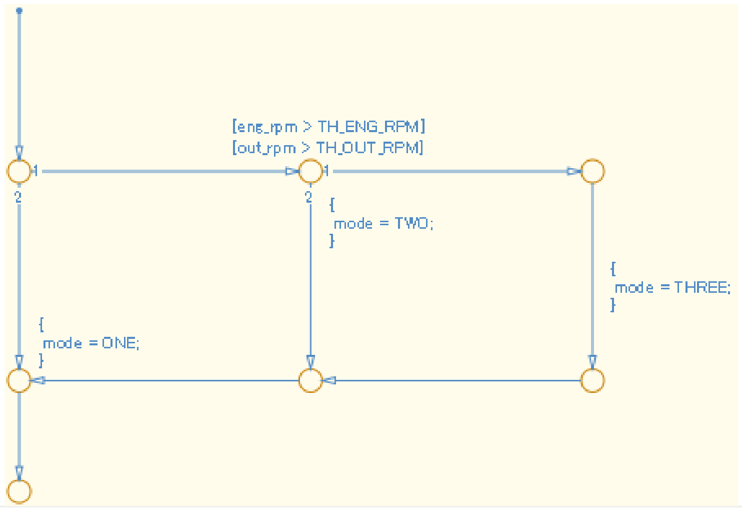
	 State1
	SubState1_off
	/* State1 OffTimerCount */
	en:
	timer = 0;
	du:
	timer += dt;
	[timer > off_time]
	[timer > on_time]
	SubState1_on
	/* State1 On TimerCount */
	en:
	timer = 0;
	du:
	timer += dt;

Rationale	
Sub ID	**Description**
a	・When the text inside a state extends beyond its boundaries, it can be difficult to determine which state the text belongs.

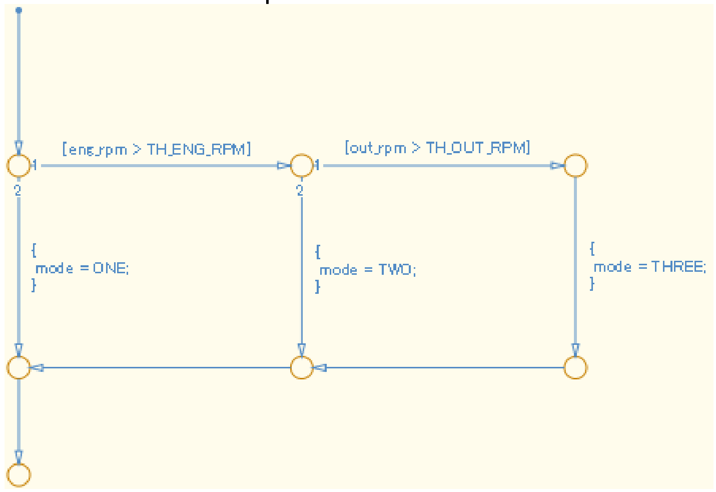
jc_0770: Position of transition label

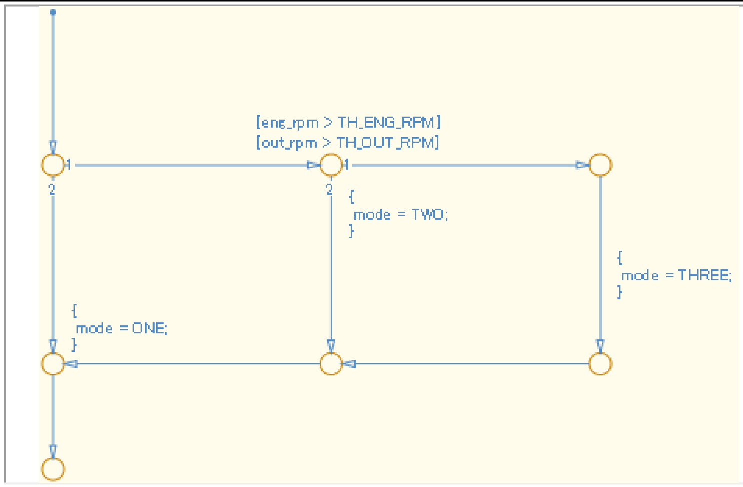
Rule ID: Title	**jc_0770: Position of transition label**		
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a1/a2		
MATLAB® Version	All		
Rule			
Sub ID	**Description**	**Custom Parameter**	
a1	Transition labels are positioned at the transition line point of origin.	-	

	【Correct】
	Transition labels are positioned at the point of origin.
	 [eng_rpm > TH_ENG_RPM]
	1
	2
	{
	mode = ONE;
	}
	[out_rpm > TH_OUT_RPM]
	1
	2
	{
	mode = TWO;
	}
	{
	mode = THREE;
	}		

	【Incorrect】
	The positioning of transition labels is inconsistent and do not correspond to the transition line.
	 [eng_rpm > TH_ENG_RPM]
	[out_rpm > TH_OUT_RPM]
	1
	2
	{
	mode = TWO;
	}
	{
	mode = THREE;
	}
	{
	mode = ONE;
	}	

a2	Transition labels are positioned near the center of the transition line.	-

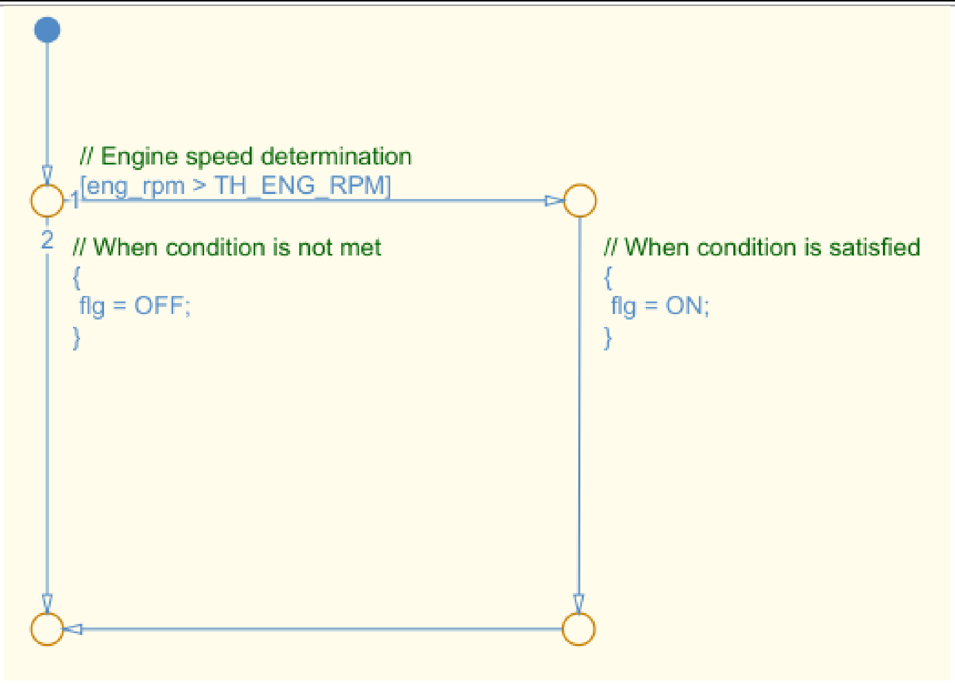
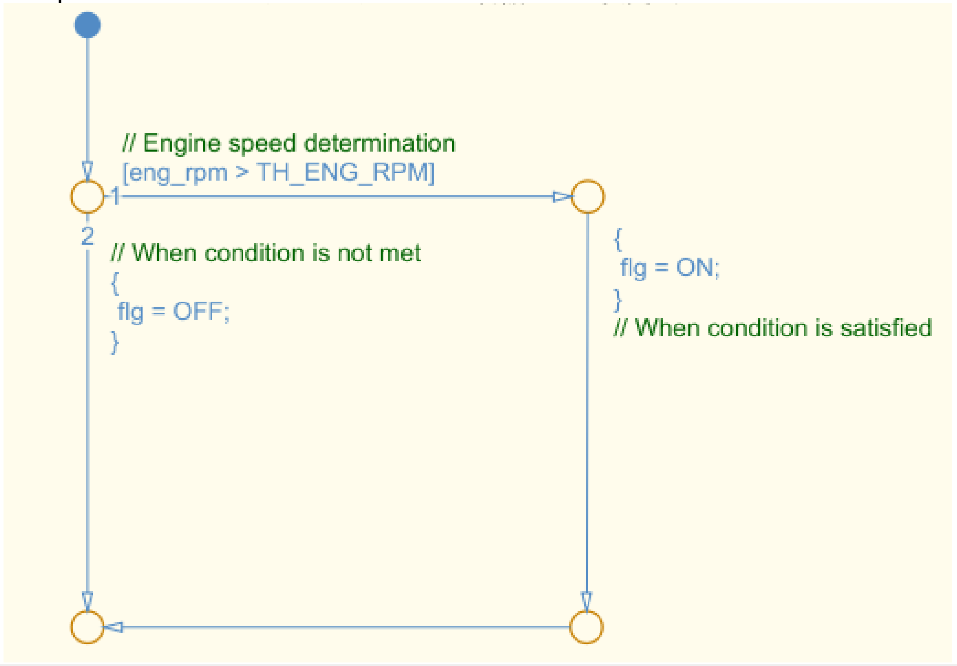
	【Correct】
	Transition labels are positioned near the center of the transition line.
	 [eng_rpm > TH_ENG_RPM]
	[out_rpm > TH_OUT_RPM]
	1
	2
	{
	mode = ONE;
	}
	{
	mode = TWO;
	}
	{
	mode = THREE;
	}
	【Incorrect】
	The positioning of transition labels is inconsistent and do not correspond to the transition line.	

	 [eng_rpm > TH_ENG_RPM]
	[out_rpm > TH_OUT_RPM]
	1
	2
	1
	2
	{
	mode = TWO;
	}
	{
	mode = THREE;
	}
	{
	mode = ONE;
	}

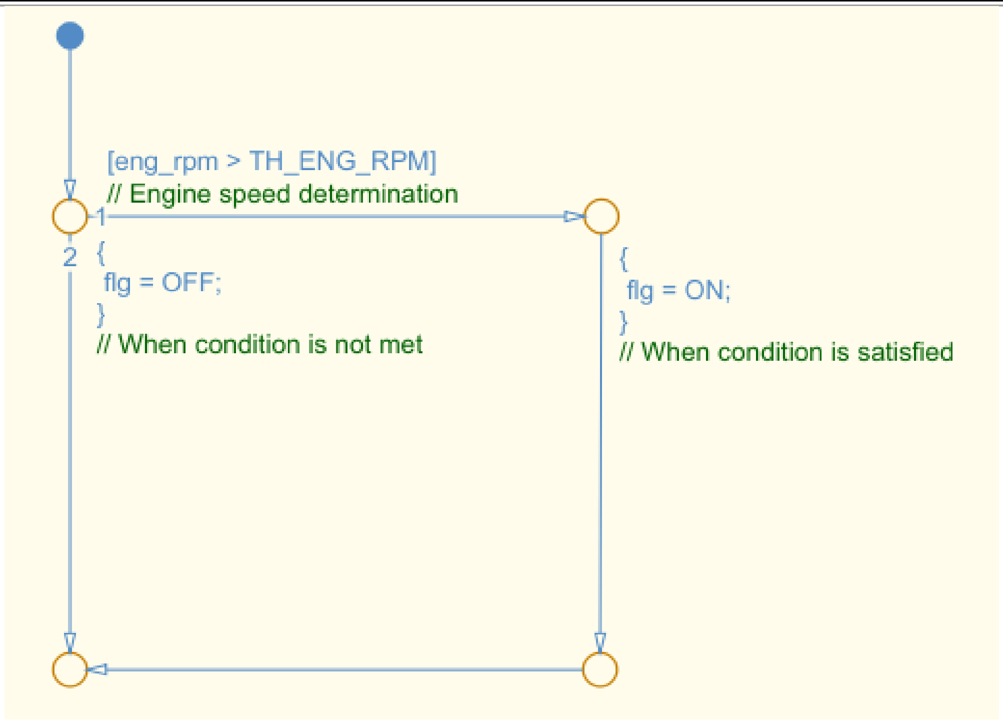
Rationale	
Sub ID	**Description**
a1a2	・Consistent positioning of transition labels makes the correspondence between label and line easier to understand.

jc_0771: Comment position in transition labels

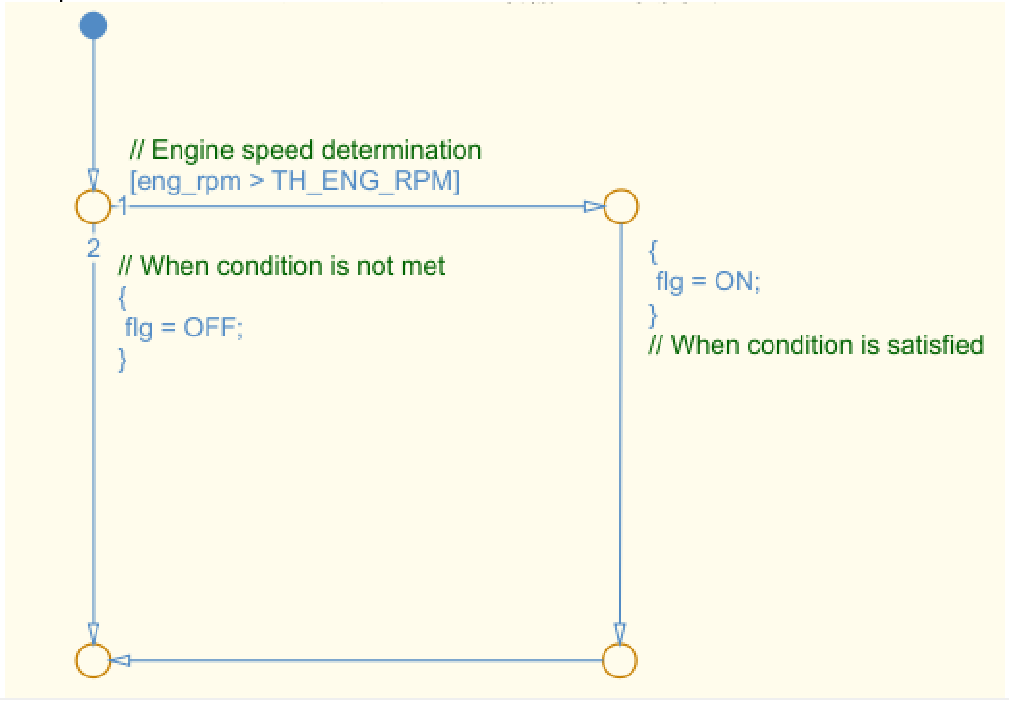
Rule ID: Title	**jc_0771: Comment position in transition labels**	
Sub ID Recommendations	NA-MAAB: a1/a2 JMAAB: a1/a2	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a1	Comments in transition labels shall be positioned above transition conditions, condition actions, transition actions, and Stateflow events.	-
	【Correct】 The position of the comments in the transition labels is uniform.	

	 // Engine speed determination
	[eng_rpm > TH_ENG_RPM]
	1
	2
	// When condition is not met
	{
	flg = OFF;
	}
	// When condition is satisfied
	{
	flg = ON;
	}
	【Incorrect】
	The position of the comments in the transition labels is inconsistent.
	 // Engine speed determination
	[eng_rpm > TH_ENG_RPM]
	1
	2
	// When condition is not met
	{
	flg = OFF;
	}
	{
	flg = ON;
	}
	// When condition is satisfied	

a2	Comments in transition labels shall be positioned below transition conditions, condition actions, transition actions, and Stateflow events.	-
	【Correct】 The position of the comments in the transition labels is uniform.	


[eng_rpm > TH_ENG_RPM]
// Engine speed determination
1
2
{
flg = OFF;
}
// When condition is not met
{
flg = ON;
}
// When condition is satisfied

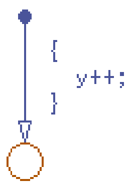
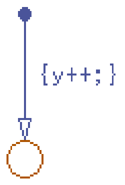
【Incorrect】
The position of the comments in the transition labels is inconsistent.


// Engine speed determination
[eng_rpm > TH_ENG_RPM]
1
2
// When condition is not met
{
flg = OFF;
}
{
flg = ON;
}
// When condition is satisfied

Rationale

Sub ID	**Description**
a1a2	・Uniform positioning of comments in transition labels clarifies to which transition condition, condition action, transition action, or Stateflow event the label corresponds.

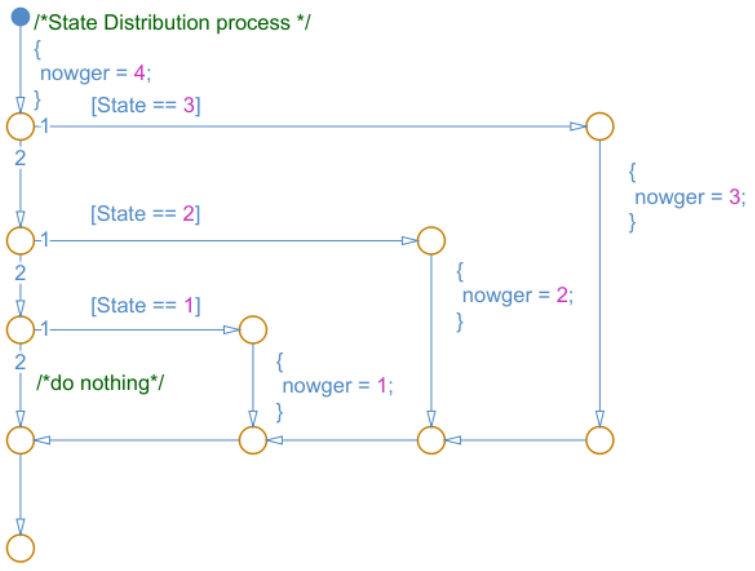
jc_0752: Condition action in transition label

Rule ID: Title	**jc_0752: Condition action in transition label**	
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Parentheses in condition actions shall use only curly brackets on a single line. (A new line shall start before and after curly brackets.)	-
	【Correct】 Note: The example is for a flow chart, but the rule also applies to state transitions.  Incorrect  	
Rationale		
Sub ID	**Description**	
a	· Clarifying condition actions improves readability.	

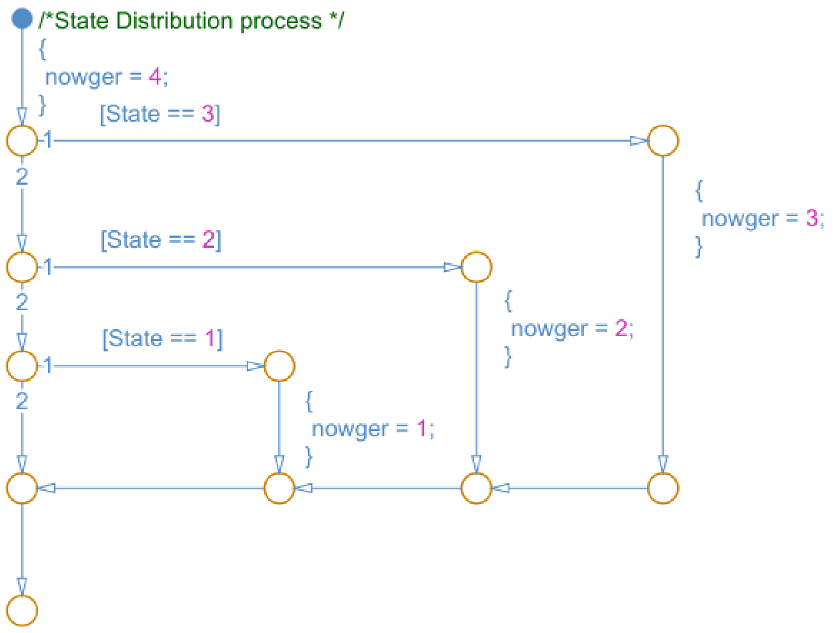
jc_0774: Comments for through transition

Rule ID: Title	**jc_0774: Comments for through transition**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When there is no processing in an unconditional transition, a clarifying comment shall be written on the transition label.	-
	【Correct】	

A clarifying comment is provided.



【Incorrect】
A clarifying comment is not provided on the condition path, so it is difficult to determine whether the lack of action is intentional.



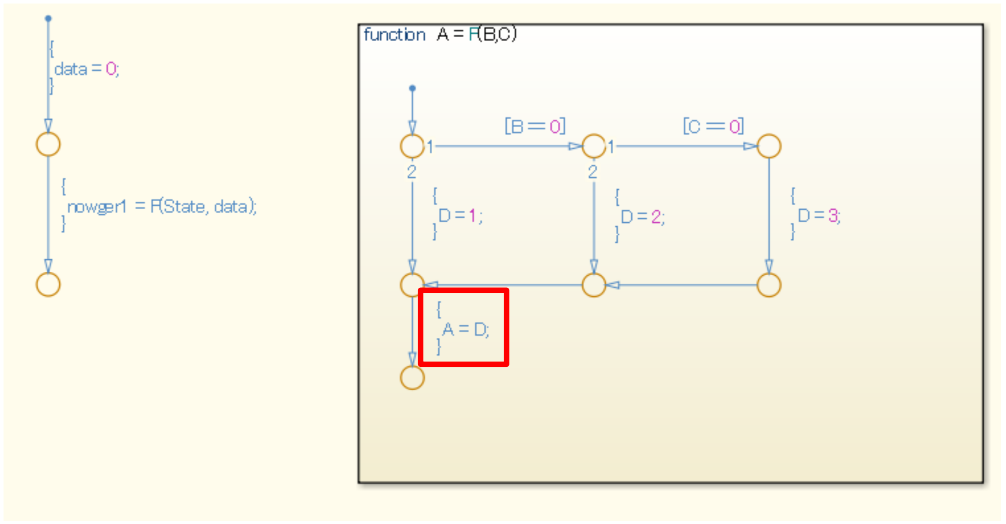
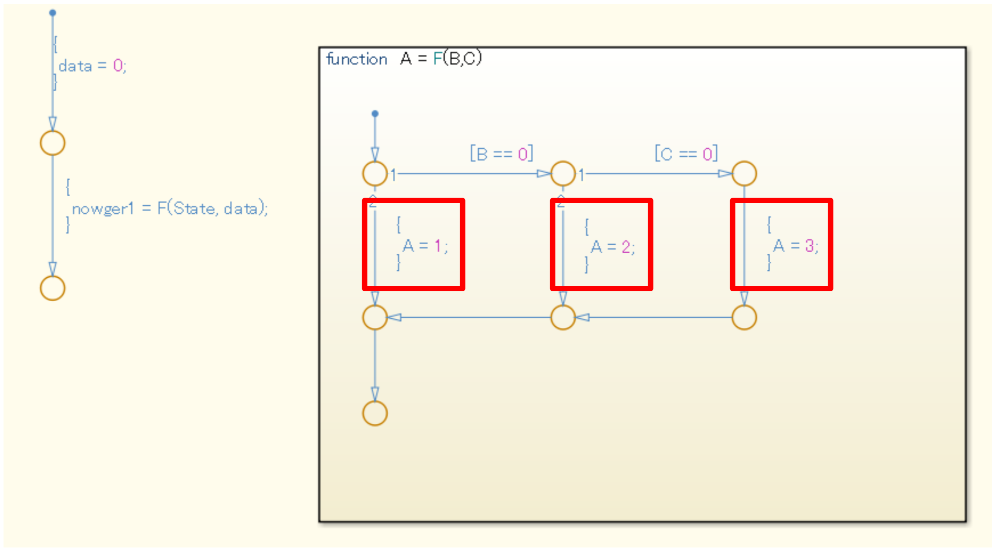
Rationale	
Sub ID	**Description**
a	・Clarifies that the processing is deliberately excluded. ・The comment that is added to a transition label is also included in the generated code.

4.5. Miscellaneous

jc_0511: Return values from a graphical function

Rule ID: Title	**jc_0511: Return values from a graphical function**

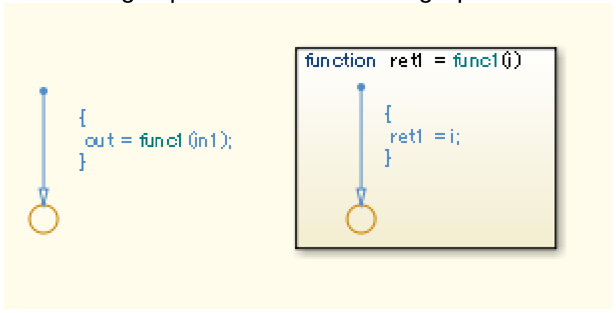
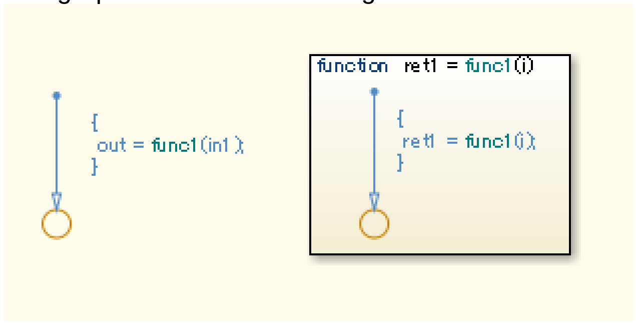
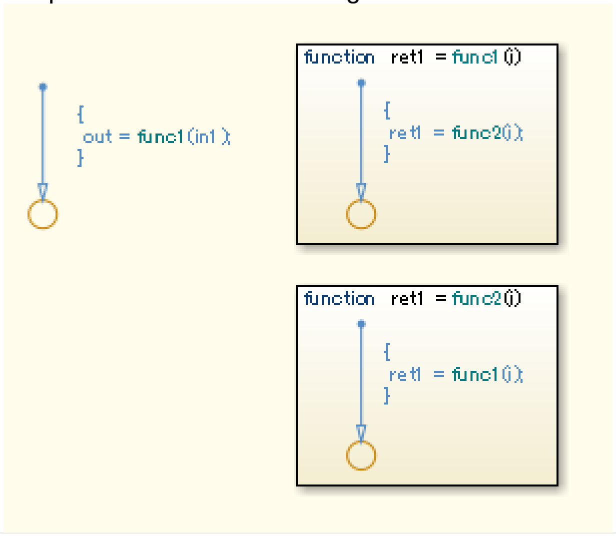
Sub ID Recommendations	NA-MAAB: No recommendations JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The return value for graphical functions shall be set in one place only.	-

	【Correct】  { data = 0; }
	{ nowger1 = F(State, data); }
	function A = F(B,C)
	[B == 0]
	[C == 0]
	1
	2
	{ D = 1; }
	{ D = 2; }
	{ D = 3; }
	{ A = D; }
	【Incorrect】  { data = 0; }
	{ nowger1 = F(State, data); }
	function A = F(B,C)
	[B == 0]
	[C == 0]
	1
	{ A = 1; }
	{ A = 2; }
	{ A = 3; }	

Rationale		
Sub ID	**Description**	
a	・Modifications to the output name is limited to prevent the changes from being missed or overlooked.	

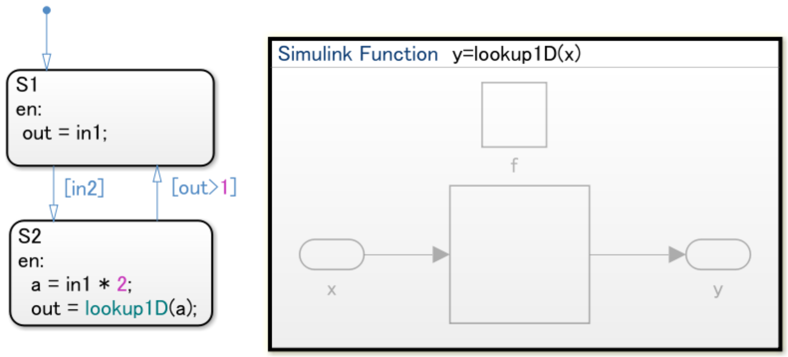
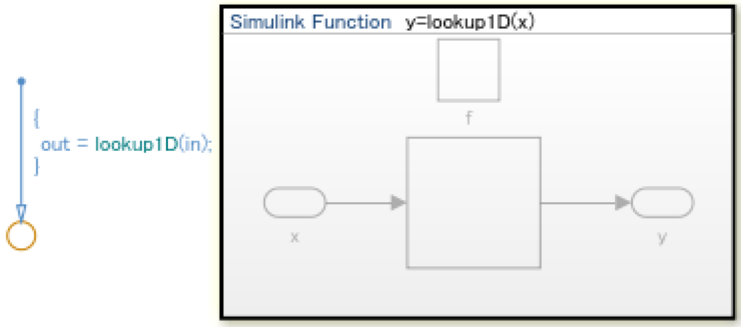
jc_0804: Prohibited use of recursive calls with graphical functions

Rule ID: Title	**jc_0804: Prohibited use of recursive calls with graphical functions**

Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Calls from a graphical function to itself and calls between graphical functions shall be prohibited.	-
	【Correct】 Processing is performed within the graphical function.  { out = func1(in1); } function ret1 = func1() { ret1 = i; } 【Incorrect】 The graphical function is calling itself.  { out = func1(in1); } function ret1 = func1() { ret1 = func1(); } Graphical functions are calling each other.  { out = func1(in1); } function ret1 = func1() { ret1 = func2(); } function ret1 = func2() { ret1 = func1(); }	
Rationale		
Sub ID	**Description**	
a	・Readability decreases. Deviation from the rule can cause unintended overflows and infinite loops.	

na_0042: Usage of Simulink functions

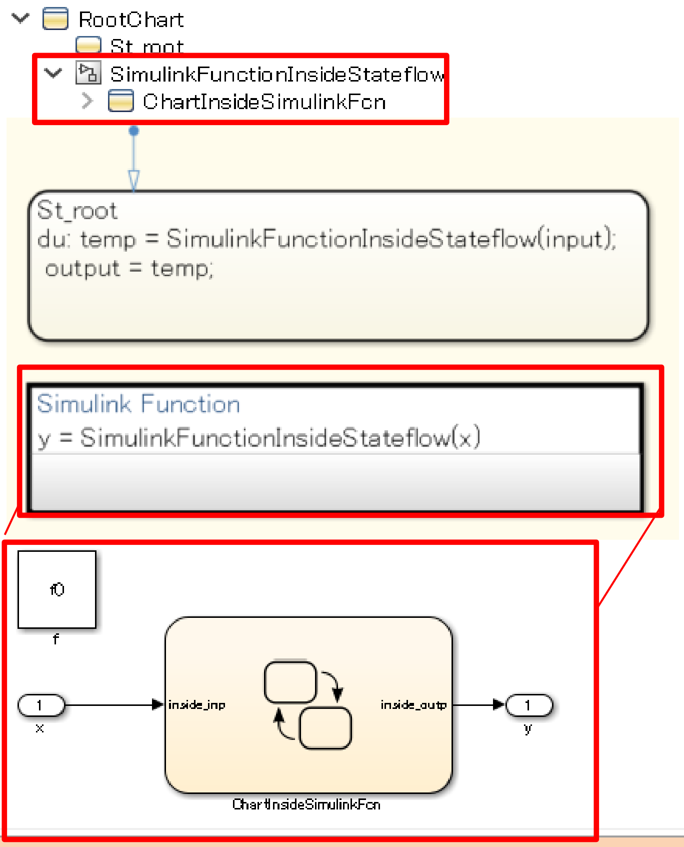
Rule ID: Title	**na_0042: Usage of Simulink functions**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	When using [Simulink Function] in [Chart], one or more of the following conditions shall be met. ・Input/output variables shall use only local [Chart] data in the [Simulink Function]. ・Input/output variables shall use only local [Chart] data and input data in the [Simulink Function]. ・ [Simulink Function] shall be called from multiple places in [Chart]. ・ [Simulink Function] shall not be called at every time step.	-

	【Correct】
	[Simulink Function] lookup1D is not called from every time step and, therefore, can be used.
	
	 S1
	en:
	out = in1;
	[in2]
	[out>1]
	S2
	en:
	a = in1 * 2;
	out = lookup1D(a);
	Simulink Function y=lookup1D(x)
	f
	x
	y
	【Incorrect】
	[Simulink Function] lookup1D is called from every time step and, therefore, cannot be used. (out is the Stateflow output data)
	
	 {
	out = lookup1D(in);
	}
	Simulink Function y=lookup1D(x)
	f
	x
	y	

Rationale		
Sub ID	**Description**	
a	・To improve model readability, the use of [Simulink Functions] should be used with caution in charts.	

na_0039: Limitation on Simulink functions in Chart blocks

Rule ID: Title	na_0039: Limitation on Simulink functions in Chart blocks	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Stateflow blocks shall not be used in [Simulink Functions] that are included in Stateflow [Chart].	-

	【Incorrect】 RootChart
	St_root
	SimulinkFunctionInsideStateflow
	ChartInsideSimulinkFcn
	St_root
	du: temp = SimulinkFunctionInsideStateflow(input);
	output = temp;
	Simulink Function
	y = SimulinkFunctionInsideStateflow(x)	

Rationale		
Sub ID	**Description**	
a	・Readability decreases and can result in design errors.	

5. MATLAB

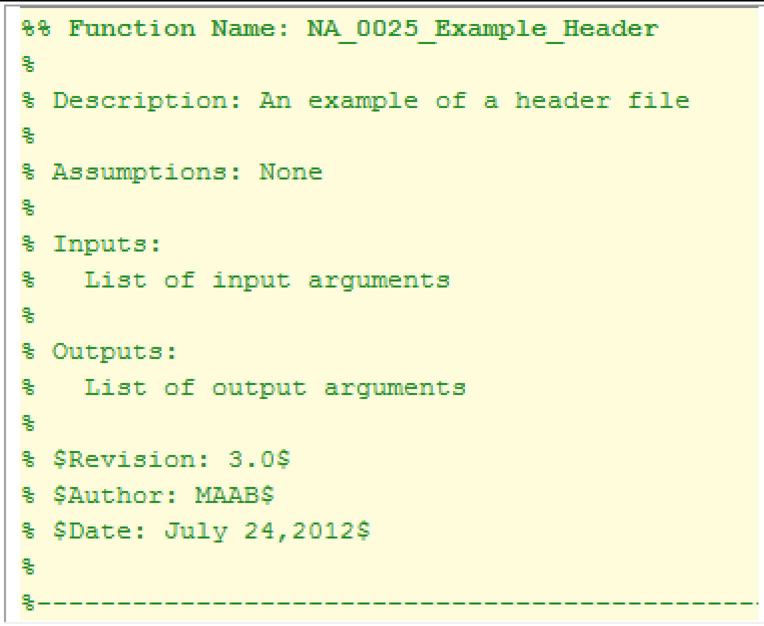
5.1. MATLAB Appearance

na_0018: Number of nested if/else and case statements

Rule ID: Title	**na_0018: Number of nested if/else and case statements**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The number of levels of nested if /else and case statements shall be limited, typically to three levels.	Maximum nested levels
Rationale		
Sub ID	**Description**	
a	· Improves readability · Code generation may not be possible.	

na_0025: MATLAB Function headers

Rule ID: Title	**na_0025: MATLAB Function headers**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	[MATLAB Functions] 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 Example:	-

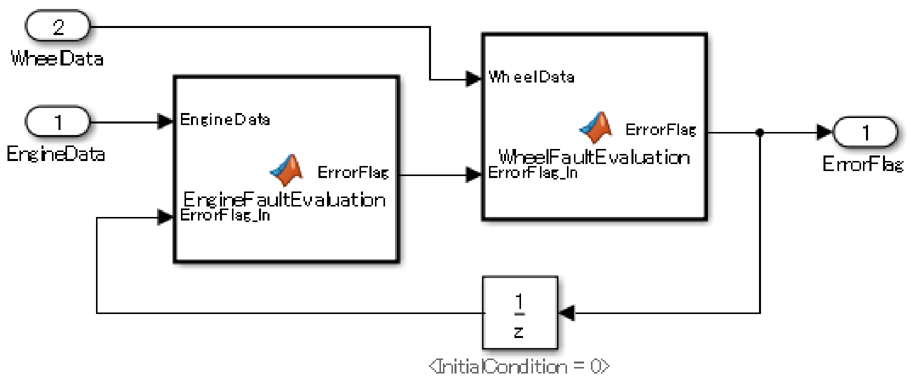


```
%% Function Name: NA_0025_Example_Header
%
% Description: An example of a header file
%
% Assumptions: None
%
% Inputs:
%   List of input arguments
%
% Outputs:
%   List of output arguments
%
% $Revision: 3.0$
% $Author: MAAB$
% $Date: July 24,2012$
%
%------------------------------------------------
```

Rationale	
Sub ID	**Description**
a	・Improves readability, model simulation, testability, and workflow ・Code generation may not be possible.

5.2. MATLAB Data and Operations

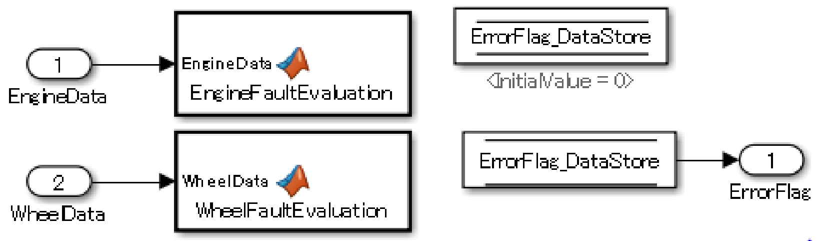
na_0024: Shared data in MATLAB functions

Rule ID: Title	**na_0024: Shared data in MATLAB functions**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Signal lines shall be used to connect data that is shared between MATLAB functions.	-
	【Correct】 	

```
function ErrorFlag =
EngineFaultEvaluation(EngineData,ErrorFlag_In)
%#codegen
    RPM_HIGH = 10000;
    RPM_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
```

【Incorrect】
This type of pattern cannot be used when the rule is applied.



```
function EngineFaultEvaluation(EngineData)
%#codegen
    global ErrorFlag_DataStore
    RPM_HIGH = 10000;
    RPM_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);
```

	```
	function WheelFaultEvaluation(WheelData)
	%#codegen
	    global ErrorFlag_DataStore
	    SLIP_HIGH = 1000;
	    WHEELSLIP = 2^3;
	    if WheelData > SLIP_HIGH
	        ErrorFlag_DataStore =
	bitor(ErrorFlag_DataStore,WHEELSLIP);
	    end
	```
Rationale	
Sub ID	**Description**
a	・When a data store is used, the readability of the data flow decreases and can lead to errors in the update reference timing.

na_0031: Definition of default enumerated value

Rule ID: Title	**na_0031: Definition of default enumerated value**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Method `getDefaultValue` shall be used to explicitly define the default value of an enumeration.	-

	【Correct】
	```
	classdef(Enumeration) BasicColors < Simulink.IntEnumType
	    enumeration
	        Red(0)
	        Yellow(1)
	        Blue(2)
	    end
	    methods(Static = true)
	        function retVal = getDefaultValue()
	            retVal = BasicColors.Red;
	        end
	    end
	end
	```
	【Incorrect】
	```
	classdef(Enumeration) BasicColors < Simulink.IntEnumType
	    enumeration
	        Red(0)
	        Yellow(1)
	        Blue(2)
	    end
	end
	```

Rationale		
Sub ID	**Description**	

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.

na_0034: MATLAB Function block input/output settings

Rule ID: Title	**na_0034: MATLAB Function block input/output settings**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The data type in the model explorer shall be defined for all input and output to [MATLAB Function].	-
Rationale		
Sub ID	**Description**	
a	・Defining the data type for all input and output to [MATLAB Function] helps prevent simulation errors and unexpected behavior.	

5.3. MATLAB Usage

na_0016: Source lines of MATALAB Functions

Rule ID: Title	**na_0016: Source lines of MATALAB Functions**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	The length of MATLAB functions shall be limited. This restriction applies to MATLAB Functions that reside in the Simulink block diagram and external MATLAB files with a `.m` extension. The recommended limit is 60 lines of code. Subfunctions may use an additional 60 lines of code.	Maximum effective lines of code per function
Rationale		
Sub ID	**Description**	
a	・Improves readability and workflow ・Code generation may not be possible.	

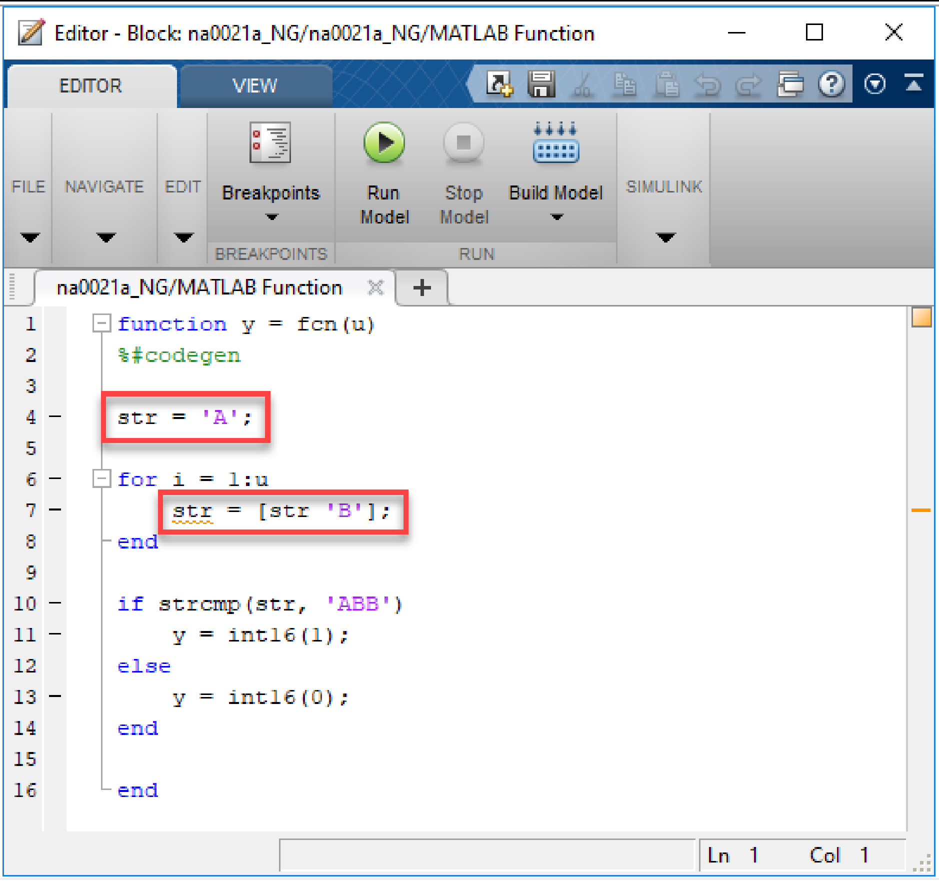
na_0017: Number of called function levels

Rule ID: Title	**na_0017: Number of called function levels**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**

a	The number of sub-function levels shall be limited, typically to three levels. MATLAB function blocks that resides at 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 MATLAB block and those in separate `.m` files. Standard utility functions, such as built in functions like sqrt or log, are not included in the number of levels. Likewise, commonly used custom utility functions can be excluded from the number of levels.	Maximum function call levels
Rationale		
Sub ID	**Description**	
a	・Improves readability and testability	

na_0021: Strings in MATLAB functions

Rule ID: Title	**na_0021: Strings in MATLAB functions**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Assignment statements for strings shall not be used in MATLAB functions.	-
	【Incorrect】 An assignment statement for strings is being used in the MATLAB function.	

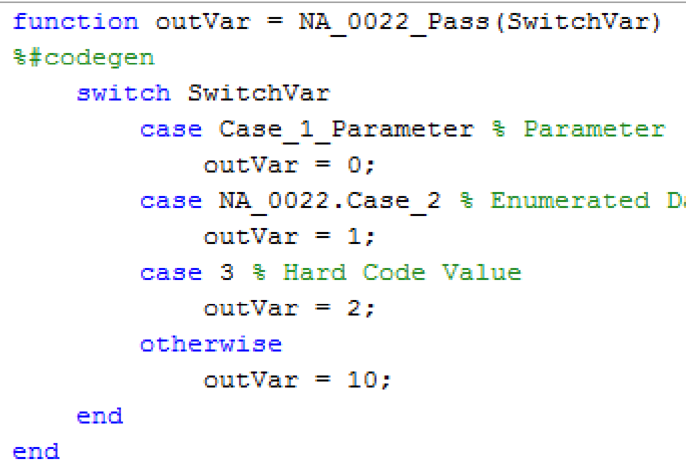


```
function y = fcn(u)
%#codegen

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	**Description**
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 [Switch Case])

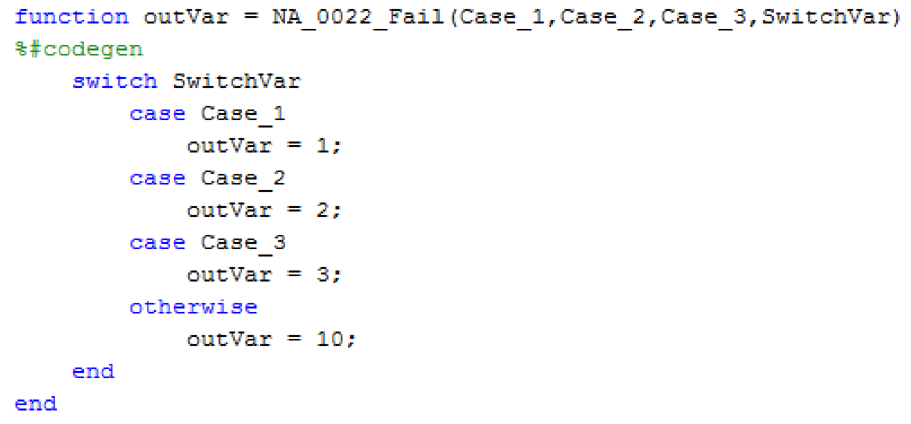
na_0022: Recommended patters for Switch/Case statements

Rule ID: Title	**na_0022: Recommended patters for Switch/Case statements**	
Sub ID Recommendations	NA-MAAB: a JMAAB: Not supported	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	Switch / Case statements shall use constant values for the “Case” arguments. Input variables shall not be used in the “Case” arguments. 【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 = 1;
        case 3 % Hard Code Value
            outVar = 2;
        otherwise
            outVar = 10;
    end
end
```

【Incorrect】

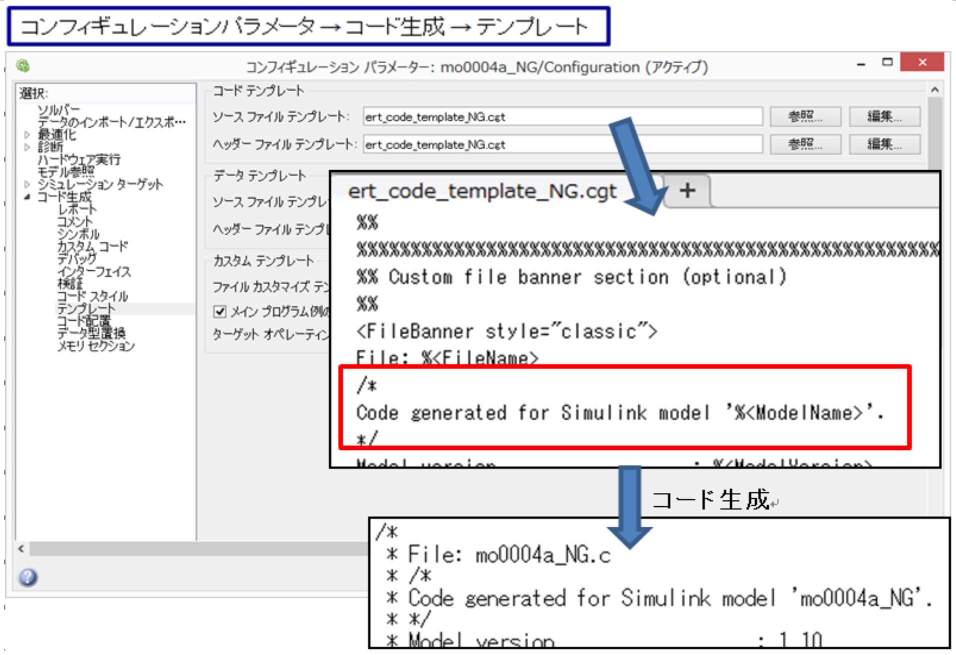


```
function outVar = NA_0022_Fail(Case_1,Case_2,Case_3,SwitchVar)
%#codegen
    switch SwitchVar
        case Case_1
            outVar = 1;
        case Case_2
            outVar = 2;
        case Case_3
            outVar = 3;
        otherwise
            outVar = 10;
    end
end
```

Rationale	
Sub ID	**Description**
a	・Improves model simulation and testability. ・Code generation may not be possible.

jc_0801: Prohibited use of the /* and */ comment symbols

Rule ID: Title	**jc_0801: Prohibited use of the /* and */ comment symbols**	
Sub ID Recommendations	NA-MAAB: a JMAAB: a	
MATLAB® Version	All	
Rule		
Sub ID	**Description**	**Custom Parameter**
a	As comment symbols /* and */ are automatically assigned in the generated code, the symbol shall not be used in: ・`cgt` file ・`mpt` Signal description ・`mpt` parameter description	-
	【Incorrect】 In a cgt file.	



ソース ファイル テンプレート: ert_code_template_NG.cgt
ヘッダー ファイル テンプレート: ert_code_template_NG.cgt
☑ メイン プログラム例の生成

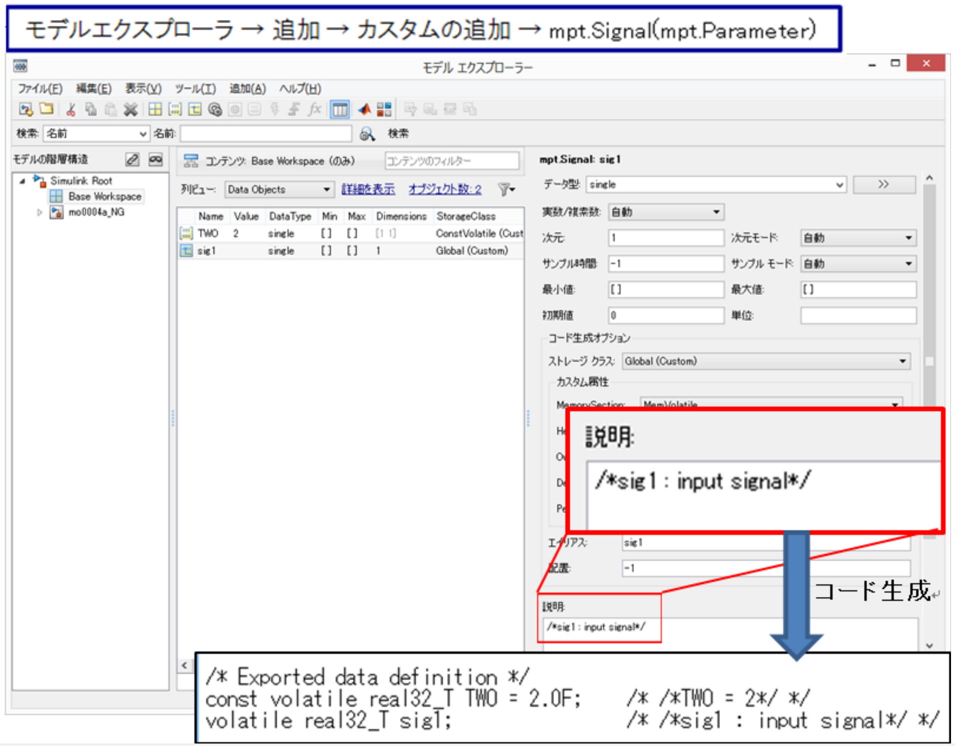
ert_code_template_NG.cgt

```
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Custom file banner section (optional)
%%
<FileBanner style="classic">
File: %<FileName>
/*
Code generated for Simulink model '%<ModelName>'.
*/
Model version                  : %<ModelVersion>
```

```
/*
 * File: mo0004a_NG.c
 * /*
 * Code generated for Simulink model 'mo0004a_NG'.
 * */
 * Model version                  : 1.10
```

【Incorrect】

mpt Signal description (the same also applies to `mpt.Parameter`).



Name	Value	DataType	Min	Max	Dimensions	StorageClass
TWO	2	single	[]	[]	[1 1]	ConstVolatile (Cust
sig1		single	[]	[]	1	Global (Custom)

mpt.Signal: sig1

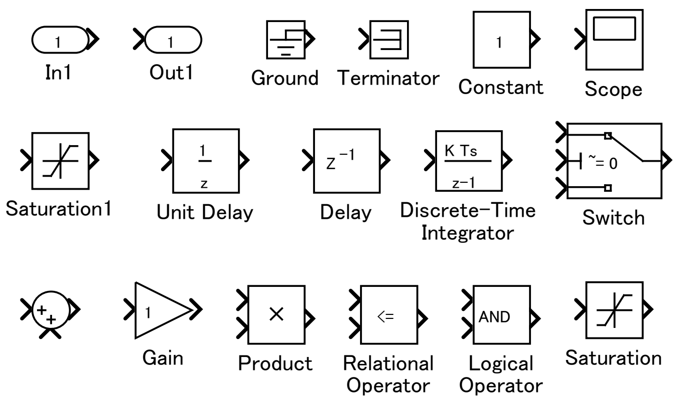
データ型: single
実数/複素数: 自動
次元: 1　次元モード: 自動
サンプル時間: -1　サンプル モード: 自動
最小値: []　最大値: []
初期値: 0　単位:
ストレージ クラス: Global (Custom)
MemorySection: MemVolatile
エイリアス: sig1
配置: -1
説明: /*sig1 : input signal*/

```
/* Exported data definition */
const volatile real32_T TWO = 2.0F;      /* /*TWO = 2*/ */
volatile real32_T sig1;                  /* /*sig1 : input signal*/ */
```

0011 Rationale	
Sub ID	**Description**
a	Since comment symbols /* and */ are automatically assigned in the generated code, comments can be unintentionally nested and behave differently than expected.

6. Glossary

This section provides clarification of terms that are used in the guidelines.

Terms	Definition
Parameters	When modifications have not been made, this term refers to constants that are defined in the base workspace/model workspace.
Built-in MATLAB functions	MATLAB functions and scripts.
Reserved MATLAB words	
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.
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:  In1 Out1 Ground Terminator Constant Scope Saturation1 Unit Delay Delay Discrete-Time Integrator Switch Gain Product Relational Operator Logical Operator Saturation
Structural subsystem	[Subsystems], [models], [charts], and [MATLAB functions] are frameworks for defining the structure, blocks with user-defined internal processing.
Subsystem	A subsystem that can be internally modeled by using Simulink basic projection. Even if {BlockType} is "subsystem", [Chart], [MATLAB Function], etc. blocks that describe the inside (other than Simulink basic projection) are not included. [Model] is not included.
Conditional subsystem	A subsystem with conditional input ports.
Atomic subsystem	A {BlockType} that is a “subsystem” and executes the structural subsystem as a single unit. Conditional subsystems, [Chart], and [MATLAB Function] are considered atomic subsystems.
Port label name	The input/output port labels of a structural subsystem. The names of [Inport] and [Outport] blocks are placed in a subsystem by default. Names of Stateflow input/output data are displayed by default. The display option can be changed when masking a subsystem.

Conditional input block	Includes [Trigger], [Enable], [Function Call], and [Reset].
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]. (As of R201b at later) [Delay] blocks can also be used 2. A block that retains past values. Uses [Unit Delay], [Memory], [Delay], and [Tapped Delay].
Calculation block	Blocks with “Sum” {BlockType} that carry out addition and subtraction operations. Includes [Sum], [Add], [Subtract], and [Sum of Elements].
Multiplication and division block	Blocks with “Product” {BlockType} that carry out division and multiplication operations. Includes [Product], [Divide] and [Product of Elements].
Stateflow block	Includes [Chart], [State Transition Table], and [Truth Table].
Machine level	.
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 modelled as flow charts.
State action type	Basic state action types and combined state action types.
Basic state action type	Types include entry(en), during(du), and exit(ex).
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)
State	An atomic subchart is considered a state.

7. Determining Guideline Operation Rules

This section provides general information about identifying which guidelines to adopt and the application of these guidelines to your project.

7.1. 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.

7.2. MATLAB/Simulink Version

The version of MATLAB/Simulink 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 R2020a.
It is necessary to regularly check the bug report published by MathWorks (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

Deleted Simulink Check Chapters.msg

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.

7.3. MATLAB/Simulink Settings

MATLAB/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 are listed below.

- 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 and db_0043

7.4. 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.

Note: 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.

Note: You must register custom blocks in the project's user library.

- See guideline db_0143 for defining usable blocks

7.5. Using Optimization and Configuration Parameters

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.

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 [Switch].

In contrast, for the aviation industry, this 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

- 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], or [Truth Table].

- High-integrity configuration Settings

  Please refer to the MathWorks High-Integrity System Modeling Guidelines (hisl) for additional information the configuration settings.

- Code Generation Configuration Settings

  Please refer to the MathWorks Code Generation Modeling Guidelines (cgsl) for additional information the configuration settings

7.6. Applying Guidelines for a Project

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 [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 aren't.

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]), whether [Unit Delay] should be inside or outside the subsystem, or whether [Abs] 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 fields that are 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 part 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 a guideline

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).

8. Model Architecture Explanation

This section provides a high-level overview of model architecture that is suitable for model-based development without specifying specific rules.

8.1. Roles of Simulink and Stateflow

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 [Switch Case].

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 [Relay]. 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 can be said to be 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. If 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.

8.2. 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.

Types of Hierarchies

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 concepts is not needed, it does not need to be allocated to a layer.
- Multiple layer concepts can be allocated to one layer.

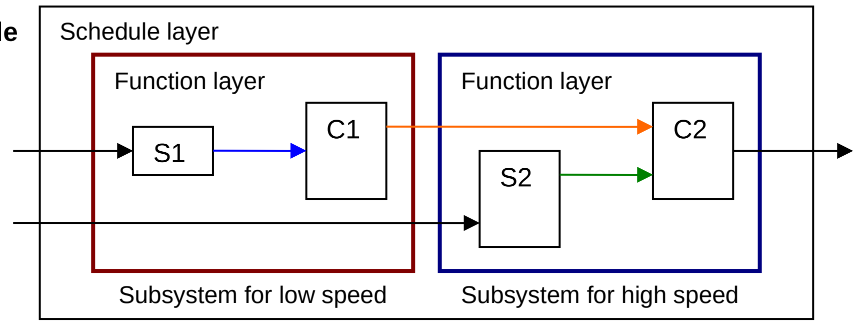
When building hierarchies, division into subsystems for the purpose of saving space within the layer shall be avoided.

Top Layer

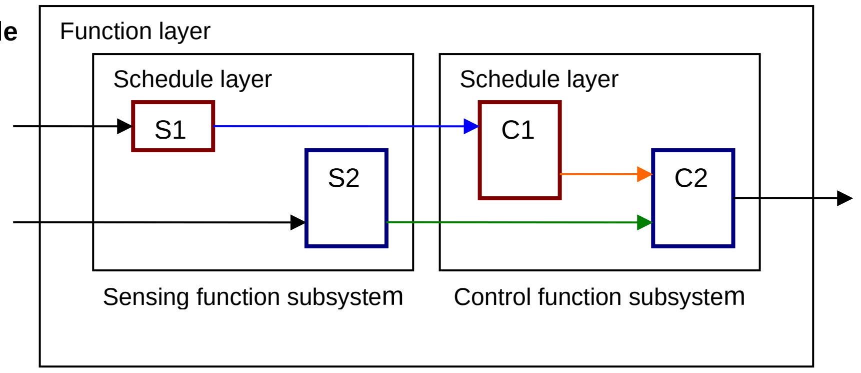
Layout methods for the top layer include:

- Simple control model — Represents both the function layer and schedule layer in the same layer. Here, function = 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.

Example
Type α



Example
Type β

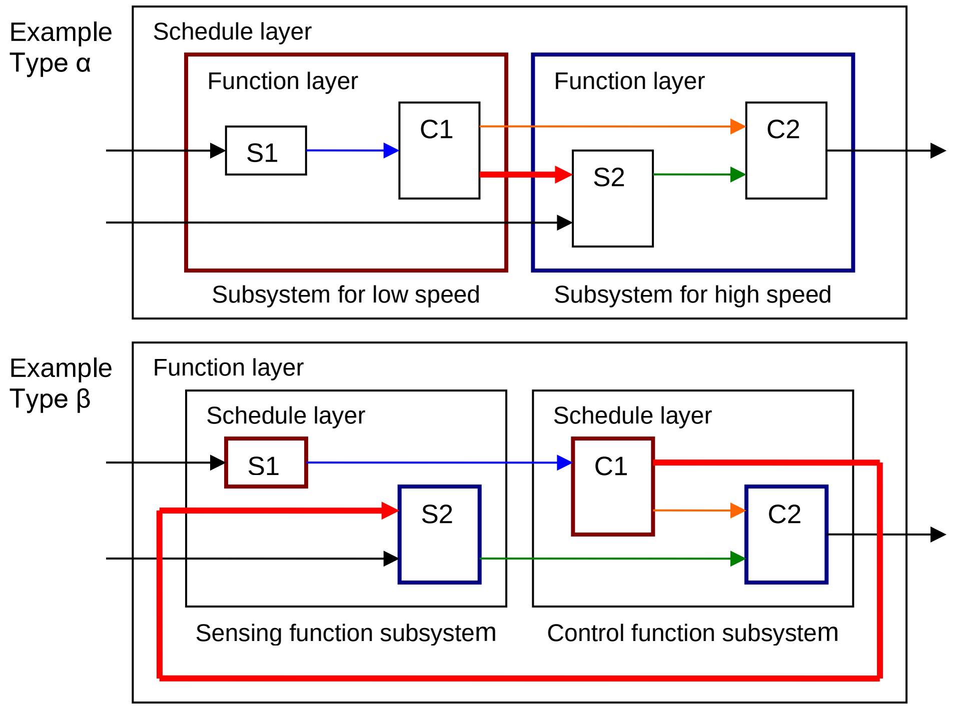


The thick frame is an Atomic setting

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)

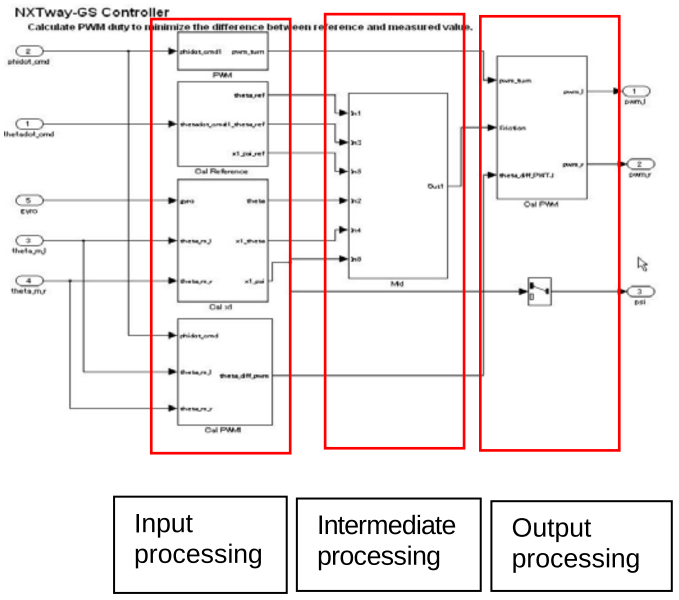
- For the generated code, affected factors include:
  - o Applicability of a real-time OS
  - o Consistency of usable sampling intervals and computation cycles to be implemented
  - o Applicable area (application domain or basic software)
  - o Source code type: AUTOSAR compliant - not compliant - not supported.

- o RAM, ROM margins (specifically RAM)

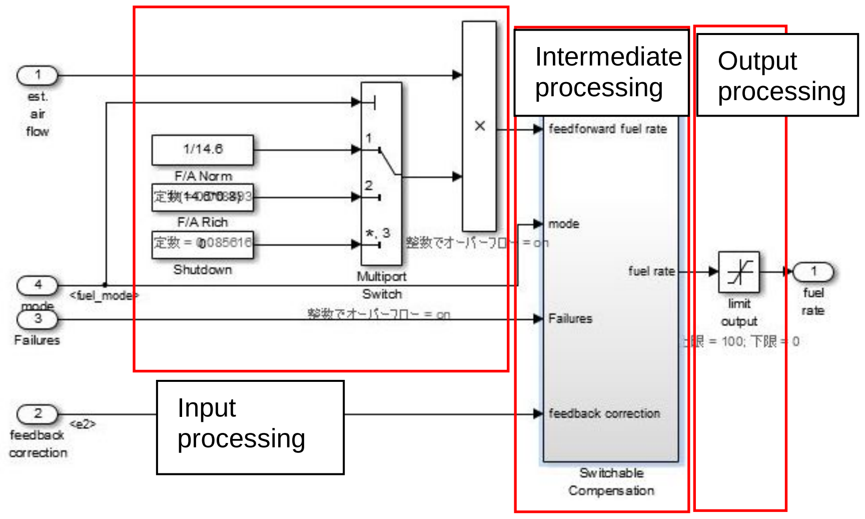
Control Flow Layers

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.



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

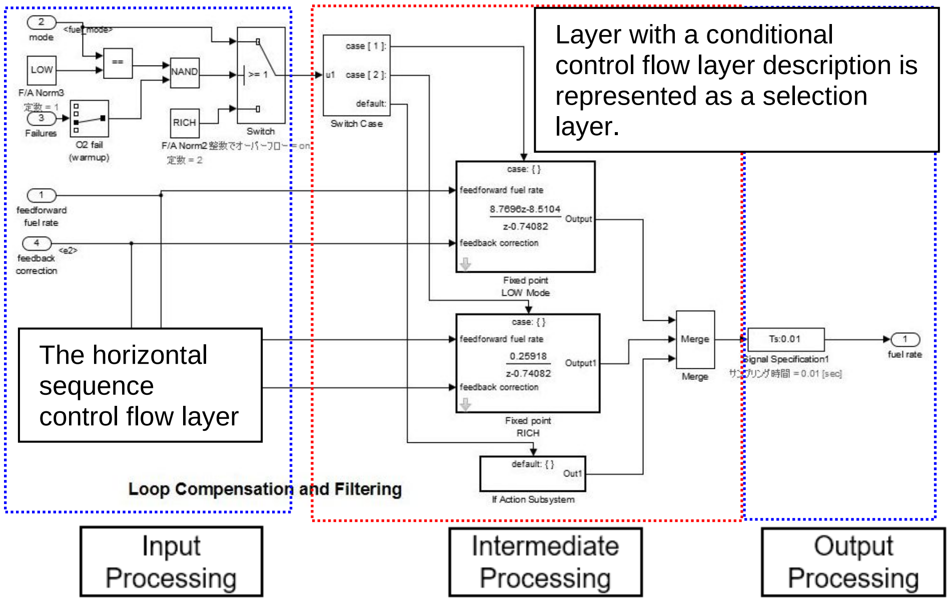
When modeling selection layers:

- 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

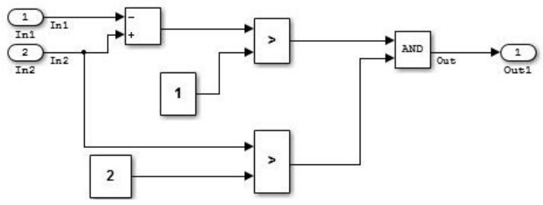
A data flow layer is the layer below the control flow layer and selection layer.

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.

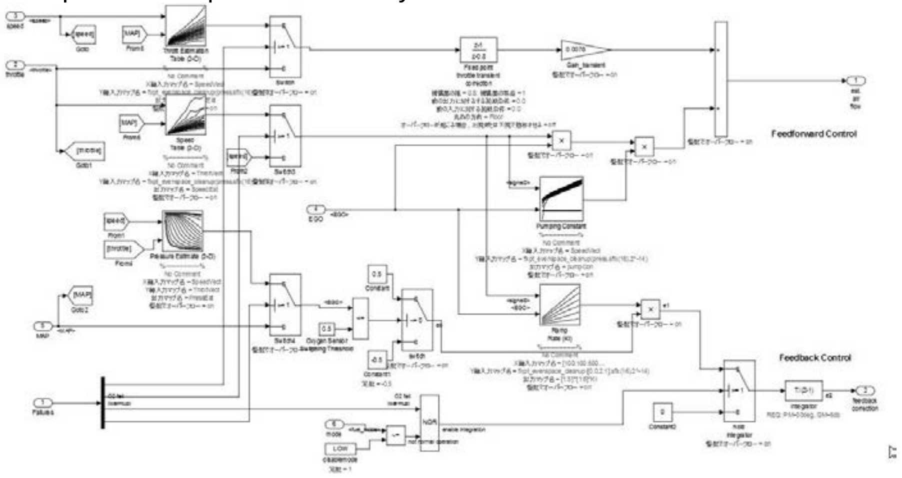
Data flow layers cannot co-exist with subsystems apart from those where exclusion conditions apply. Exclusion conditions include:

- 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.

8.3. 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.

8.3.1.1. Scheduler Settings in Embedded Software

The scheduler in embedded software has single-task and multi-task settings.

Single-task schedule settings

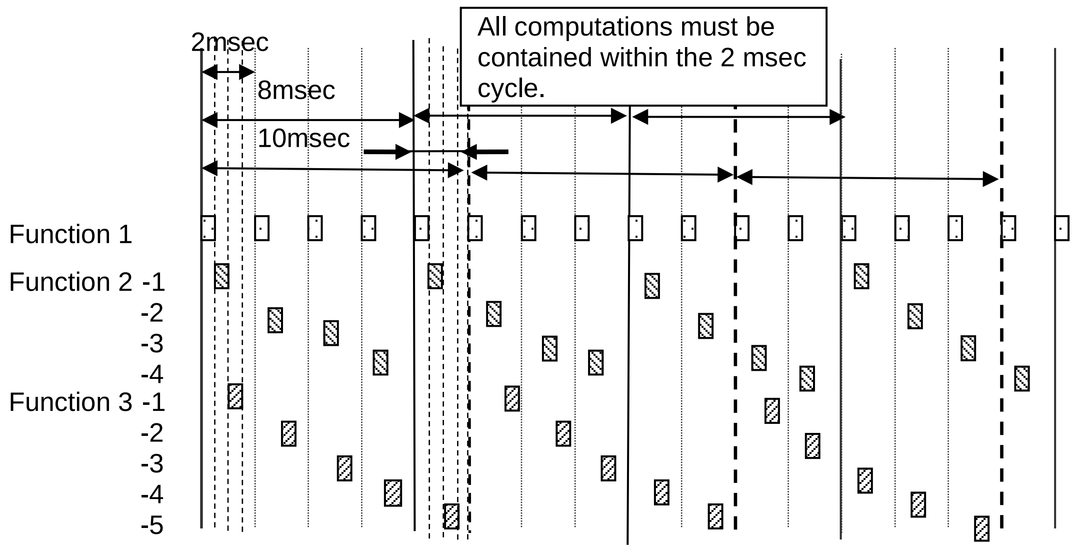
A single-task scheduler performs all processing by using basic sampling. Therefore, when processing of longer sampling is needed, the function is split so the CPU load is as evenly distributed as possible, and then processed using basic sampling. However, as equal splitting is not always possible, functions may not be able to be allocated to all cycles.

For example, basic sampling is 2 msec, and sampling rates of 2 msec, 8 msec and 10 msec exist within the model. An 8 msec function is executed once for every four 2 msec cycles, and a 10 msec function is executed once for every five. The number of executions is counted every 2 msec and the sampling function specified by this frequency is executed. Attention needs to be paid to the fact that the 2 msec, 8 msec and 10 msec cycles are all computed with the same 2 msec. Because all computations need to be completed within 2 msec, the 8 msec and 10 msec functions are split into several and adjusted so that all 2 msec computations are of an almost equal volume.

The following diagram shows the 10 msec function split into 5, and the 8 msec function split into 4.

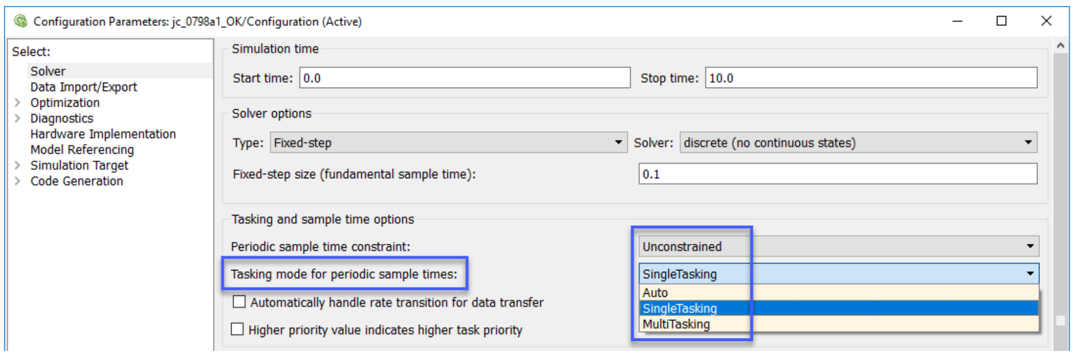
Functions	Fundamental frequency	Offset
	8msec	0msec
2-2	8msec	2msec
2-3	8msec	4msec
2-4	8msec	6msec

Functions	Fundamental frequency	Offset
3-1	10msec	0msec
3-2	10msec	2msec
3-3	10msec	4msec
3-4	10msec	6msec
3-5	10msec	8msec



To set frequency-divided tasking:

1. Set configuration parameter {Tasking mode for periodic sample times} to “Single Tasking” for Simulink task setting.


Configuration Parameters: jc_0798a1_OK/Configuration (Active)

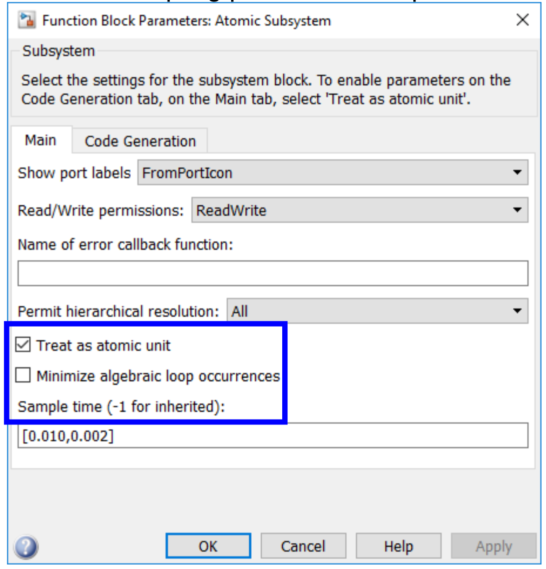
Select:
Solver
Data Import/Export
Optimization
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation

Simulation time
Start time: 0.0
Stop time: 10.0

Solver options
Type: Fixed-step
Solver: discrete (no continuous states)
Fixed-step size (fundamental sample time): 0.1

Tasking and sample time options
Periodic sample time constraint: Unconstrained
Tasking mode for periodic sample times: SingleTasking
Auto
SingleTasking
MultiTasking
☐ Automatically handle rate transition for data transfer
☐ Higher priority value indicates higher task priority

2. Enter sampling period, offset" values in the subsystem block {Sample Time}" field. A subsystem for which a sampling period can be specified is an atomic subsystem.


Function Block Parameters: Atomic Subsystem

Subsystem

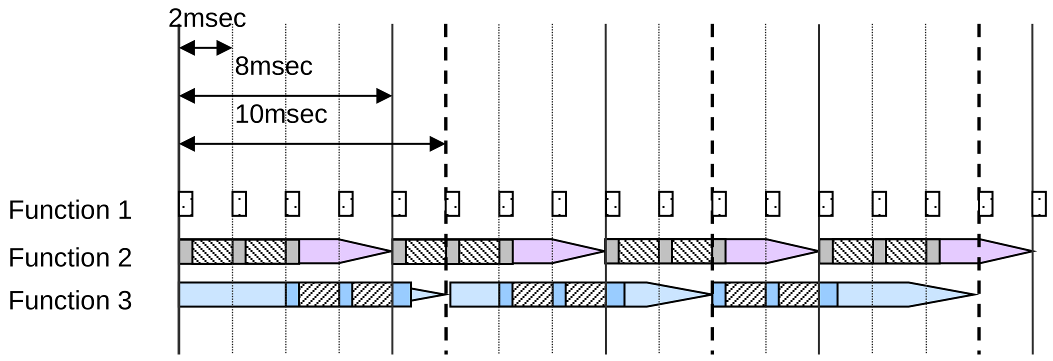
Select the settings for the subsystem block. To enable parameters on the Code Generation tab, on the Main tab, select 'Treat as atomic unit'.

Main | Code Generation

Show port labels: FromPortIcon
Read/Write permissions: ReadWrite
Name of error callback function:
Permit hierarchical resolution: All
☑ Treat as atomic unit
☐ Minimize algebraic loop occurrences
Sample time (-1 for inherited):
[0.010,0.002]

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.



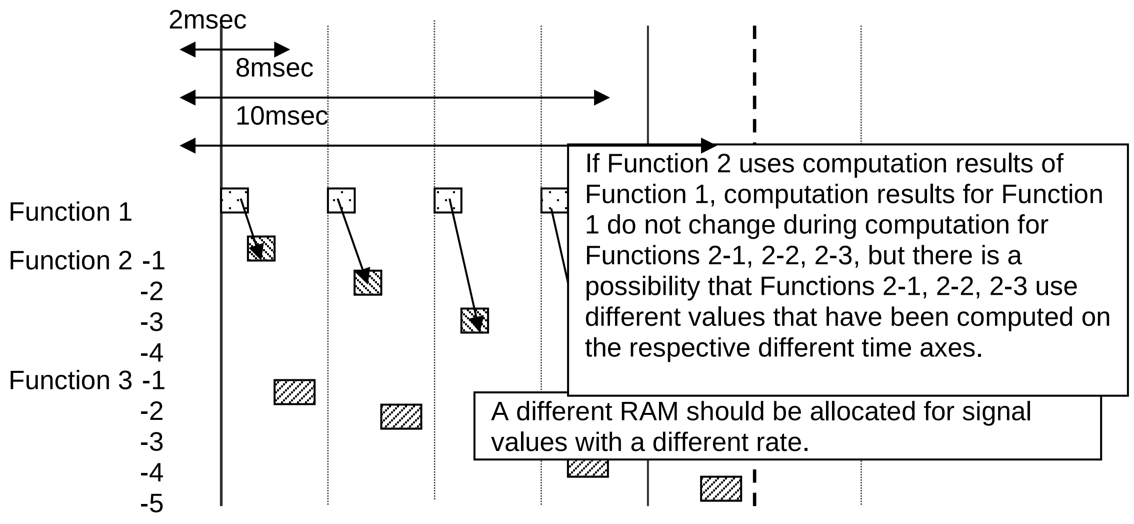
It is important that computations are completed within the cycle, including slow tasks. When the processing of a high priority computation finishes and the CPU is available, the computation for the system with the next priority ranking begins. A high priority computation process can interrupt a low priority computation, which is then aborted so the high priority computation process can execute first.

8.3.1.2. Effect of Connecting Subsystems with Sampling Differences

If subsystem B with a 20 msec sampling interval uses the output of subsystem A with a 10 msec 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.

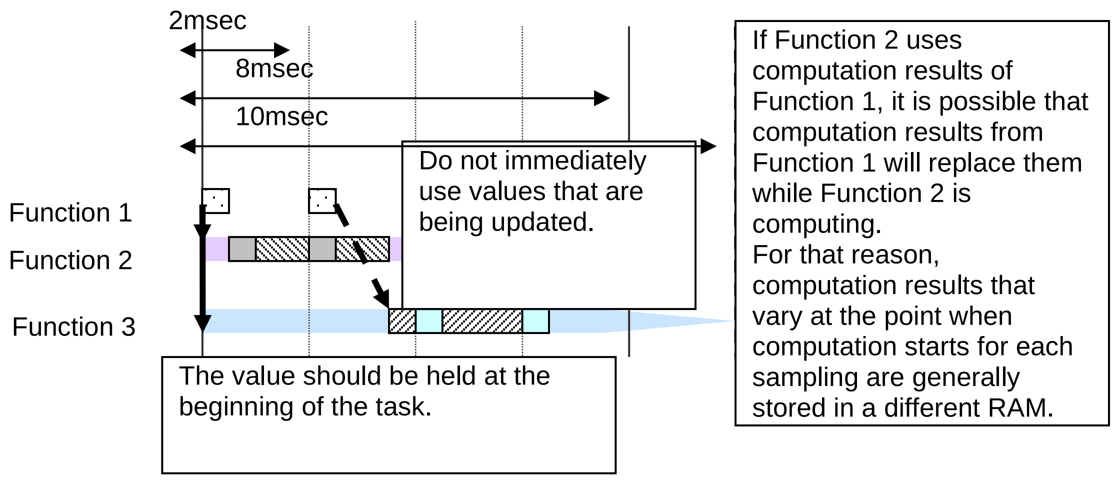

2msec
8msec
10msec
Function 1
Function 2 -1
-2
-3
-4
Function 3 -1
-2
-3
-4
-5
If Function 2 uses computation results of Function 1, computation results for Function 1 do not change during computation for Functions 2-1, 2-2, 2-3, but there is a possibility that Functions 2-1, 2-2, 2-3 use different values that have been computed on the respective different time axes.
A different RAM should be allocated for signal values with a different rate.

Single-task scheduler settings

Signal values are the same within the same 2 msec cycle, but when there are different 2 msec 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.


2msec
8msec
10msec
Function 1
Function 2
Function 3
Do not immediately use values that are being updated.
The value should be held at the beginning of the task.
If Function 2 uses 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 generally stored in a different RAM.

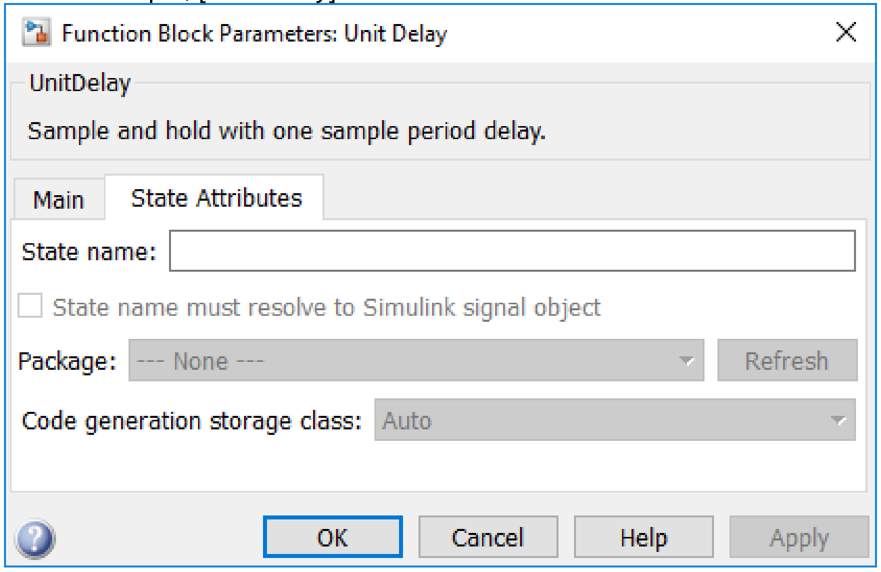
9. Appendices

9.1. Simulink Functions

9.1.1.1. 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, [Unit Delay] has State Attributes.


Function Block Parameters: Unit Delay

UnitDelay

Sample and hold with one sample period delay.

Main | State Attributes

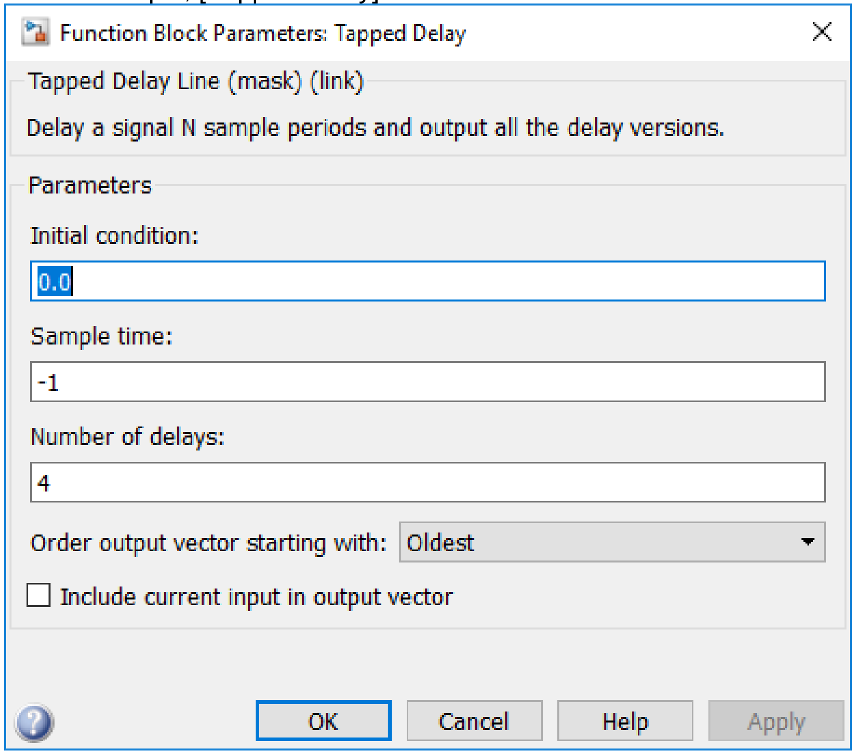
State name:

☐ State name must resolve to Simulink signal object

Package: --- None ---

Code generation storage class: Auto

In this example, [Tapped Delay] does not have State Attributes.


Function Block Parameters: Tapped Delay

Tapped Delay Line (mask) (link)

Delay a signal N sample periods and output all the delay versions.

Parameters

Initial condition: 0.0

Sample time: -1

Number of delays: 4

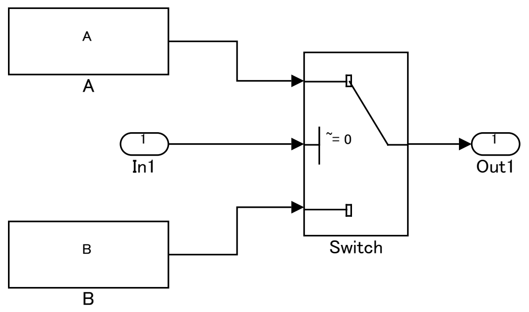
Order output vector starting with: Oldest

☐ Include current input in output vector

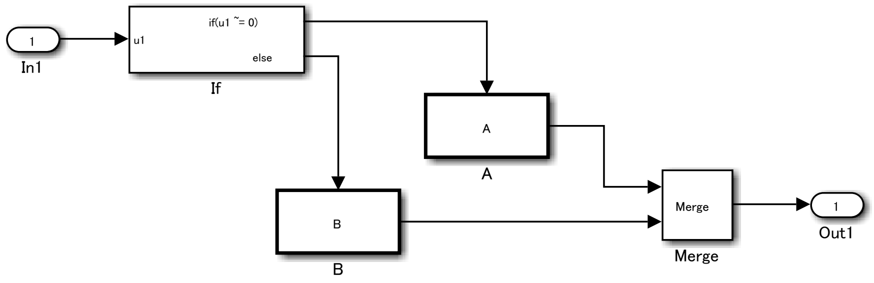
See guideline: jc_0640

9.1.1.2. Branch Syntax with State Variables

[Switch] and conditional subsystems behave differently when state variables are used.
Depending on the configuration setting, when any state variable exists, [Switch] 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 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 [Switch] and a conditional control flow is listed in the following tables. Familiarize yourself with these behaviors to determine which structure, [Switch], or conditional subsystem is most suitable for the intended purpose.

Behavior of subsystem A

Control port condition	(in subsystem A) State variables	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	

Initialization timing of subsystem A

	ActionPort	Initialize
Switch	—	First time only
Conditional subsystem	Hold	First time only
	Reset	At returned by condition

See guidelines: jc_0656 and jc_0657

9.1.1.3. Subsystem

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, jc_0243, db_0143, db_0144, db_0141, jc_0653, jc_0171, jc_0602, jc_0081, db_0081

9.1.1.4. 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 and jc_0245

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 [Inport], set the {Data type} to ”auto”
・For [Outport], set the {Data type} to ”auto”
・For [Sum], set the output {Data type} to ”Inherit via back propagation”

See guideline: jc_0644

9.1.1.5. Vector Signals/Path Signal

Individual scholar 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. [Bus Selector] shall be used only with bus signal inputs. It shall not be used to extract a scholar signal from a vector signal.

Example
The following 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]

Types of vector	Size
Cylinder vector	[1 cylinder number]
Location vector based on a 2-dimensional coordination	[1 2]
Location vector based on 3-dimensional coordination	[1 3]

The following 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, jc_0222, jc_0245, db_0097, jc_0630, and jc_0659

9.1.1.6. 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, refer to the Simulink user help "Use Enumerated Data in Simulink Models.

9.2. Stateflow Functions

9.2.1.1. Operators Available for Stateflow

For additional information about the Stateflow operators, see " Supported Operations for Chart Data" in the Stateflow user help.
Related ID: na_0001, jc_0655

9.2.1.2. 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.

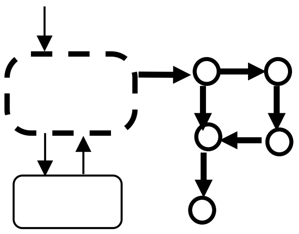
Flow chart and state transition diagram

	Start point	End point
Flow chart	Default transition Or,	All terminations from the state are connected to the connective junction.
State transition diagram	Default transition Or,	Either termination should be connected to the state

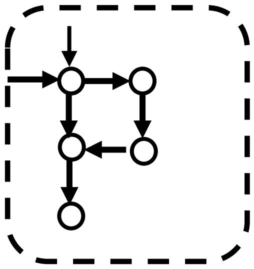
Difference between a general flow chart and state transition diagram

Flow Chart

Flow chart outside a state

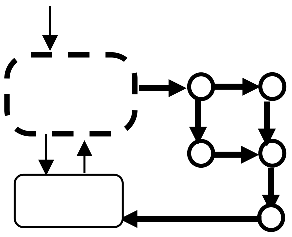


Flow chart inside a state

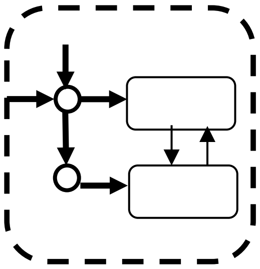


State Transition Diagram

State transition outside a state

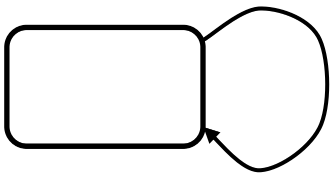
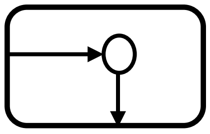


State transition inside a state



Mixture of flow charts and state transition diagrams with self-transition has more strict constraints.

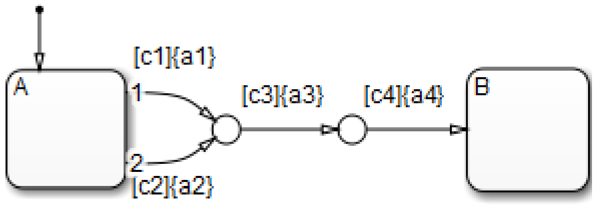
Example of flow chart with self-transition

	State Transition Diagram	
	Self transition outside a state	Self transition inside a state
State transition diagram	 *A self transition is formed outside a state and then reset after execution.*	 *A self transition is formed inside a state and then reset using a during action.*

See guidelines: db_0132 and jc_0752

9.2.1.3. Backtrack

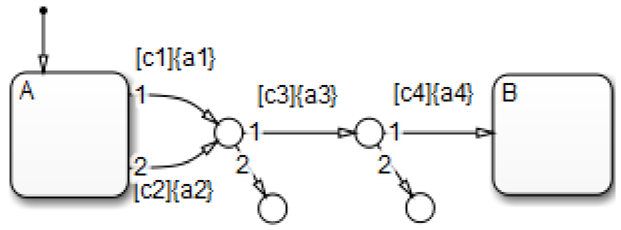
This example shows the behavior of transitions with junctions that force backtracking behavior in flow charts. The chart uses implicit ordering of outgoing transitions.



Initially, state A is active and transition conditions c1, c2, and c3 are true and 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.
2. Transition condition c1 is true, so action a1 executes.
3. Transition condition c3 is true, so action a3 executes.
4. Transition condition c4 is not true and, therefore, the control flow backtracks to state A.
5. 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.
6. Transition condition c2 is true, so action a2 executes.
7. Transition condition c3 is true, so action a3 executes.
8. Transition condition c4 is not true and, therefore, the control flow backtracks to state A.
9. 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.



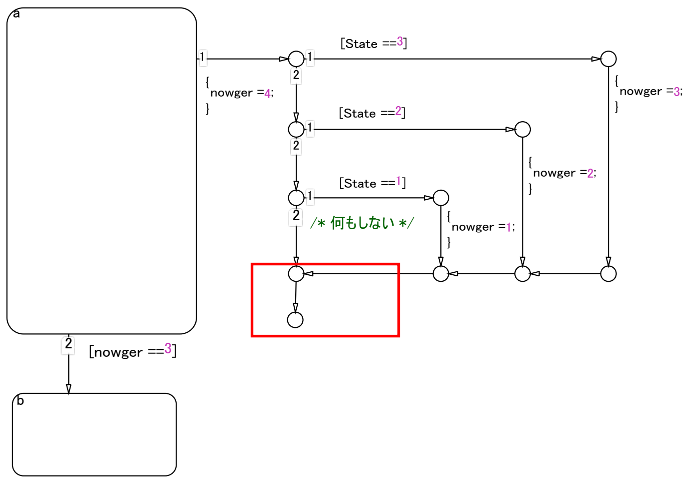
See guidelines: jc_0751 and jc_0773

9.2.1.4. 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.

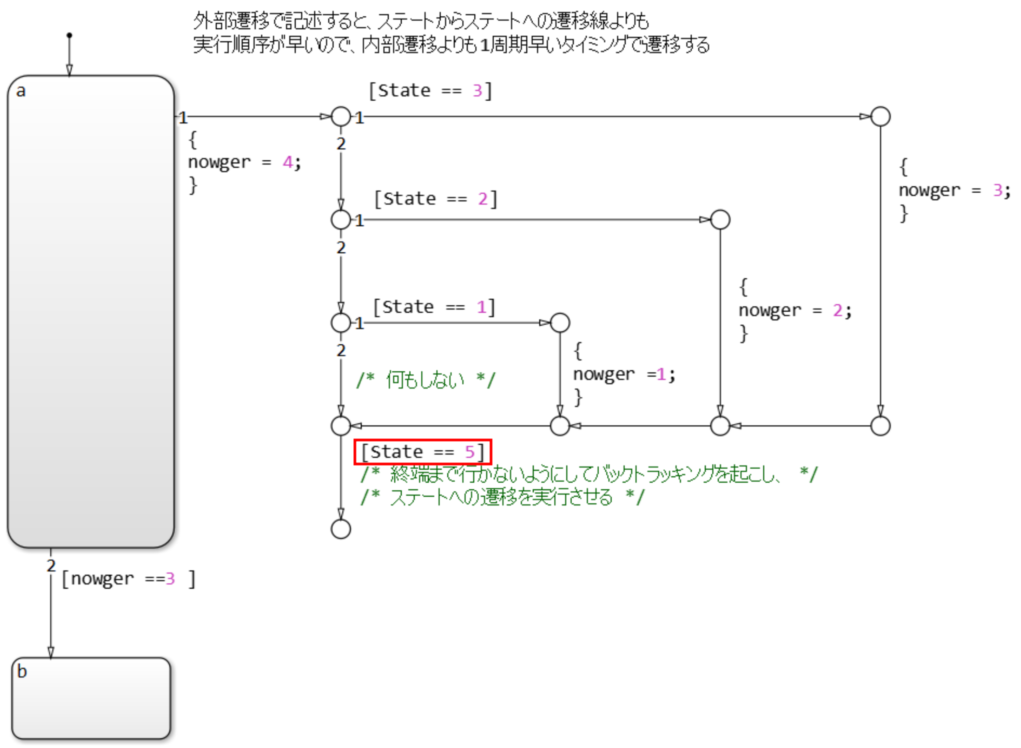
The following flow chart, which evaluates transition from a to b after executing the flow chart outside the state, appears to execute the transition within the same period as that of a newer calculation.

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 with the line below, embed a transition condition that is intentionally not positioned at the termination of the external flow chart; it should be described so that the transition line from a to b is evaluated after the flow chart has been executed.

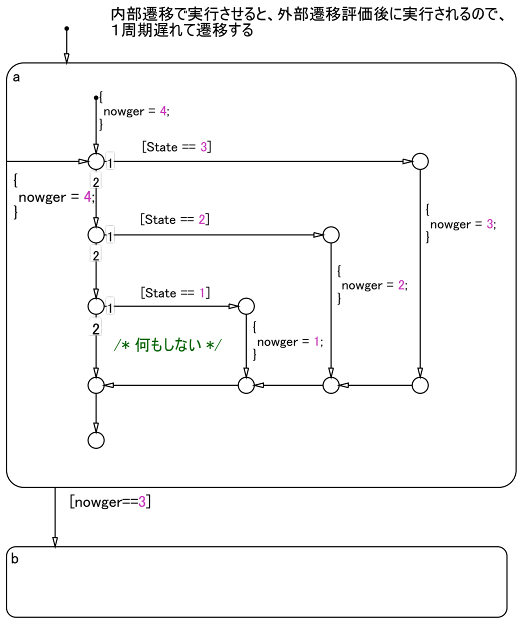
This enables the external flow chart to execute before the transition, and to be evaluated using the most recent value at the instant of the transition. Note that this chart contains a dead path where the transition condition will never hold, which can cause an error when the specification is changed in the future. Use this chart structure with caution.


外部遷移で記述すると、ステートからステートへの遷移線よりも
実行順序が早いので、内部遷移よりも1周期早いタイミングで遷移する
a
1
{
nowger = 4;
}
[State == 3]
1
2
{
nowger = 3;
}
[State == 2]
1
2
{
nowger = 2;
}
[State == 1]
1
2
/* 何もしない */
{
nowger =1;
}
[State == 5]
/* 終端まで行かないようにしてバックトラッキングを起こし、 */
/* ステートへの遷移を実行させる */
2 [nowger ==3]
b

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 and jc_0773

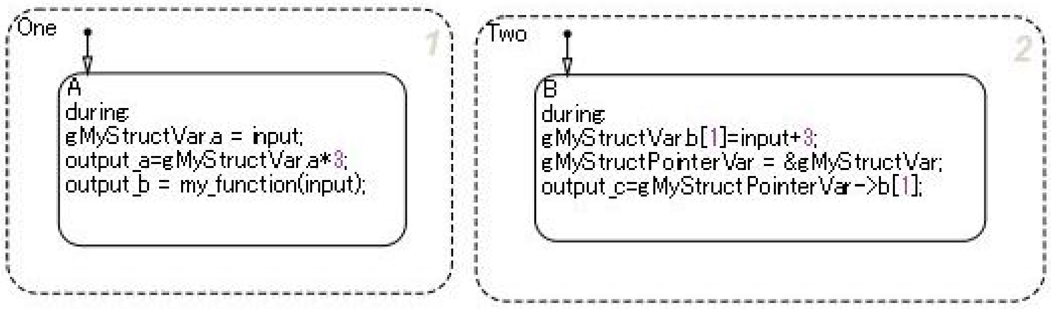
9.2.1.5. Pointer Variables

Describe using the example `model sf_custom`.

gMyStructVar is not defined in Stateflow.
Loading of C source code is set on the Code Generation pane of Configuration Parameter.
Normally, 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.

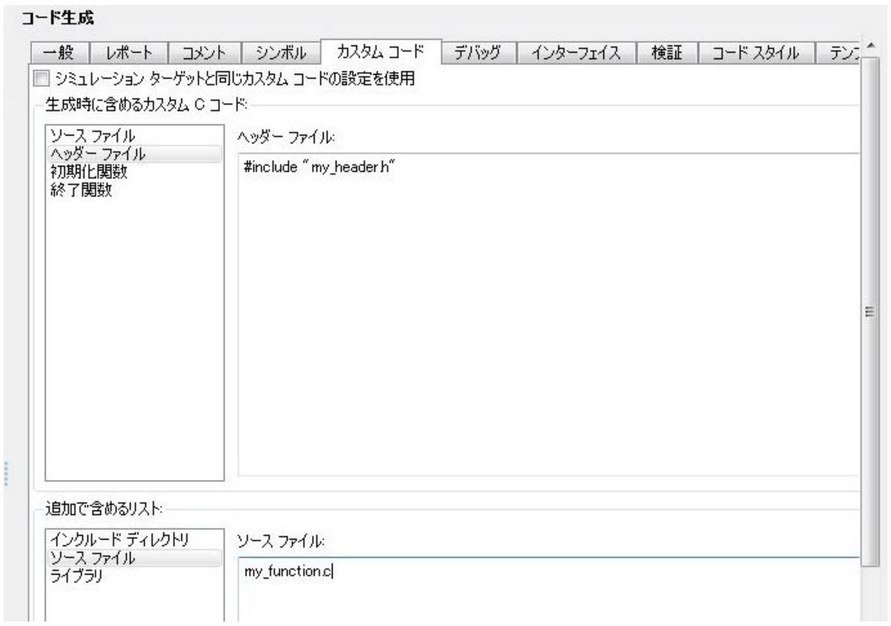
```
---------my_header.h--------------
#include "tmwtypes.h"

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;

---------------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);
}
```

------------------------Inside of Stateflow -----------------------------



```
during:
gMyStructVar.a = input;
output_a=gMyStructVar.a*3;
output_b = my_function(input);
```

```
during:
gMyStructVar.b[1]=input+3;
gMyStructPointerVar = &gMyStructVar;
output_c=gMyStructPointerVar->b[1];
```



シミュレーション ターゲットと同じカスタム コードの設定を使用

生成時に含めるカスタム C コード:

ヘッダー ファイル:

```
#include "my_header.h"
```

追加で含めるリスト:

ソース ファイル:

```
my_function.c
```

9.3. Initialization

9.3.1.1. 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:

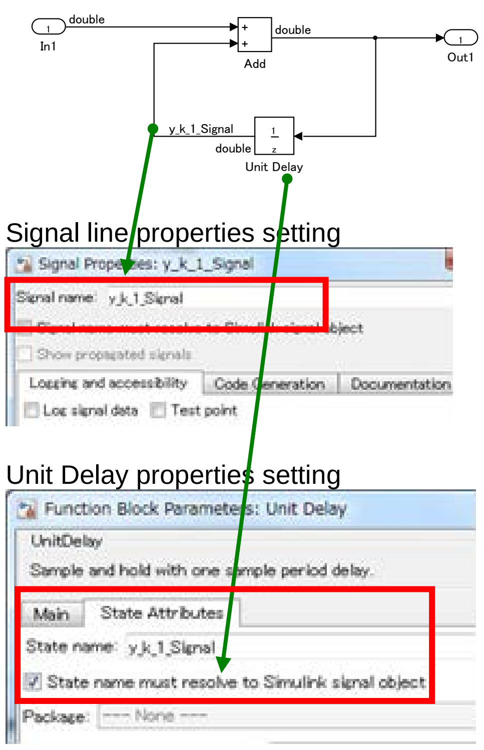
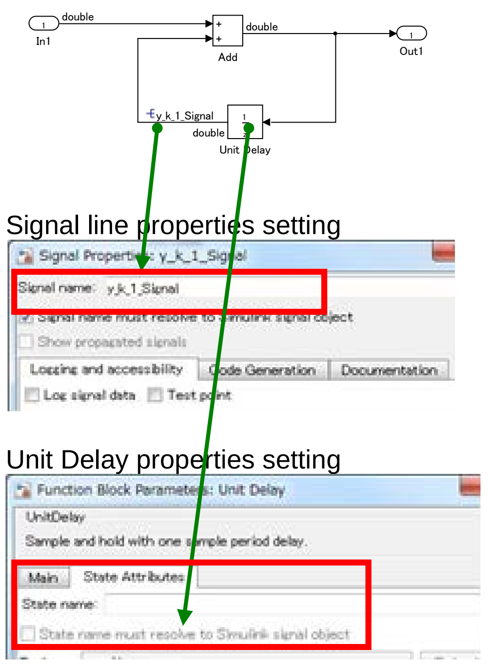
- When state variables are defined AND blocks that have state variables are used.
  - o Use the internal block settings.
  - o Use the external input values.
- When state variables are defined AND initial values are enabled for a block when a specific configuration is performed.
  - o Set initial values in Merge blocks.
  - o Use signals registered in the data dictionary.
- When signal settings (with RAM) have been defined that can be referenced from the outside.
  - o Use signals registered in the data dictionary.

9.3.1.2. Initial Values of Signals Registered in the Data Dictionary

Set initial values for signals registered in the data dictionary.

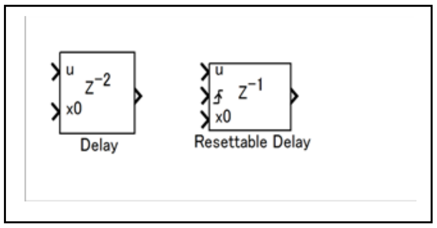
・Discrete block groups, such as [Unit Delay] and [Data Store Memory] have state variables.
In the case of automatic code generation, the signal name, type, and initial value can be set for state variables by matching it to the signal in the data dictionary (associated with Simulink signal objects). When using a signal defined in the data dictionary for a state variable, the respective initial values should conform to the same value.

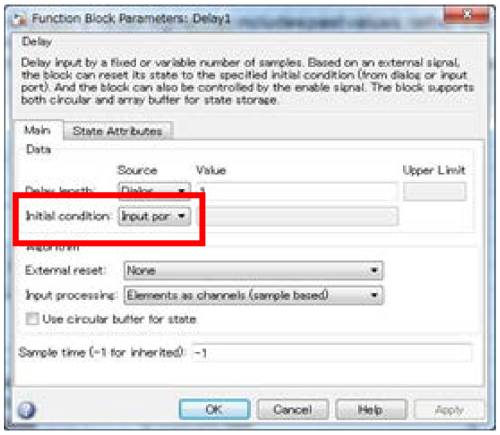
・When using a signal defined in the data dictionary for a state variable
For discrete blocks, such as a [Unit Delay] and [Data Store Memory], settings are performed not when using signals defined in the data dictionary for the block output line, but for the state variables inside the block. Even when the signal name of the data dictionary is assigned to the signal line, RAM is reserved in duplicate, which is a waste of RAM.

【Correct】When the signal is defined for the state variables inside the block.	【Incorrect】When the signal is defined for the output signal of the block that has state variables.
 Signal line properties setting Signal Properties: y_k_1_Signal Signal name: y_k_1_Signal ☐ Signal name must resolve to Simulink signal object ☐ Show propagated signals Logging and accessibility / Code Generation / Documentation ☐ Log signal data ☐ Test point Unit Delay properties setting Function Block Parameters: Unit Delay UnitDelay Sample and hold with one sample period delay. Main / State Attributes State name: y_k_1_Signal ☑ State name must resolve to Simulink signal object Package: ---- None ----	 Signal line properties setting Signal Properties: y_k_1_Signal Signal name: y_k_1_Signal ☑ Signal name must resolve to Simulink signal object ☐ Show propagated signals Logging and accessibility / Code Generation / Documentation ☐ Log signal data ☐ Test point Unit Delay properties setting Function Block Parameters: Unit Delay UnitDelay Sample and hold with one sample period delay. Main / State Attributes State name: ☐ State name must resolve to Simulink signal object

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 (model name`. 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.

9.3.1.3. Block Whose External Input Value is the Initial Value




Function Block Parameters: Delay1

Delay

Delay input by a fixed or variable number of samples. Based on an external signal, the block can reset its state to the specified initial condition (from dialog or input port). And the block can also be controlled by the enable signal. The block supports both circular and array buffer for state storage.

Main | State Attributes

Data

	Source	Value	Upper Limit
Delay length:	Dialog	1	
Initial condition:	Input port		

Algorithm

External reset: None

Input processing: Elements as channels (sample based)

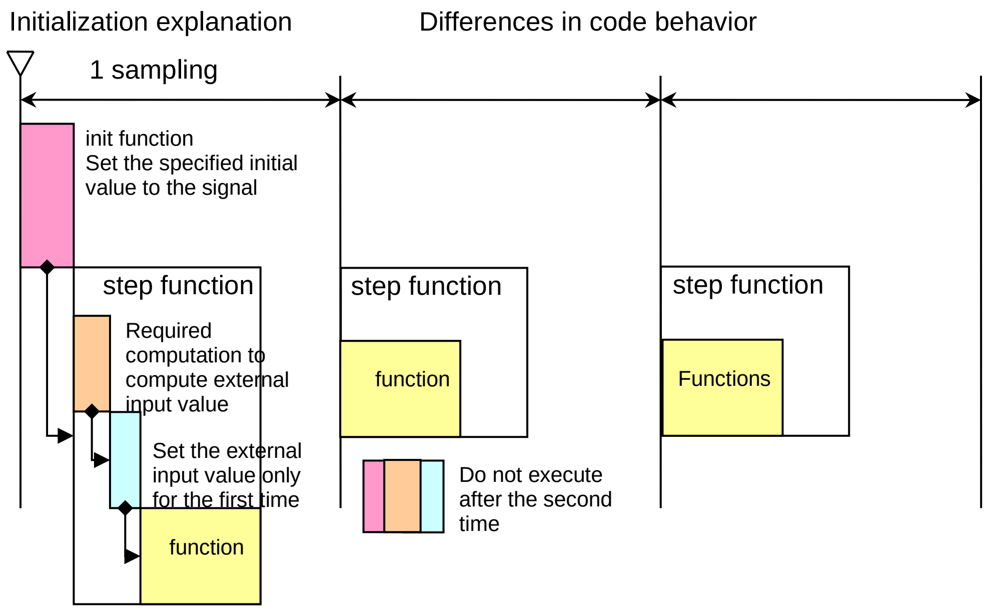
☐ Use circular buffer for state

Sample time (-1 for inherited): -1

OK | Cancel | Help | Apply

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.



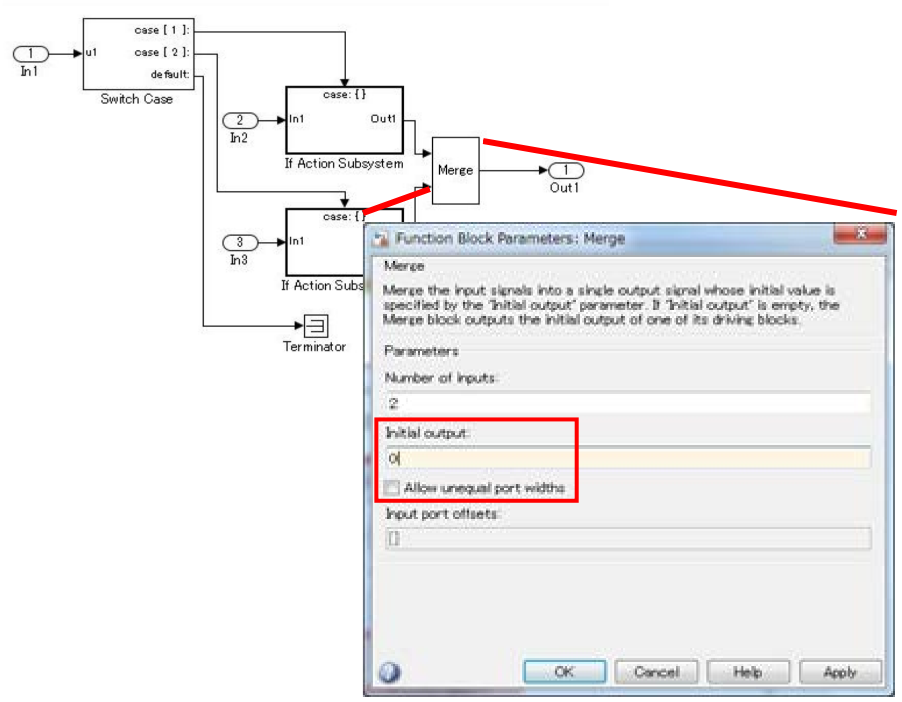
Initialization explanation
Differences in code behavior
1 sampling
init function
Set the specified initial value to the signal
step function
Required computation to compute external input value
Set the external input value only for the first time
function
step function
function
Do not execute after the second time
step function
Functions

9.3.1.4. 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]. When initial values are required in theses combinations, either of the following modeling methods is performed:
・Set in [Outport]
・Set in [Merge]
・If an `mpt` signal is defined behind [Merge], set in `mpt` signal

Exception:
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.

【Correct】Initial value set in [Merge]



Function Block Parameters: Merge

Merge

Merge the input signals into a single output signal whose initial value is specified by the 'Initial output' parameter. If 'Initial output' is empty, the Merge block outputs the initial output of one of its driving blocks.

Parameters

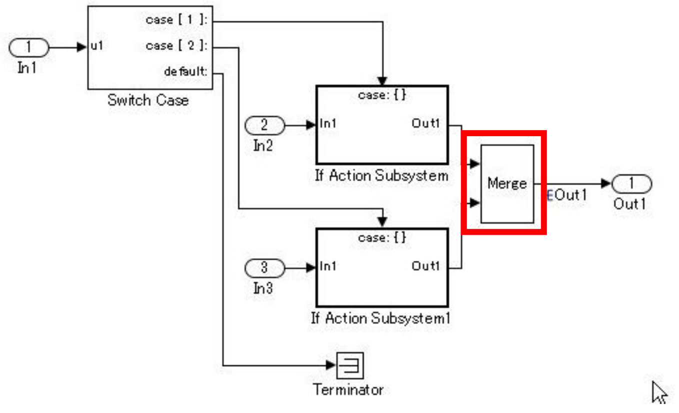
Number of inputs:
2

Initial output:
0

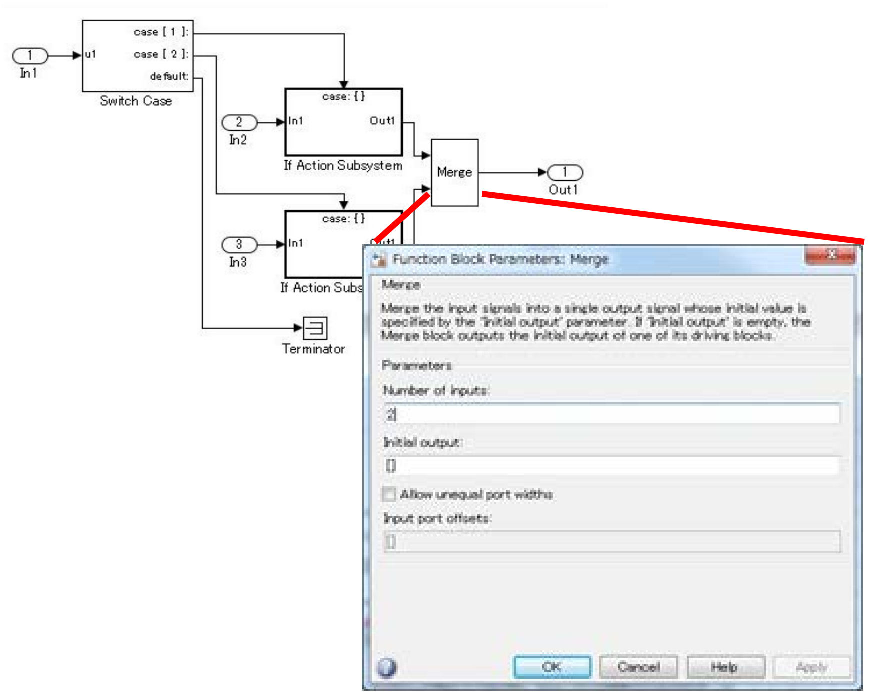
☐ Allow unequal port widths

Input port offsets:
[]

【Correct】Initial value set in mpt object



【Incorrect】Despite the requirement for an initial value setting, it is not shown anywhere.



Function Block Parameters: Merge

Merge

Merge the input signals into a single output signal whose initial value is specified by the 'Initial output' parameter. If 'Initial output' is empty, the Merge block outputs the initial output of one of its driving blocks.

Parameters

Number of inputs:
2

Initial output:
[]

Allow unequal port widths

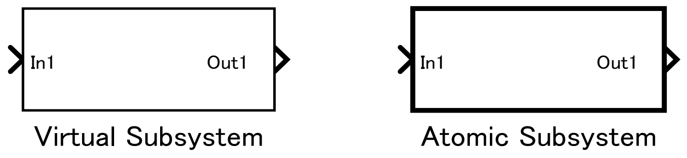
Input port offsets:
[]

9.4. Miscellaneous

9.4.1.1. Atomic Subsystems and Virtual Subsystems

There are two types of subsystems, Virtual subsystem 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.



For additional information, in the Simulink user help see:

- Subsystems
- Explanation of algebraic loops

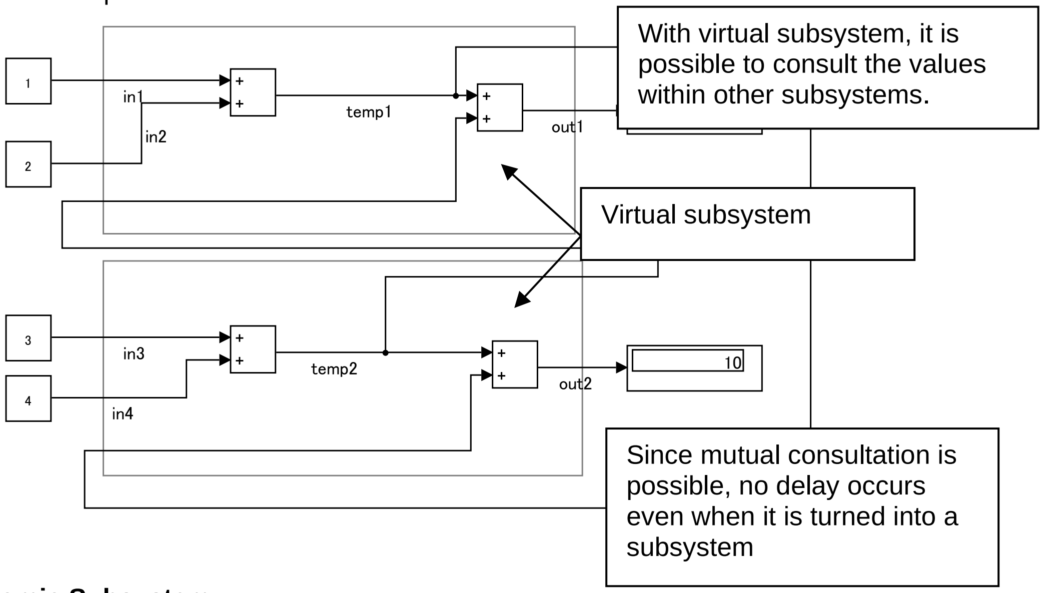
Virtual Subsystem

A block that provides a visual representation is known as a "virtual block. ". For example, [Mux] that compiles several signal lines, [From] that hands out the signal, and [Goto] blocks all correspond 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.

```
temp1= in1 + in2
temp2= in3 + in4
out1= in1 + in2 + temp2
```

```
out2= temp1 + in3 + in4
```

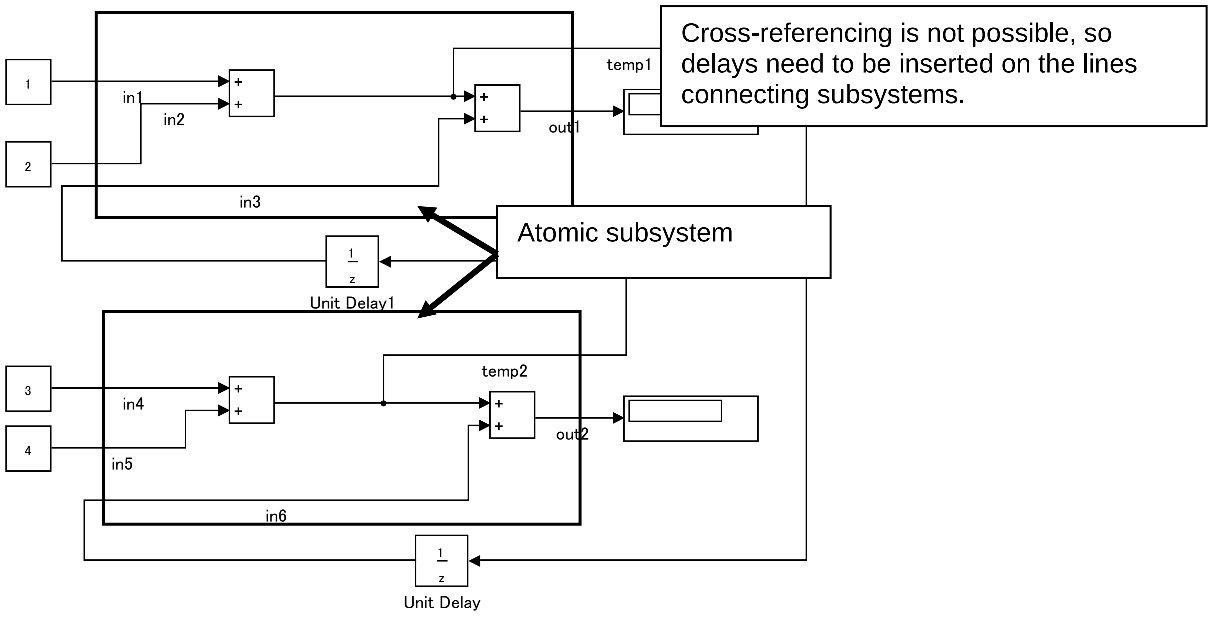

With virtual subsystem, it is possible to consult the values within other subsystems.
Virtual subsystem
Since mutual consultation is possible, no delay occurs even when it is turned into a subsystem
1
2
3
4
in1
in2
in3
in4
temp1
temp2
out1
out2
10

Atomic Subsystem

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.

```
temp1= in1 + in2
temp2= in4 + in5
out1= in1+ in2 + in3
out2= in4+ in5 + in6
in3= temp2
in6= temp1
```

Atomic subsystems prohibit the direct referencing of the interim calculation results to other subsystems.


Cross-referencing is not possible, so delays need to be inserted on the lines connecting subsystems.
Atomic subsystem
1
2
3
4
in1
in2
in3
in4
in5
in6
temp1
temp2
out1
out2
Unit Delay1
Unit Delay

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).

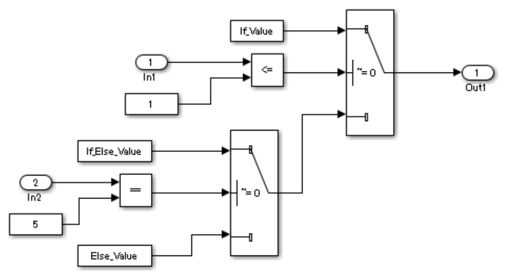
- Depending on the relationship before and after, a static RAM section should be secured inside the subsystem for the output signal.
- Atomic subsystems (including the addition of function settings) should be used with caution. Factor setting will not simply have a factor name inserted within a C code. 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.

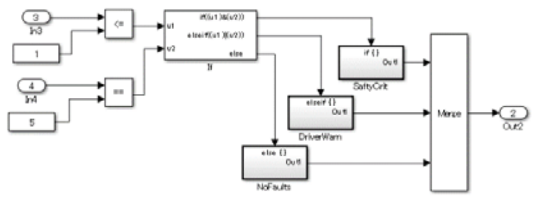
9.5. Modeling Knowledge / Usage Patterns

Appendix 1: Simulink Patterns for If, elseif, else Constructs

These patterns shall be used for if, elseif, else constructs.

Function	**Simulink pattern**

[Switch] is used.
If, elseif, else construct
 	
`if (If_Condition)`
`{`
`  output_signal = If_Value;`
`}`
`else if (Else_If_Condition)`
`{`
`  output_signal = Else_If_Value;`
`}`
`else`
`{`
`  output_signal = Else_Value;`
`}`
	 If_Value
	1
	In1
	1
	<=
	~= 0
	1
	Out1
	If_Else_Value
	2
	In2
	==
	5
	~= 0
	Else_Value

If, elseif, else construct using Action Subsystem
 	
`if (Fault_1_Active & Fault_2_Active)`
`{`
`  ErrMsg = SaftyCrit;`
`}`
`else if (Fault_1_Active |`
`Fault_2_Active)`
`{`
`  ErrMsg = DriverWarn;`
`}`
`else`
`{`
`  ErrMsg = NoFaults;`
`}`
	 3
	In3
	1
	<=
	4
	In4
	5
	==
	u1
	u2
	if((u1)&&(u2))
	elseif((u1)||(u2))
	else
	If
	if { }
	Out1
	SaftyCrit
	elseif { }
	Out1
	DriverWarn
	else { }
	Out1
	NoFaults
	Merge
	2
	Out2

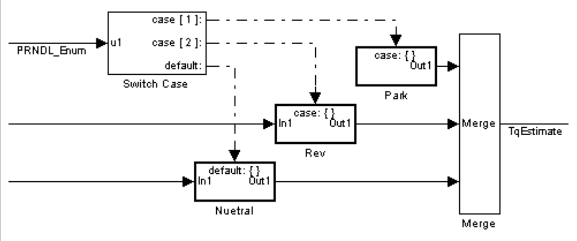
Appendix 2: Simulink Patterns for Case Constructs

These patterns shall be used for case constructs.

Function	**Simulink pattern**

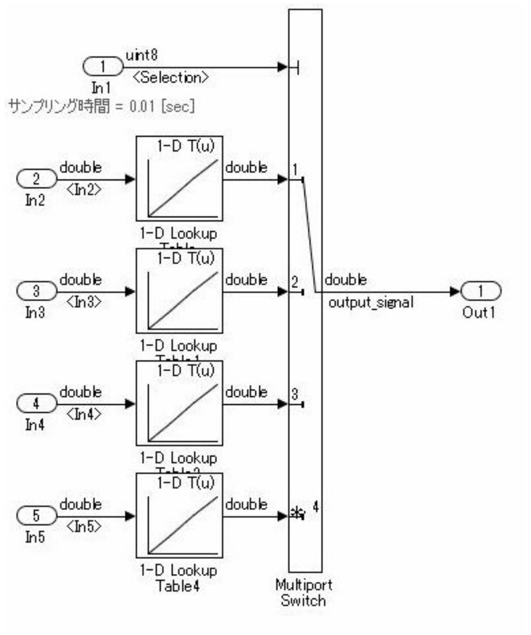
Case construct using if Action Subsystem
 	
`switch (PRNDL_Enum)`
`{`	

```
  case 1
    TqEstimate = ParkV;
    break;
  case 2
    TqEstimate = RevV;
    break;
  default
    TqEstimate = NeutralV;
    break;
}
```



Case construct using Multiport Switch

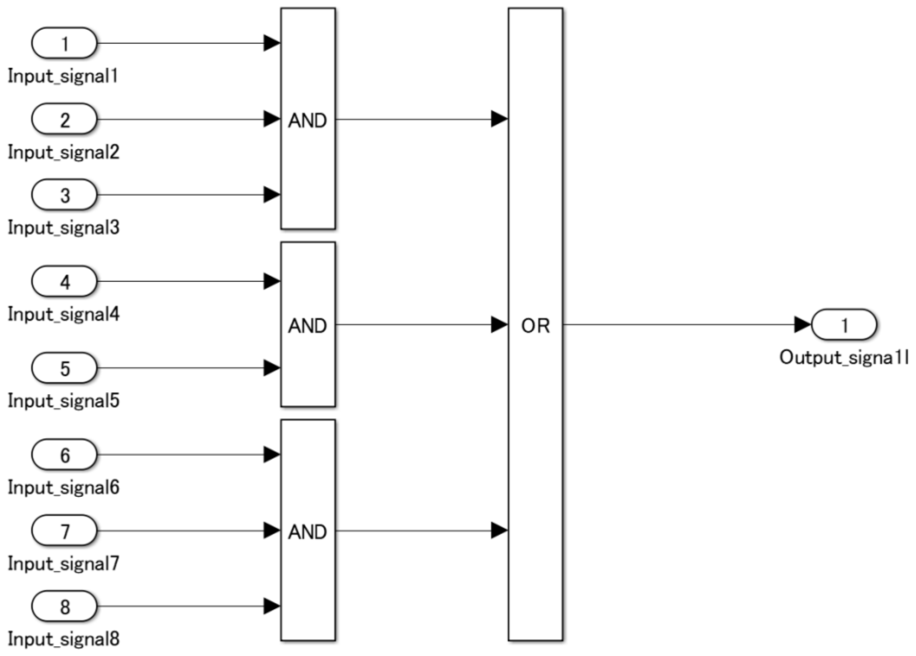
```
switch (Selection)
{
  case 1:
    output_signal =
      look1_binlxpw(In2,y1,x1,3U);
    break;
  case 2:
    output_signal =
      look1_binlxpw(In3,y2,x2,3U);
    break;
  case 3:
    output_signal =
      look1_binlxpw(In4,y3,x3,3U);
    break;
  default:
    output_signal =
      look1_binlxpw(In5,y4,x4,3U);
    break;
}
```



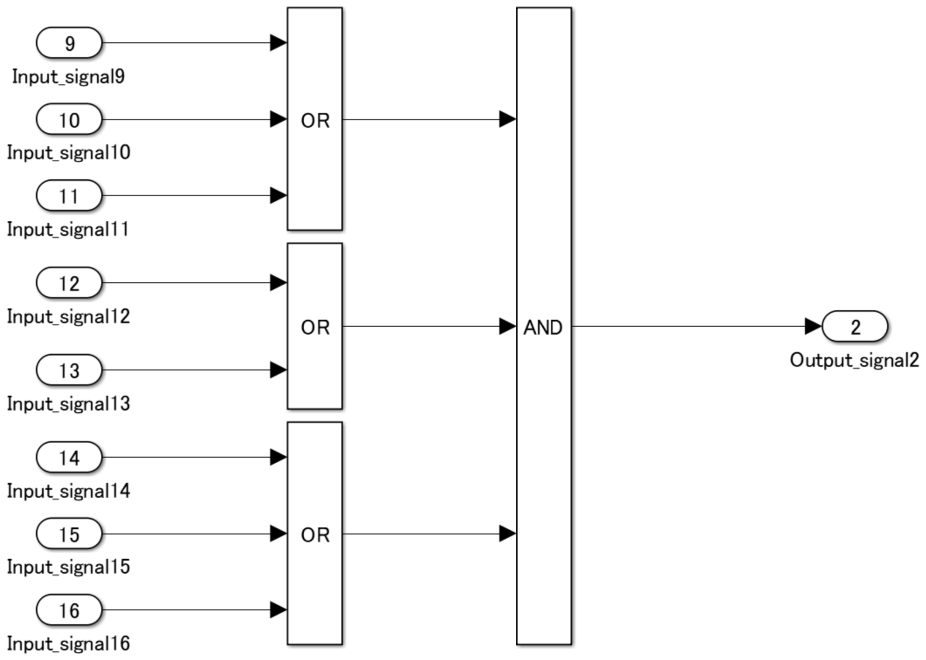
Appendix 3: Simulink Patterns for Logical Constructs

These patterns shall be used for Simulink logical constructs.

Conjunctive normal form



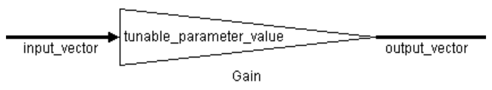
Disjunctive normal form



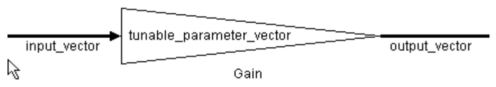
Appendix 4: Simulink Patterns for Vector Signals

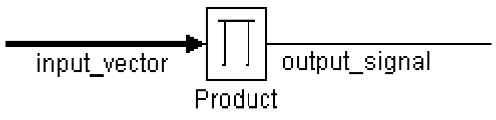
These patterns shall be used for vector signals.

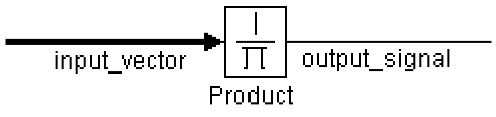
Function	Simulink pattern

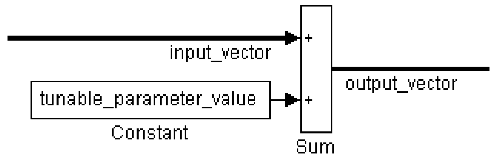
Vector signal and parameter (scalar) multiplication
`for (i=0; i>input_vector_size; i++) {`
`  output_vector[i] = input_vector[i] *`
`    tunable_parameter_value;`
`}`
 	
(Reference: generated code of R2013b)
	 input_vector
	tunable_parameter_value
	Gain
	output_vector

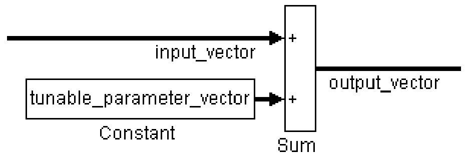
`for (i = 0; i < input_vectorDim; i++) {`
`  output_vector[i] =`
`    tunable_parameter_value *`
`    input_vector[i];`
`}`
 	
(As the code is generated using a variable number of dimensions, the upper limit of the normal loop is a direct value.)	

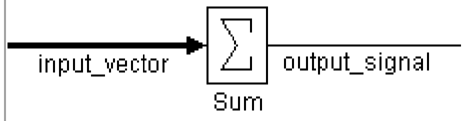
Multiplication of vector signals and parameters (vectors)
 	
`for (i=0; i>input_vector_size; i++) {`
`  output_vector[i] = input_vector[i] *`
`    tunable_parameter_vector[i];`
`}`
	 input_vector
	tunable_parameter_vector
	Gain
	output_vector

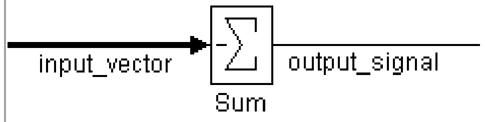
Vector signal element multiplication
 	
`output_signal = 1;`
`for (i=0; i>input_vector_size; i++) {`
`  output_signal = output_signal *`
`    input_vector[i];`
`}`
	

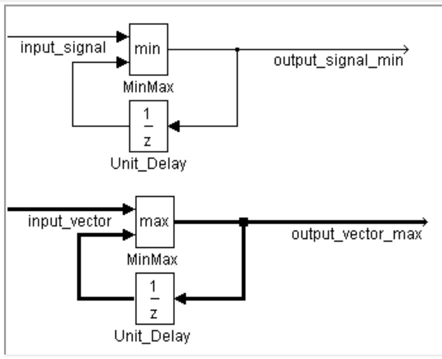
Vector signal element division
 	
`output_signal = 1;`
`for (i=0; i>input_vector_size; i++) {`
`  output_signal = output_signal /`
`    input_vector[i];`
`}`
	

Vector signal and parameter (scalar) addition
 	
`for (i=0; i>input_vector_size; i++) {`
`  output_vector[i] = input_vector[i] +`
`    tunable_parameter_value;`
`}`
	 input_vector
	tunable_parameter_value
	Constant
	Sum
	output_vector

Vector signal and parameter (vector) addition
 	
`for (i=0; i>input_vector_size; i++) {`
`  output_vector[i] = input_vector[i] +`
`    tunable_parameter_vector[i];`
`}`
	 input_vector
	tunable_parameter_vector
	Constant
	Sum
	output_vector

Vector signal and parameter (vector) addition
 	
`for (i=0; i>input_vector_size; i++) {`
`  output_vector[i] = input_vector[i] +`
`    tunable_parameter_vector[i];`
`}`
	

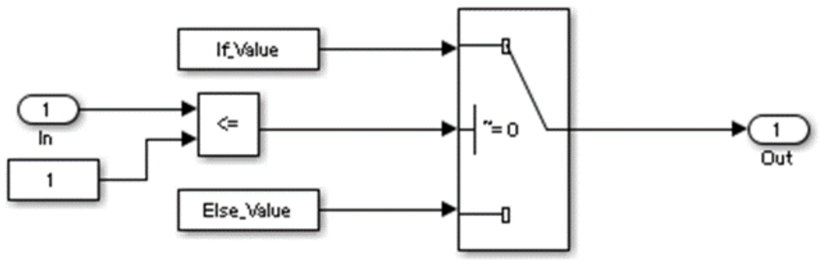
Vector signal element subtraction
 	
`output_signal = 0;`
`for (i=0; i>input_vector_size; i++) {`
`  output_signal = output_signal –`
`    input_vector[i];`
`}`
	

Retention of minimum value/maximum value	 input_signal min MinMax output_signal_min 1/z Unit_Delay input_vector max MinMax output_vector_max 1/z Unit_Delay

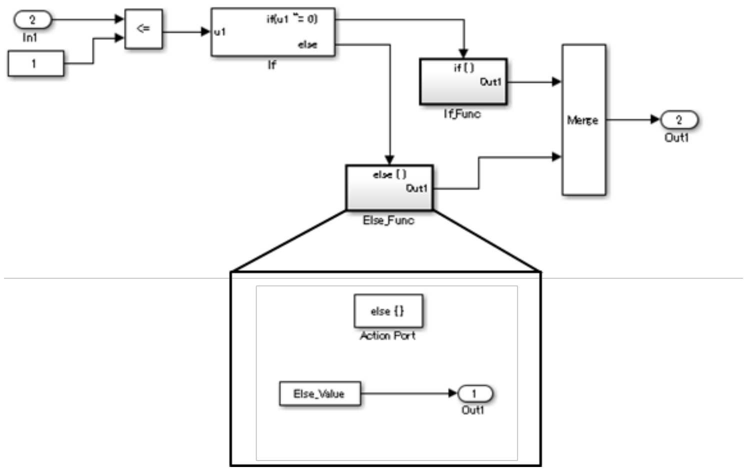
Appendix 5: Using Switch and if-then-else Action Subsystems

[Switch] shall be used for modeling simple if, elseif, else structures when the associated elseif and else actions involve only the assignment of constant values.

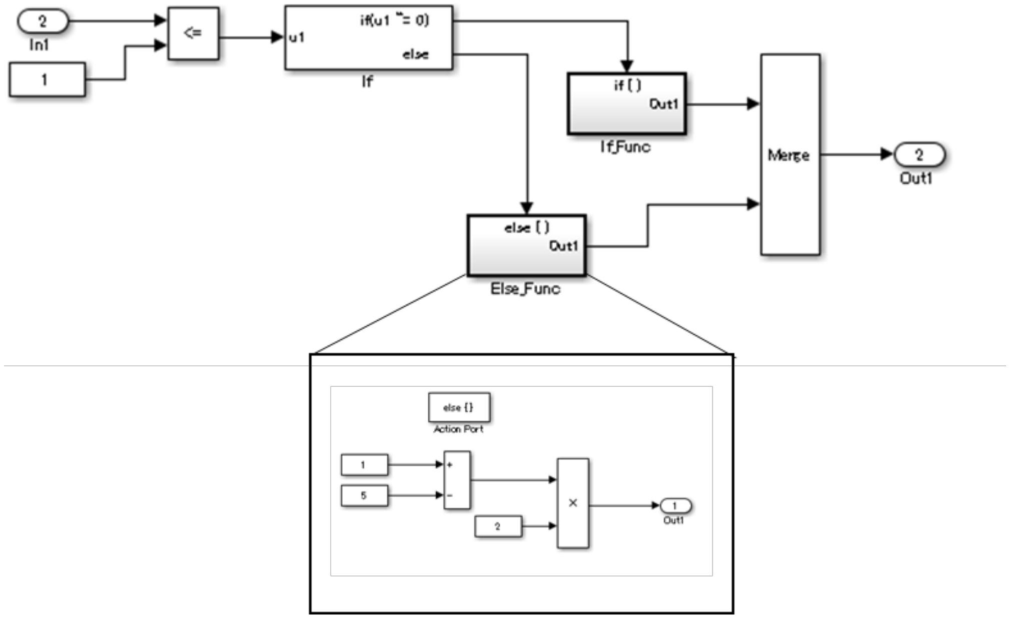
Recommended: For a simple if, elseif, else structure, use [Switch].



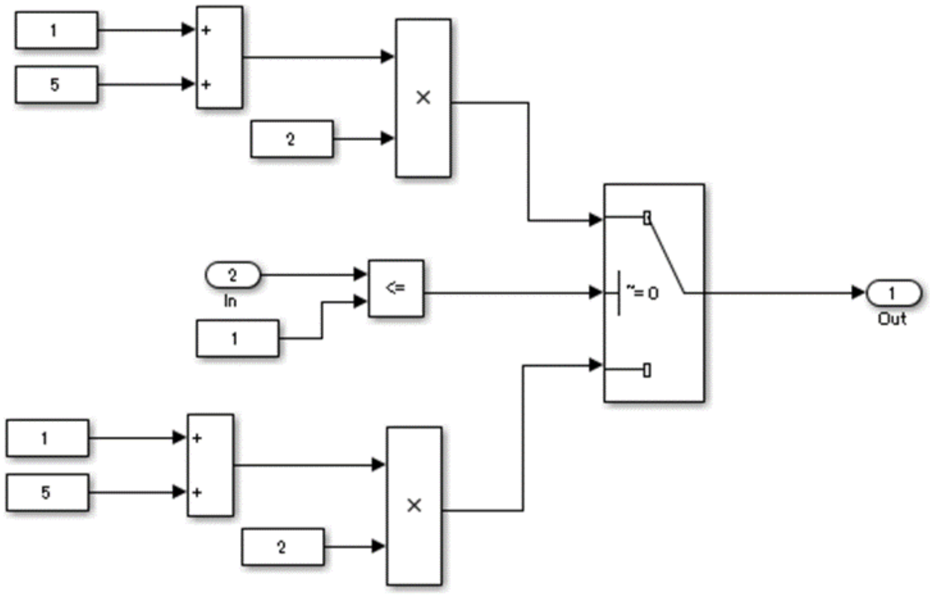
Not recommended: Using [If], [If Action Subsystem] for a simple if, elseif, else structure.



Recommended: For a complex if, elseif, else structure, use [If], [If Action Subsystem].



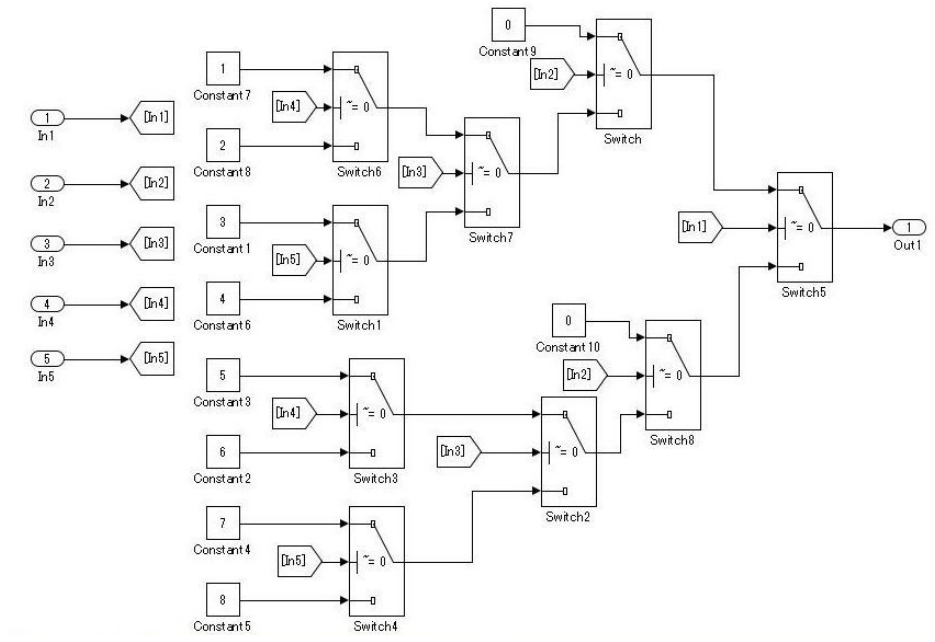
Not recommended: Using [Switch] for a complex if, elseif, else structure.



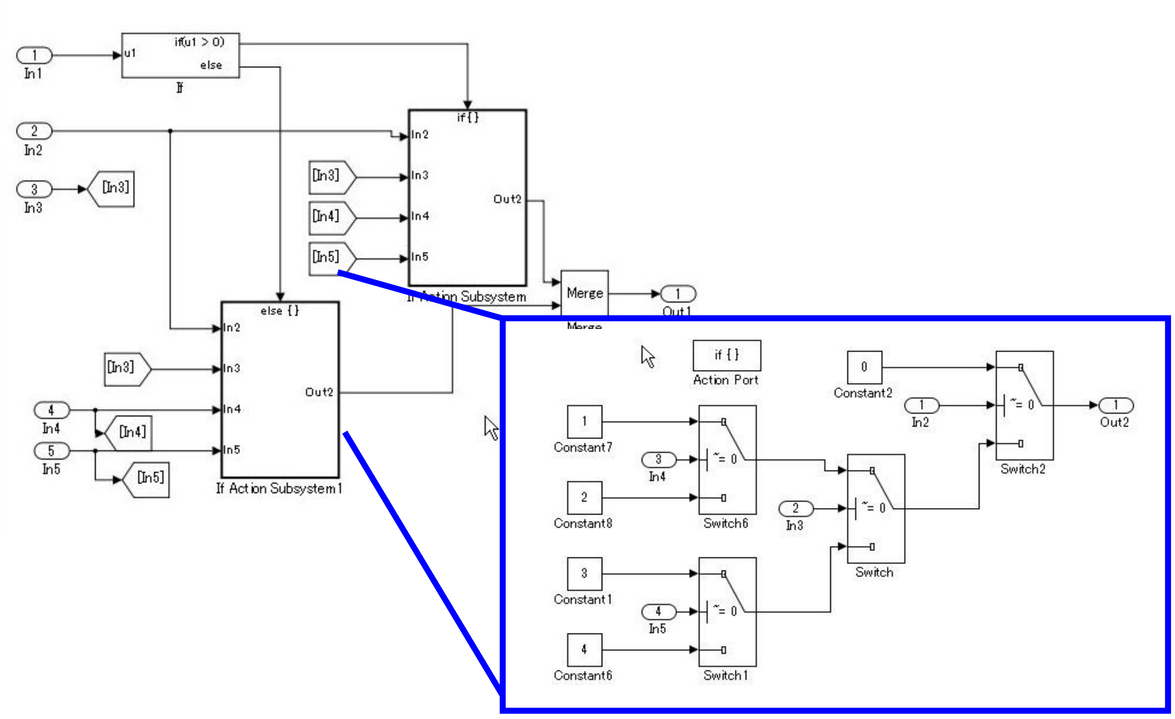
Appendix 6: Use of if, elseif, else Action Subsystem to Replace Multiple Switches

Frequent use of [Switch] 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.

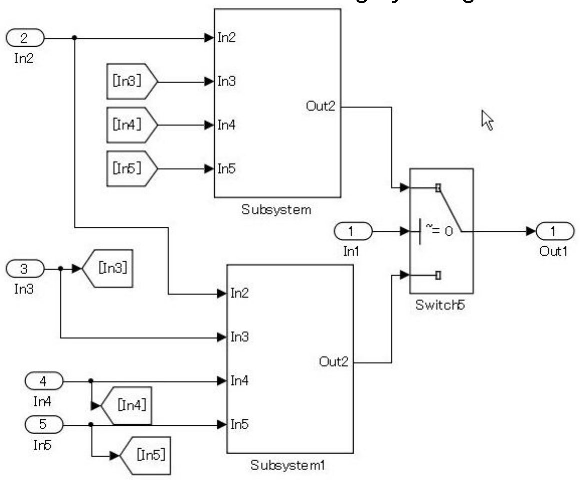
Not recommended: Four levels of nesting.



Recommended: By setting the fourth level as an if Action subsystem, nesting is limited to a single level.



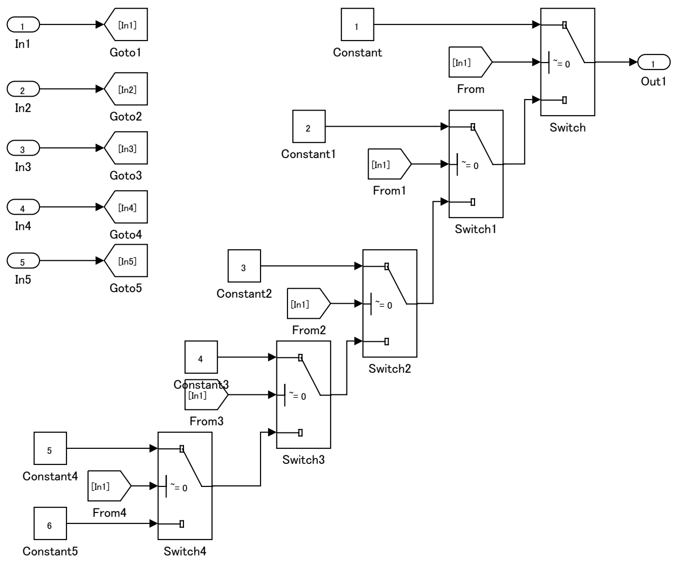
Not recommended: Not dividing by using an if Action subsystem.



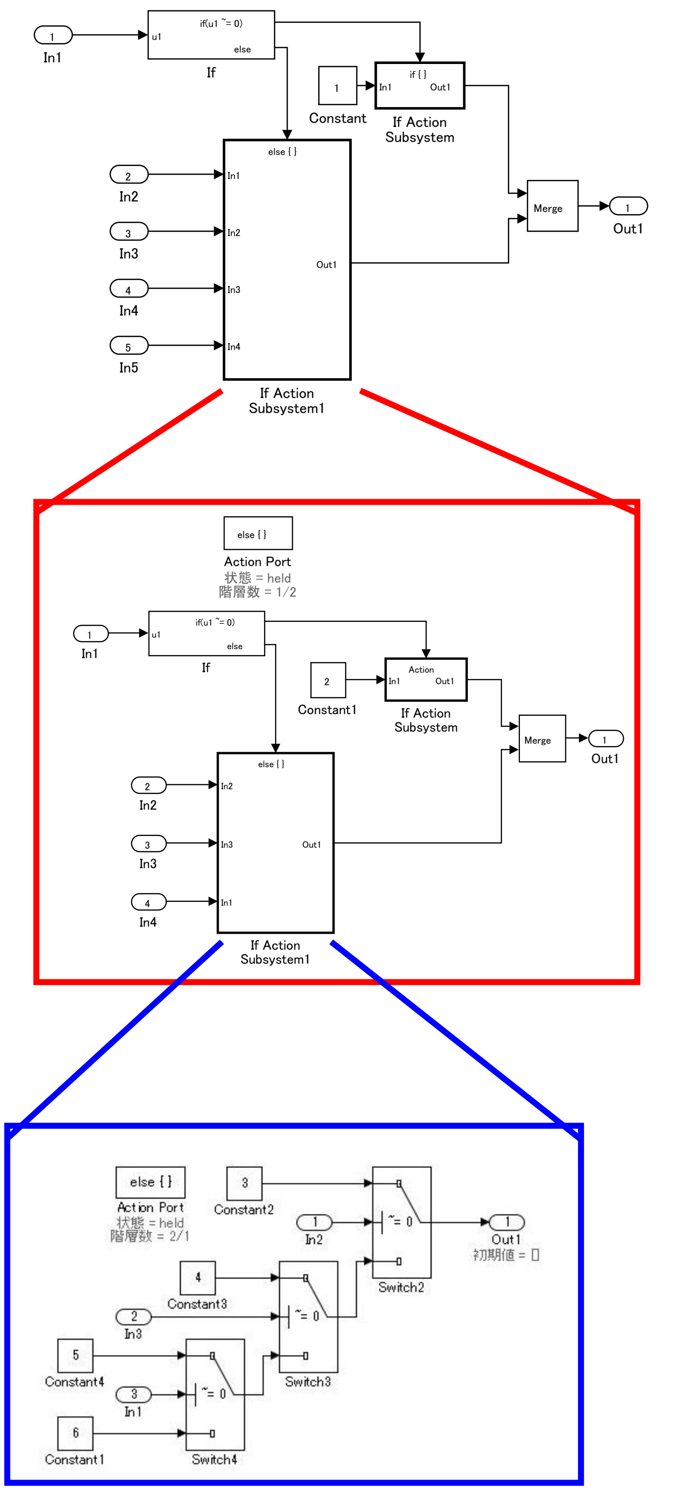
Use atomic subsystem + function setting when the C code limit is applied. In this case, there is no need to use the if, elseif, else Action Subsystem, but the configuration of [Switch] can be split and encapsulated in the subsystem.

Example of model with five levels of nesting.

Not recommended:



Recommended: Use a description method that avoids layering of [Switch] nesting

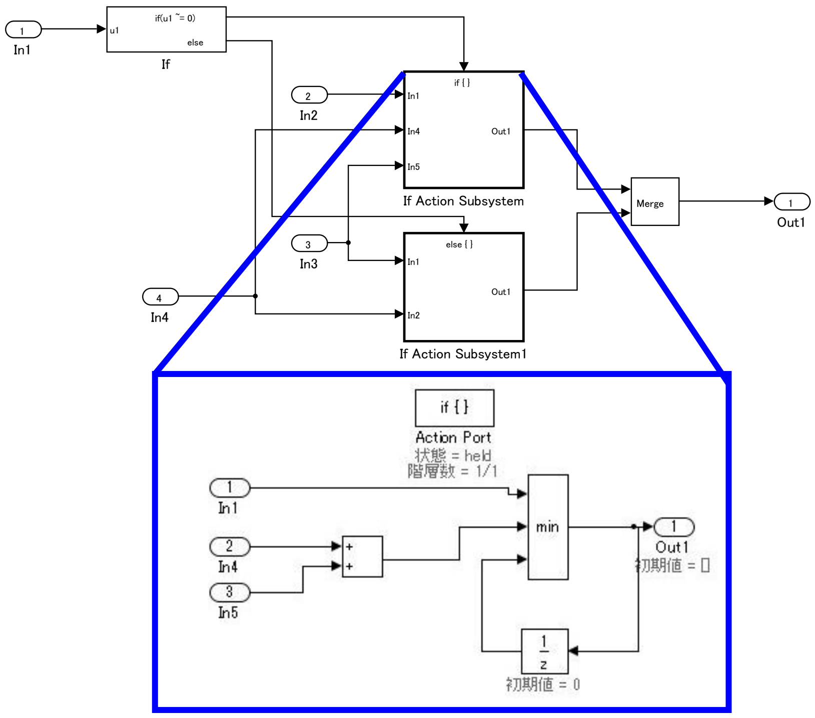


While provided as an example, If Action Subsystem a are not typically used for switching the fixed value. In these Recommended and Not Recommended examples, the generated C code will be the same if the user does not add a function conversion setting. (Confirmed in R2010b to R2013a) The C code is unconstrained.

Appendix 7: Usage Rules for Action Subsystems Using Conditional Control Flow

An If Action subsystem shall not be used when the associated actions do not have a status variable.

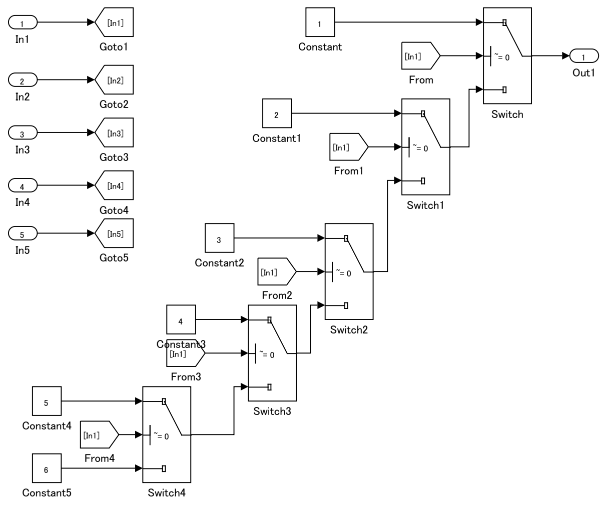
Recommended



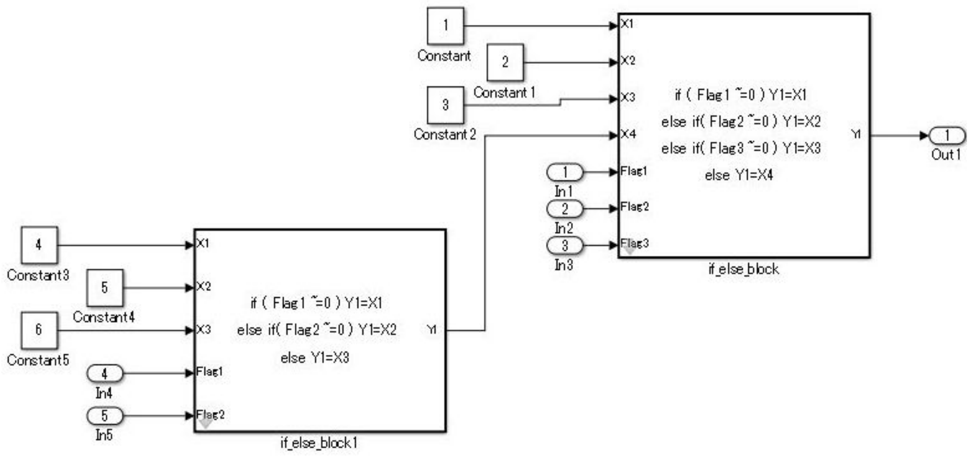
Example of model with 5 levels of nesting

Recommended

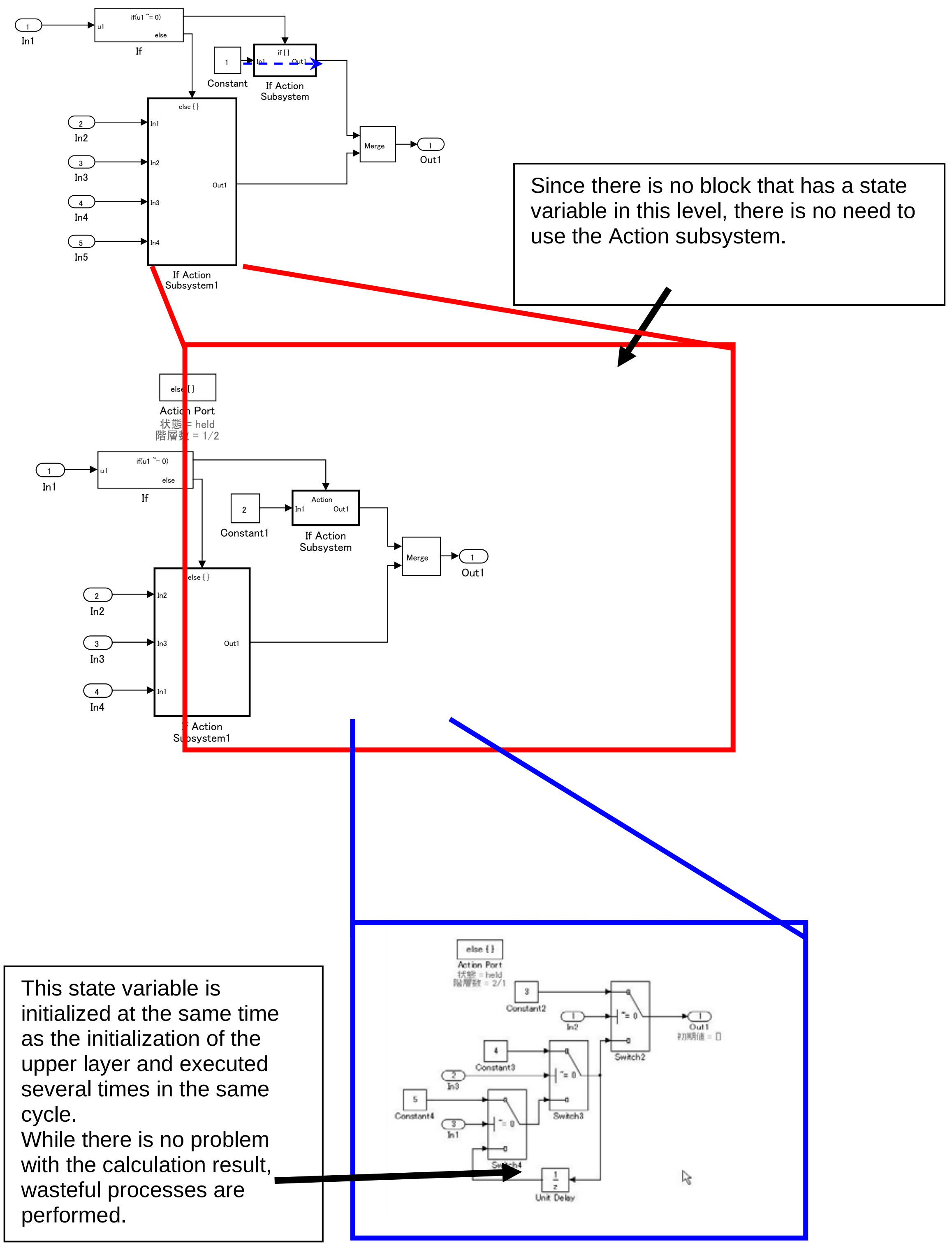
Layering by using a subsystem does not occur because there is no internal state.



Recommended
An atomic subsystem is used to split either side of [Switch] without using an Action subsystem.



Not recommended:
Layering through the use of an unnecessary Action subsystem.



Since there is no block that has a state variable in this level, there is no need to use the Action subsystem.

This state variable is initialized at the same time as the initialization of the upper layer and executed several times in the same cycle.
While there is no problem with the calculation result, wasteful processes are performed.

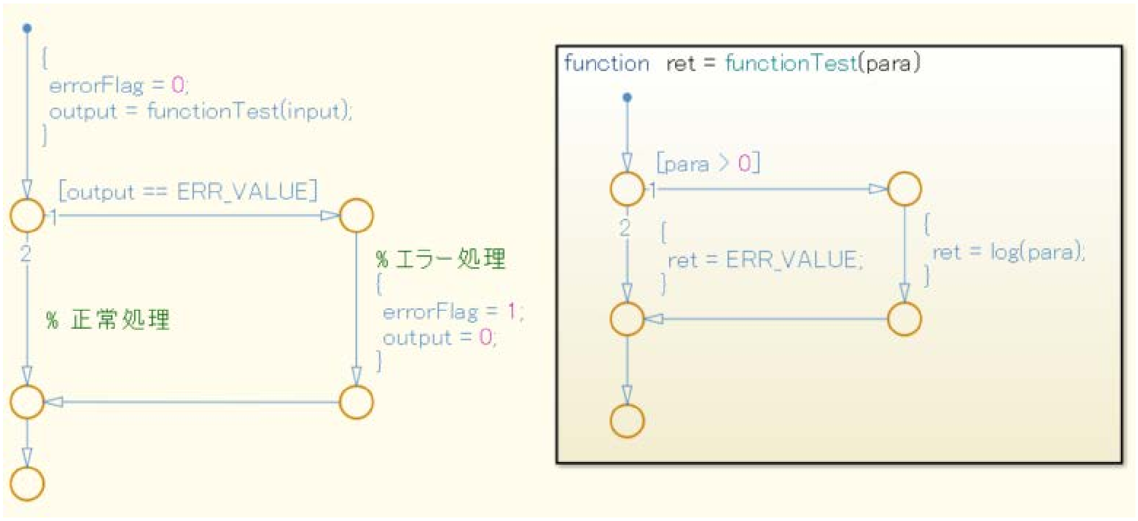
If a function can be achieved by using the Action subsystem, then layering using the Action subsystem is not performed.
In the Not Recommended example, when the lowest level [Unit Delay] on the third level is initialized, the conditional subsystem initialization is first executed one time on the upper 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 [Switch] 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.

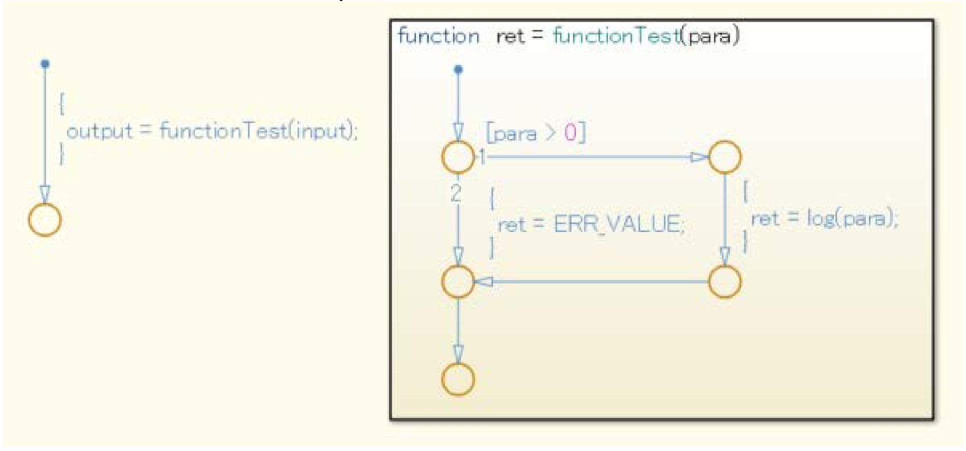
Appendix 8: Tests for Information From Errors

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.

Recommended
Error information is incorporated into the model structure, allowing the user to review and respond to the errors.

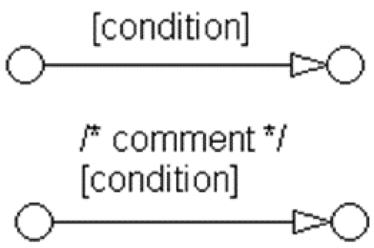
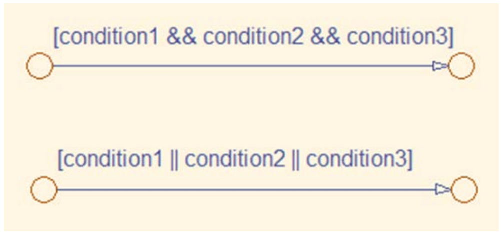


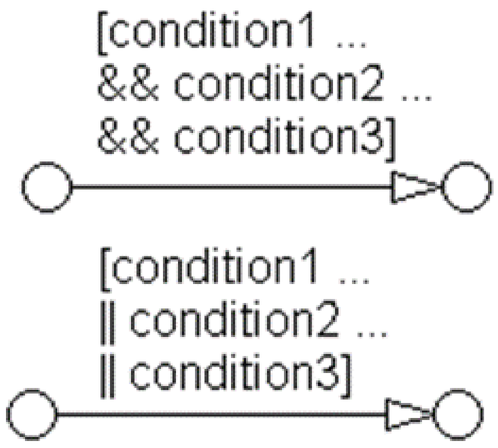
Not recommended:
Error information is not incorporated into the model structure.

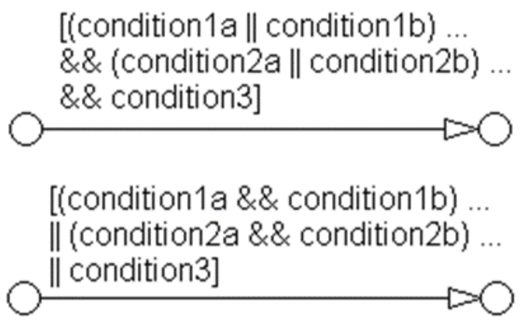

{
output = functionTest(input);
}
function ret = functionTest(para)
[para > 0]
1
2
{
ret = ERR_VALUE;
}
{
ret = log(para);
}

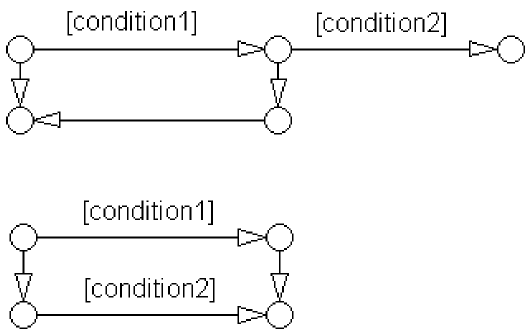
Appendix 9: Flow Chart Patterns for Conditions

These patterns shall be used for conditions within Stateflow flow charts.

Equivalent Functionality	Flow Chart 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 && condition3]* `[condition1 || condition2 || condition3]`	 [condition1 && condition2 && condition3] [condition1 || condition2 || condition3]

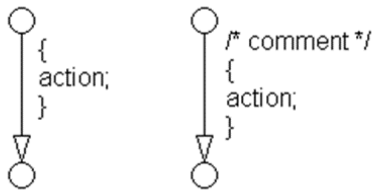
TWO OR MORE CONDITIONS, MULTILINE FORM:
(The use of different logical operators in this form is not allowed. Use sub conditions instead.)
 	
[condition1 ...
&& condition2 ...
&& condition3]
`[condition1 ...`
`|| condition2 ...`
`|| condition3]`
	 [condition1 ...
	&& condition2 ...
	&& condition3]
	[condition1 ...
	|| condition2 ...
	|| condition3]

CONDITIONS WITH SUBCONDITIONS:
(The use of different logical operators to connect sub conditions is not allowed. The use of brackets is mandatory.)
 	
[(condition1a || condition1b) ...
&& (condition2a || condition2b) ...
&& (condition3)]
`[(condition1a && condition1b) ...`
`|| (condition2a && condition2b) ...`
`|| (condition3)]`
	 [(condition1a || condition1b) ...
	&& (condition2a || condition2b) ...
	&& condition3]
	[(condition1a && condition1b) ...
	|| (condition2a && condition2b) ...
	|| condition3]

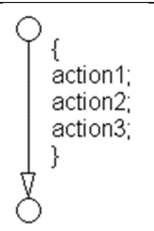
CONDITIONS THAT ARE VISUALLY SEPARATED: (This form can be combined with the preceding patterns.) *[condition1 && condition2]* [condition1 || condition2]	 [condition1] [condition2] [condition1] [condition2]

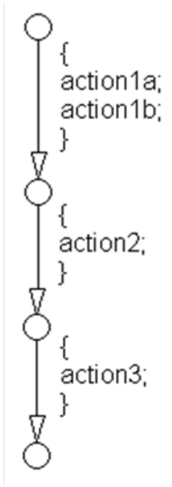
Appendix 10: Flow Chart Patterns for Condition Actions

These patterns shall be used for condition actions within Stateflow flow charts.

Equivalent Functionality	Flow Chart Pattern
ONE CONDITION ACTION: `action;`	

TWO OR MORE CONDITION ACTIONS, MULTILINE FORM:
(Two or more condition actions in one line are not allowed.)
`action1; ...`
`action2; ...`	

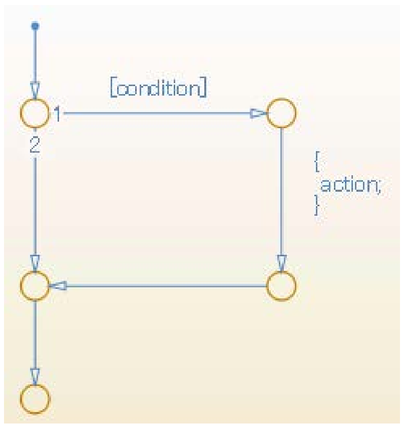
`action3; ...`	

CONDITION ACTIONS, WHICH ARE VISUALLY SEPARATED:
(This form can be combined with the preceding patterns.)
action1a;
action1b;
action2;
`action3;`
	 {
	action1a;
	action1b;
	}
	{
	action2;
	}
	{
	action3;
	}

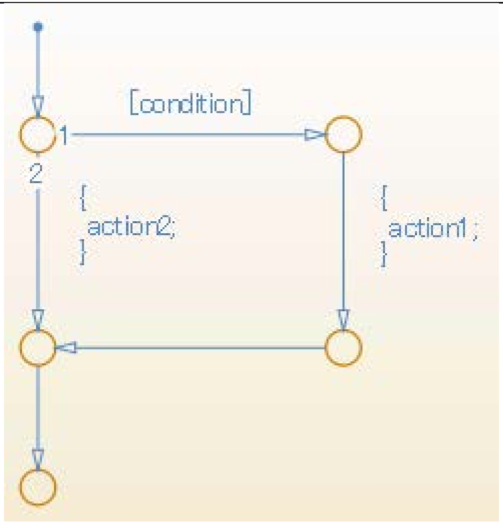
Appendix 11: Flow Chart Patterns for if Constructs

These patterns shall be used for If constructs within Stateflow flow charts.

Function	Flow Chart Pattern

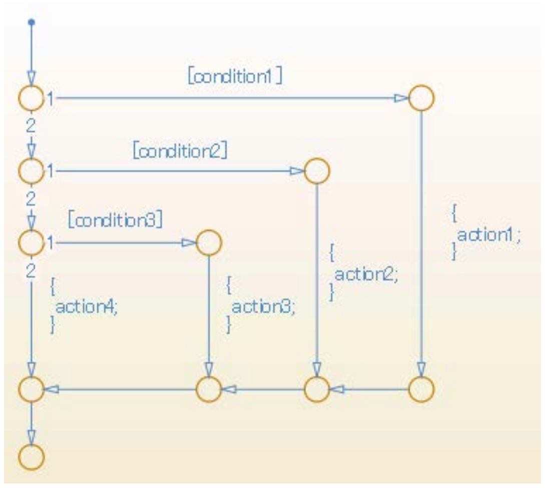
If construct
 	
`if (condition){`
`  action;`
`}`
	

If, else construct
 	
`if (condition) {`
`  action1;`
`}`
`else {`
`  action2;`
`}`	



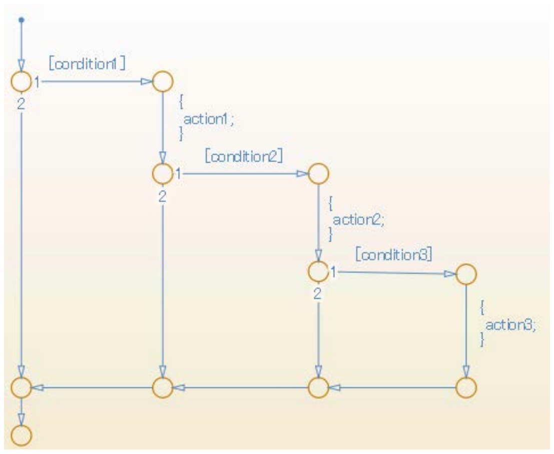
If, elseif, else construct

```
if (condition1) {
  action1;
}
else if (condition2) {
  action2;
}
else if (condition3) {
  action3;
}
else {
  action4;
}
```



Cascade of if construct

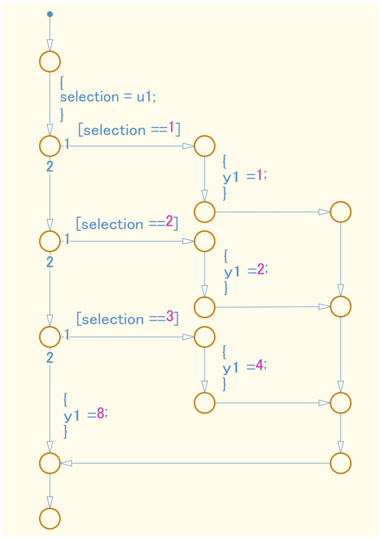
```
if (condition1) {
  action1;
  if (condition2) {
    action2;
    if (condition3) {
      action3;
    }
  }
}
```

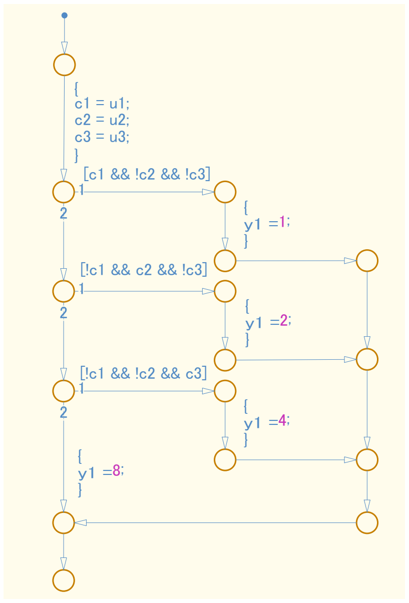


Appendix 12: Flow Chart Patterns for Case Constructs

These patterns shall be used for case constructs in Stateflow flow charts.

Function	Simulink pattern

Case construct with exclusive selection
 	
`selection = u1;`
`switch (selection) {`
`  case 1:`
`    y1 = 1;`
`    break;`
`  case 2:`
`    y1 = 2;`
`    break;`
`  case 3:`
`    y1 = 4;`
`    break;`
`  default:`
`    y1 = 8;`
`}`
	 {
	selection = u1;
	}
	[selection ==1]
	1
	2
	{
	y1 =1;
	}
	[selection ==2]
	1
	2
	{
	y1 =2;
	}
	[selection ==3]
	1
	2
	{
	y1 =4;
	}
	{
	y1 =8;
	}

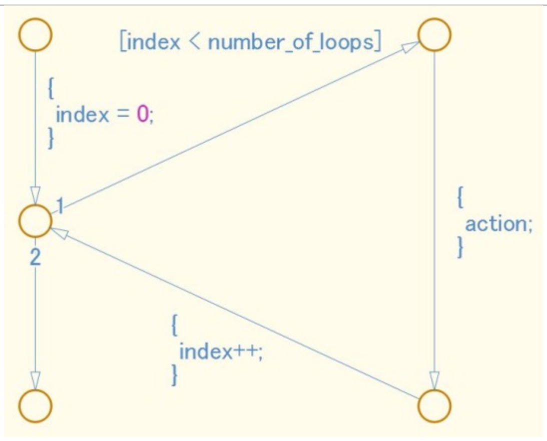
Case construct with exclusive conditions
 	
`c1 = u1;`
`c2 = u2;`
`c3 = u3;`
`if (c1 && ! c2 && ! c3) {`
`  y1 = 1;`
`}`
`elseif (! c1 && c2 && ! c3) {`
`  y1 = 2;`
`}`
`elseif (! c1 && ! c2 && c3) {`
`  y1 = 4;`
`}`
`else {`
`  y1 = 8;`
`}`
	 {
	c1 = u1;
	c2 = u2;
	c3 = u3;
	}
	[c1 && !c2 && !c3]
	1
	2
	{
	y1 =1;
	}
	[!c1 && c2 && !c3]
	1
	2
	{
	y1 =2;
	}
	[!c1 && !c2 && c3]
	1
	2
	{
	y1 =4;
	}
	{
	y1 =8;
	}

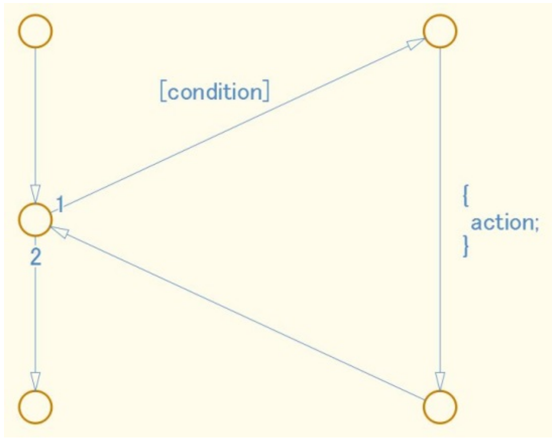
Appendix 13: Flow Chart Patterns for Loop Constructs

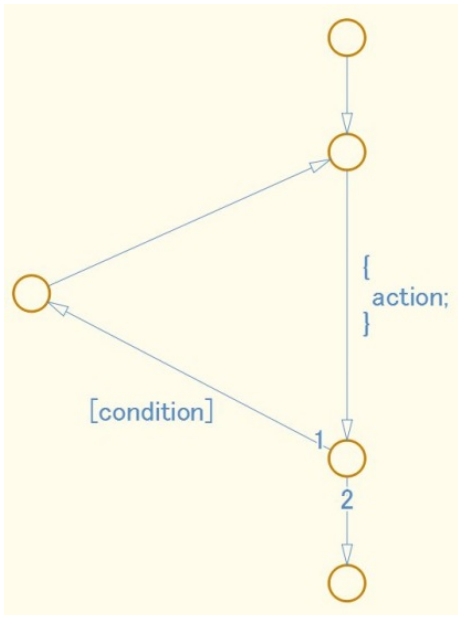
These patterns shall be used to create loop constructs in Stateflow flow charts.

Function	Flow Chart Pattern

For loop construct
 	
`for ( index = 0;`
`      index < number_of_loops;`
`      index++ )`
`{`
`  action;`	

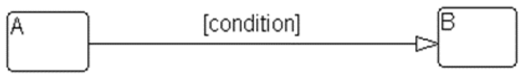
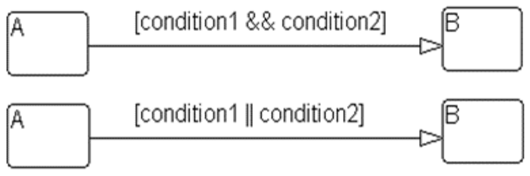
`}`	 [index < number_of_loops]
	{
	index = 0;
	}
	1
	2
	{
	action;
	}
	{
	index++;
	}

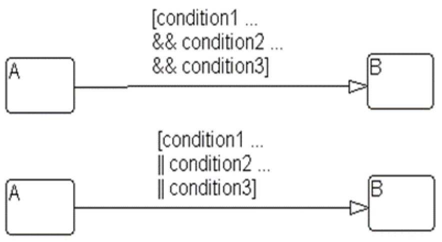
While loop construct
 	
`while ( condition )`
`{`
`  action;`
`}`
	

Do while loop construct
 	
`do`
`{`
`  action;`
`}`
`while ( condition )`
	

Appendix 14: State Machine Patterns for Conditions

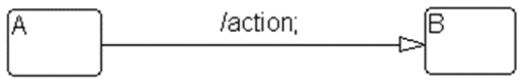
These patterns shall be used for conditions within Stateflow state machines.

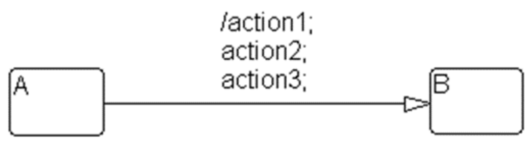
Equivalent Functionality	State Machine Pattern
ONE CONDITION: `(condition)`	
UP TO THREE CONDITIONS, SHORT FORM: (The use of different logical operators in this form is not allowed, use sub conditions instead) *(condition1 && condition2)* `(condition1 || condition2)`	 A [condition1 && condition2] B A [condition1 || condition2] B

TWO OR MORE CONDITIONS, MULTILINE FORM:
A sub condition 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)`
	 [condition1 ...
	&& condition2 ...
	&& condition3]
	A
	B
	[condition1 ...
	|| condition2 ...
	|| condition3]
	A
	B

Appendix 15: State Machine Patterns for Transition Actions

These patterns shall be used for transition actions within Stateflow state machines.

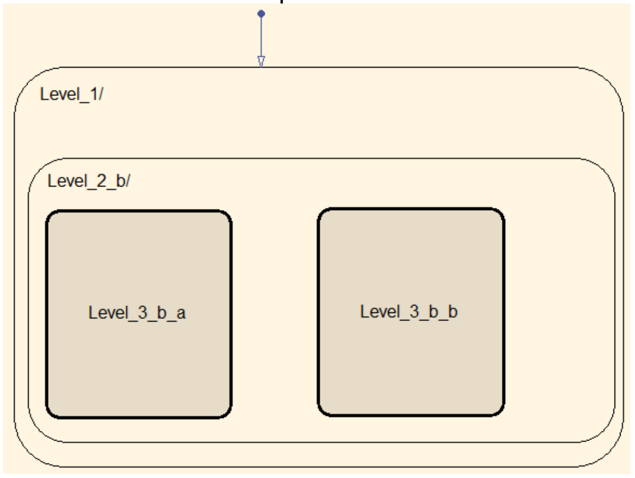
Equivalent Functionality	State Machine Pattern
ONE TRANSITION ACTION: `action;`	

TWO OR MORE TRANSITION ACTIONS, MULTILINE FORM:
(Two or more transition actions in one line are not allowed.)
 	
`action1;`
`action2;`
`action3;`
	

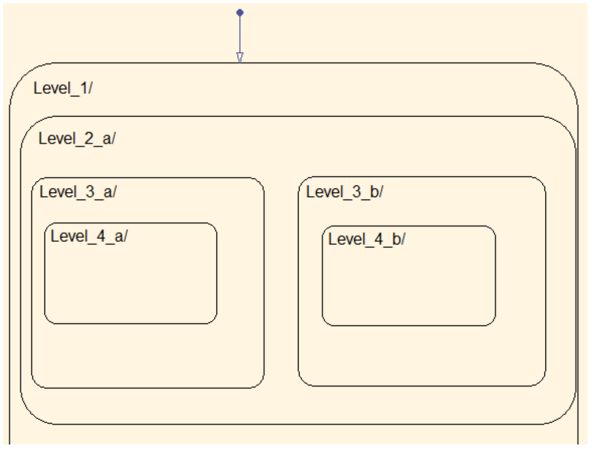
Appendix 16: Limiting State Layering

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.

Recommended
The fourth level is encapsulated in a subchart.

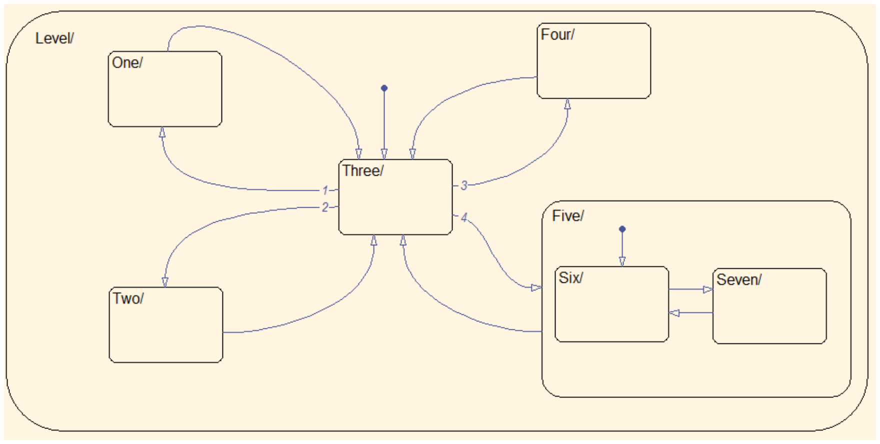


Not recommended:
The constraint goal is set to three levels, but Level_4_a and Level_4_b have more than three levels and are nested.



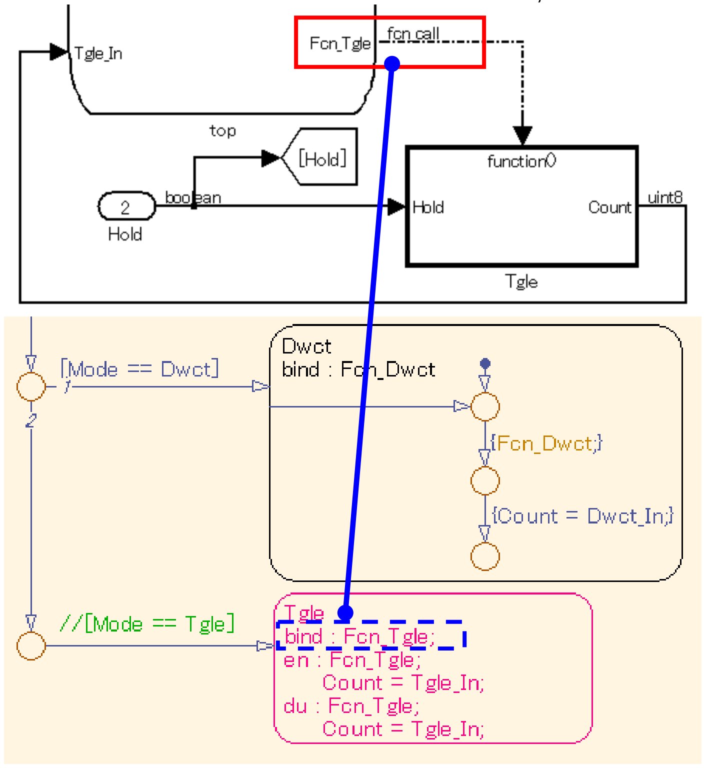
Appendix 17: Number of States per Stateflow Container

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.



Appendix 18: 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.



Appendix 19: Function Types Available in Stateflow

The functions types used in Stateflow shall be dependent on the required processing.

For graphical functions, use:

- o If, elseif, else logic

For Simulink functions, use:

- o Transfer functions
- o Integrators
- o Table look-ups

For MATLAB functions, use:

- o Complex equations
- o If, elseif, else logic