CONTROL ALGORITHM MODELING GUIDELINES USING MATLAB®, Simulink®, and Stateflow®

Version 4.01（English edition）

Japan MBD Automotive Advisory Board (JMAAB)

31-Mar- 2015

（correct 19-Jun-2015）

■ Copyright

- The copyright of this document belongs to JMAAB.
- JMAAB provides no guarantees with regard to the contents of this document. JMAAB shall not be liable for any failures which occur as a result of using this document. Please note that the information within this document is subject to change or removal without notice.

■ Handling this document

- This document may be reproduced only for internal use and non-commercial purposes. In addition, when quoting from this document, state explicitly that the quote comes from this document, and include the name of the author, Title etc., in accordance with the requirements for citation.
- Please refer to the JMAAB website for any information regarding this deliverable (http://jmaab.mathworks.jp/).
- For any other inquiries, please contact the JMAAB office (jmaab-office@mathworks.co.jp).

■ Please note:

- This document is English edition of “CONTROL ALGORITHM MODELING GUIDELINES USING MATLAB®, Simulink® and Stateflow® Version 4.0”.
- There were differences between Japanese edition and English edition until Version 3.0. However, they were fixed in Version 4.0. Thus, there may be the case that the even the same rules are different from the past version.

TABLE OF CONTENTS

1. Introduction

1.1. Purpose of these Guidelines

These guidelines stipulate important basic rules for describing Simulink / Stateflow models to allow for a simple, common understanding by authors and users in operating automotive control system of control models.

They were created with the following main objectives.

- Readability
  - Improvement of graphical understandability
  - Improvement of readability of functional analysis.
  - Prevention of connection mistake
  - Comments and so on
- Simulation and verification
  - System to enable simulation
  - Easy testing
- Code generation
  - Improvement of efficiency of generation code.(ROM,RAM efficiency)
  - Securement of robustness of a generation code
- Others

1.2. Guideline template

Guideline descriptions are documented using the following template. Use of this template is also recommended when creating original guidelines.

ID: Title	**XX_nnnn: Title of the guideline (unique, short)**
Priority	One of Mandatory / Strongly Recommended / recommended.
Scope	MAAB / NAMAAB / JMAAB / company name (if adding company rules)
MATLAB Version	ALL RX, RY, RZ RX and later RX and earlier RX through RY
Prerequisites	Links to guidelines, which are prerequisite to this guideline (ID + Title)
Description	Description of the guideline (text, images).
Notes	Notes, footnotes.
See also	ID including other helpful guidelines.
Last Change	Version number of the Last Change.

Note: This template lists the minimum requirements for a correct understanding of a guideline. New items may be added to the template as long as they do not duplicate any of the existing items.

1.2.1. ID

An ID consists of 2 lower case letters (identifying the guideline author) and a 4 digit number, separated by an underscore. An ID is permanent and cannot be changed, and is used when referring to a guideline.

db, jm, hd, ar are IDs used by established members for Ver1.0. **na**, **jp**, **jc**, **jt** are IDs used from Ver2.0 onwards.

Please use letter combinations other than these as ID when adding your own guidelines.

Parenthesized rules, (ID), are rules that have been changed from rules to a document description. These document description rules have, like other document descriptions, no priority or scope classification. They describe valuable approaches, examples for the creation of models. They have no rules that must be specifically adhered to, or counterexamples, but describe a particular approach or helpful tips.

1.2.2. Title

The Title is unique and is a brief description of the guidelines.

1.2.3. Priority

The priority level is classified as "Mandatory", "Strongly Recommended", and "Recommended". Priority does not only indicate the importance of the guideline, but also considers the gravity of the potential results if they are violated.

Mandatory	Strongly Recommended	Recommended
DEFINITION		
• Guidelines that all companies agree to that are absolutely essential • Guidelines that all companies conform to 100%	• Guidelines that are agreed upon to be a good practice, but legacy models preclude a company from conforming to the guideline 100% • Models should conform to these guidelines to the greatest extent possible; however 100% compliance is not required	• Guidelines that are recommended to improve the appearance of the model diagram, but are not critical to running the model • Guidelines where conformance is preferred, but not required
CONSEQUENCES If the guideline is violated		
• Essential items are missing • The model might not work properly	• The quality and the appearance deteriorates • There may be an adverse effect on maintainability, portability, and reusability	• The appearance will not conform with other projects
WAIVER POLICY If the guideline is intentionally ignored,		
• The reasons must be documented		

1.2.4. Scope :

The scope of a guidelines is set to one of the following:

- MAAB: Guideline that has been agreed by JMAAB and NAMAAB.
- JMAAB: Guideline that has been agreed by the Japan MBD Automotive Advisory Board alone.
- NAMAAB: Guideline that has been agreed by the North America MATLAB Automotive Advisory Board alone.

MAAB includes the subgroups JMAAB and NAMAAB.
"JMAAB" is a subgroup including automotive manufacturers and suppliers in Japan.
"NAMAAB" is a subgroup including automotive manufacturers and suppliers in the United States and Europe.

1.2.5. MATLAB version

The guidelines support all MATLAB versions, but some guidelines only support specific versions. The version information is given in one of the following 5 formats.

- ALL: all MATLAB versions.
- RX, RY, RZ: specific MATLAB versions.
- before RX: MATLAB versions before RX.
- after RX: MATLAB versions after RX.
- RX through RY: MATLAB versions for RX through RY.

Ver4.0 contains rules for R2008b through R2013a.

1.2.6. Prerequisites

The Prerequisites entry gives the ID and Title for the guidelines that are prerequisite to this guideline.

1.2.7. Description

The Description describes the content in detail, using figures and tables.

1.2.8. See Also

This field contains guideline IDs of other helpful guidelines.
Apart from the MAAB guidelines, the following guidelines are referred to.

- Modeling Guidelines for Code Generation(cgsl_)
- Modeling Guidelines for High-Integrity Systems(hisl_)
- NASA Orion Style Guidelines numbers from Orion GN&C MATLAB/Simulink Standards(Orion_[bn_,ek_,im_,jr_,jh])Ver3.0 are added as related references.
  http: //www.mathworks.co.jp/aerospace-defense/standards/nasa.html
- MISRA SLSF Guidelines (MISRA AC SLSF_)
  From Ver4.0, MISRA AC SLSF Guidelines, published by MISRA, are added as related references.

The content contained in these guidelines are not included in the text of this document.
The content of these guidelines and the content in the guidelines listed above may vary.
The ultimately correct rules are the MAAB rules, describing the required rules for controller modeling.
They do not correspond to all numbers for the guidelines listed above.

1.2.9. Last Change

This field contains the version number for the Last Change.
However, a version number is not changed for simply printing error corrections or additional explanations.
It lists a modified version which includes changes to the intention of rules, changes in conditions or additional conditions.

1.3. Organization of these Guidelines

Explanation of this document is described in chapter 1.
Rules are described in from chapter 2 to chapter 6.
Where rules for prohibited use and limited or restricted use with regard to specific blocks or functions conflict, list the rules for prohibited use first. Then list the rules for limited use.

1. Prohibited use rule : Recommended
2. Limited / restricted use rule : Strongly Recommended (or Mandatory)

This explanation concerns the listing order. Investigate the adoption of these two rules for the operation procedures.

Rationals of rules establishment and adjustable parameters of rules are listed in chapter 7.
Of the term as for 8 chapters for beginners comment.
Chapters 9 to 11 the model architecture and operation required by advanced users .
Change history of these guidelines is described in chapter 12.

2. Naming Conventions

2.1. Naming Conventions - Overall summary

2.1.1. Rule IDs for characters that can be used in names

Character restrictions and characters that can be used in names are described in the following rules.

ar_0001: Usable characters for file names
ar_0002: Usable characters for folder names
jc_0201: Usable characters for Subsystem name
jc_0211: Usable characters for Inport block and Outport block
jc_0222: Usable characters for signal line and bus names
jc_0232: Usable characters for parameter names
jc_0231: Usable characters for block names

2.1.2. Rule IDs for character length

Limitations relating to the length of name lengths are described in the following rules.

jc_0241: Length restrictions for file names
jc_0242: Length restrictions for folder names
jc_0243: Length restrictions for Subsystem names
jc_0244: Length restrictions for Inport and Outport names
jc_0245: Length restrictions for signal and bus names
jc_0246: Length restrictions for parameter names
jc_0247: Length restrictions for block names

2.1.3. List of naming rule constraints "character type / character length"

<table>
<tr><td>Availability for use by character type</td><td>File name, folder</td><td>Subsystem the code is generated for, Inport/Outport, signal name, bus name, parameter name</td><td>Other blocks</td></tr>
<tr><td>single-byte alphabetic character</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>single-byte numerical character</td><td colspan="3">○ not allowed as the first character, otherwise allowed</td></tr>
<tr><td>single-byte underscore</td><td colspan="3">○ not allowed for first or last character, no two underscores in succession</td></tr>
<tr><td>single-byte space</td><td colspan="2">not allowed</td><td rowspan="2">○
not allowed for first or last character, no two spaces in succession</td></tr>
<tr><td>line break</td><td colspan="2">not allowed</td></tr>
<tr><td>other characters (local language)</td><td>×</td><td>×</td><td>×</td></tr>
<tr><td>Length limitations</td><td>File name, folder name</td><td>Subsystem the code is generated for, Inport/Outport, signal name, bus name, parameter name</td><td>Other blocks</td></tr>
<tr><td>character length</td><td colspan="2">3 to 63 characters (example)</td><td>～63 characters</td></tr>
</table>

2.2. General Rules

2.2.1. ar_0001: Usable characters for file names

ID: Title	ar_0001: Usable characters for file names

Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	File names are subject to the following constraints. **Subject of Applications** Please operate by determining extention to be subject of application of the rules. When application of this rule is limited to model names, the 2 types are [mdl] and [slx] **Valid form** filename = name.extension • name: may not start with a numerical character, no spaces,no any MATLAB Keywords. • extension: no spaces **Uniqueness** • None of the file names in a new project folder may be duplicates. There may be no identically named models, including in subfolders via a MATLAB path. **Usable characters** Name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ Extension: (Extensions are determined individually for used tools.) a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 **Underscores** Name: • Underscores may be used to separate words • Underscores may not be used in succession • Underscores may not be used as the first character • Underscores may not be used as the last character Extension: (Extensions are determined individually for used tools.) Underscores may not be used
Notes	Occasions when both test1.slx and test1.m exist. When running test1 by command line, test1.m is not run and the test1.slx model file can open. In other words, constants described in test1.m cannot be loaded into the MATLAB workspace. If there are model files with identical names in a folder without a path, please use switching the path according to operation.
Last Change	V4.0

2.2.2. ar_0002: Usable characters for folder names

ID: Title	**ar_0002: Usable characters for folder names**
Priority	Recommended
Scope	MAAB

MATLAB Version	ALL
Prerequisites	
Description	A folder name conforms to the following constraints: **Valid form** directory name =name name: may not start with a numerical character, no spaces **Usable characters** name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ **Underscores** name: • can use underscores to separate words • cannot have more than one consecutive underscore • cannot start with an underscore • cannot end with an underscore
Notes	There is no problem even if same folder names are included into path.. (No need of identity.) As of R2013b, even if local language is used in folder name, C source code can be generated.
Last Change	V4.0

2.2.3. jc_0241: Length restrictions for file names

ID: Title	**jc_0241: Length restrictions for file names**
Priority	Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	File names should be made up of 3 to 63 characters (not including dots and extension).
Notes	Past versions limited the number of characters to 63 for model referencing.
See Also	
Last Change	V4.0

2.2.4. jc_0242: Length restrictions for folder names

ID: Title	**jc_0242: Length restrictions for folder names**
Priority	Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	Folder names on every level of a model should be made up of 3 to 63 characters.
Notes	It is better to restrict the overall number of folder characters (full path name). Long full path names may lead to problems such as incomplete display of the path name in

	the GUI that is used for the project.
Last Change	V4.0

2.3. Internal model rules

2.3.1. jc_0201: Usable characters for Subsystem names

ID: Title	jc_0201**: Usable characters for Subsystem names**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	The names of all subsystem blocks should conform to the following constraints: **Valid form** name: • should not start with a number • should not have blank spaces • should not have carriage returns **Usable characters** name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ **Underscores** name: • can be used to separate words • cannot have more than one consecutive underscore • cannot start with an underscore • cannot end with an underscore
Notes	Subsystems subject to this are subsystems subject to code generation. Subsystems (Model-Wide Utilities/Model Info etc.) that have no Input/Output ports are classified in the annotations, and therefore not subject to this rule. Also check whether function names for code generation will be subject to this rule.
Last Change	V2.2

2.3.2. jc_0211: Usable characters for Inport block and Outport block

ID: Title	jc_0211**: Usable characters for Inport block and Outport block**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	The names of all Inport blocks and Outport blocks should conform to the following constraints: **Valid form** name: • may not start with a numerical character • no spaces • may not include line breaks

	Usable characters name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ **Underscores** name: • Underscores may be used to separate words • Underscores may not be used in succession • Underscores may not be used as the first character • Underscores may not be used as the last character
Last Change	V2.2

2.3.3. jc_0222: Usable characters for signal line and bus names

ID: Title	jc_0222**: Usable characters for signal line and bus names**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	Indicates the constraints on signals with a name. Valid form name: • may not start with a numerical character • no spaces • no control characters • may not include line breaks Usable characters name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ Underscores name: • Underscores may be used to separate words • Underscores may not be used in succession • Underscores may not be used as the first character • Underscores may not be used as the last character
Notes	The naming convention for signal lines does not differentiate between signal line type (scalars, vectors, busses).
Last Change	V4.0

2.3.4. jc_0232: Usable characters for parameter names

ID: Title	**jc_0232: Usable characters for parameter names**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	

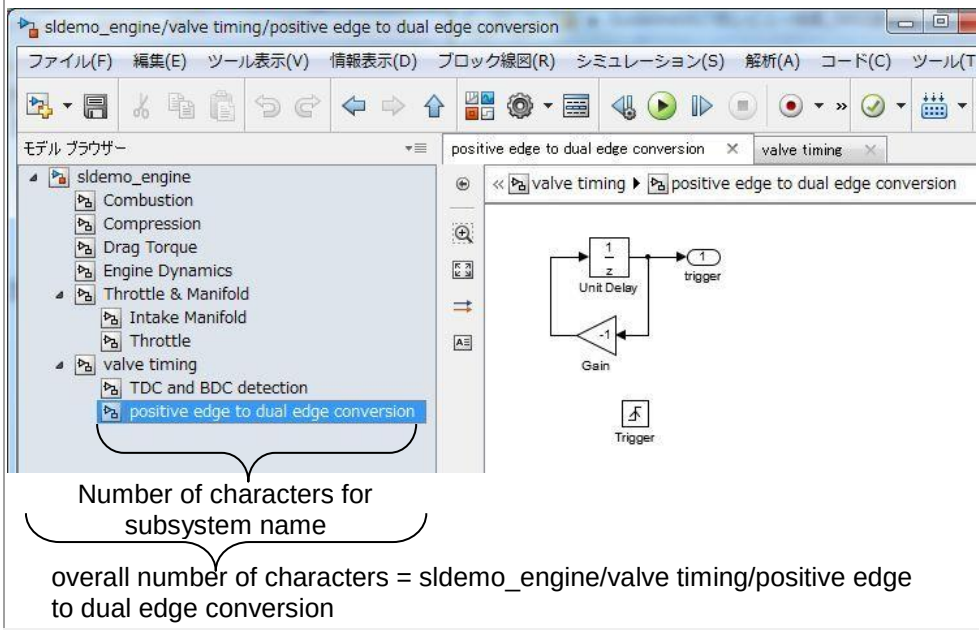
Description	Indicates the constraints on signals with a name. Valid form name: • may not start with a numerical character • no spaces • no control characters • may not include line breaks Usable characters name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _ Underscores name: • Underscores may be used to separate words • Underscores may not be used in succession • Underscores may not be used as the first character • Underscores may not be used as the last character
Last Change	V4.0

2.3.5. jc_0231: Usable characters for block names

ID: Title	**jc_0231: Usable characters for block names**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	**jc_0201: Usable characters for subsystem names**
Description	All block names are subject to the following constraints. Valid form name: • should not start with a number • should not have blank spaces • should not include double-byte characters • can have carriage returns Usable characters name: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 _
Notes	This rule does not apply to subsystem blocks, Inport/Outport blocks.
Last Change	V2.0

2.3.6. jc_0243: Length restrictions for subsystem names

ID: Title	**jc_0243: Length restrictions for subsystem names**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL

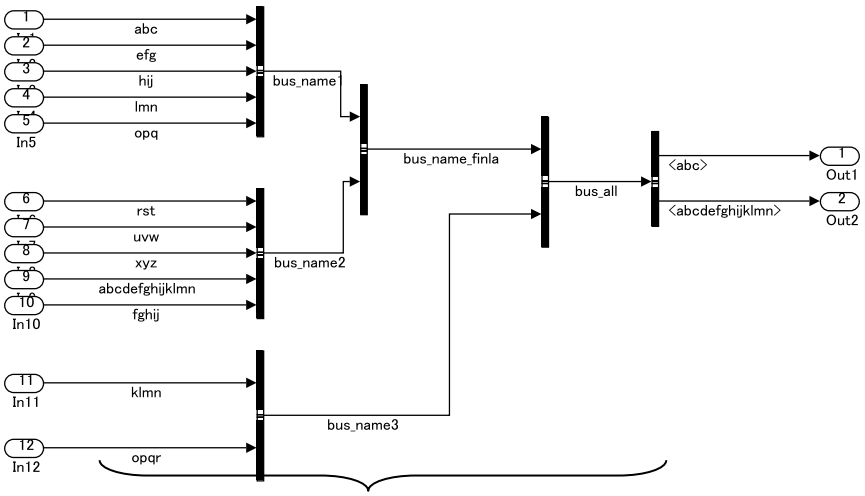
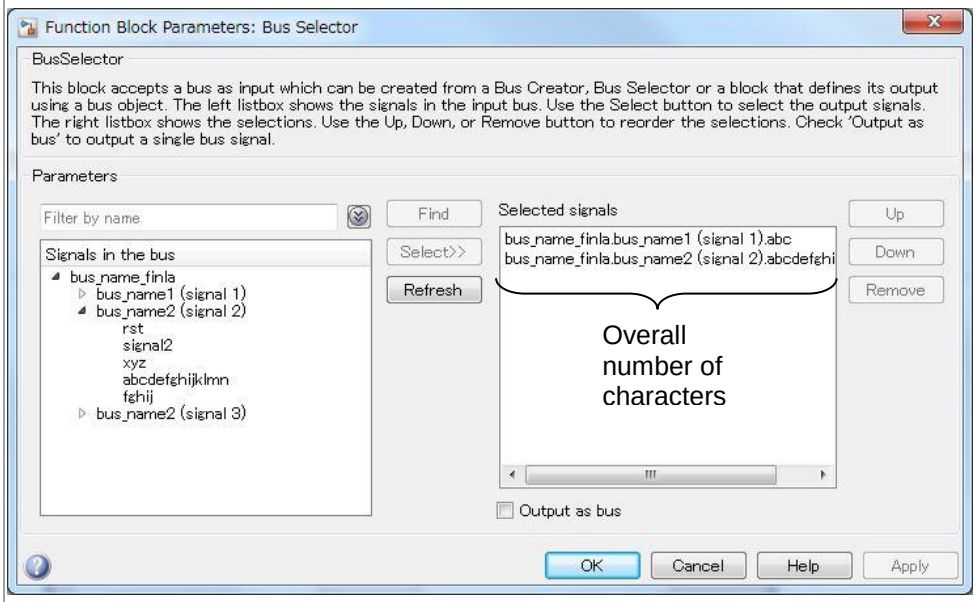
Prerequisites	
Description	Subsystem name lengths should be 3 to 63 characters. 
Notes	It is better to restrict the overall number of characters (full path name including model name) too.
See Also	
Last Change	V4.0

2.3.7. jc_0244: Length restrictions for Inport and Outport names

ID: Title	**jc_0244: Length restrictions for Inport and Outport names**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	Port name lengths should be 3 to 63 characters.
See Also	
Last Change	V4.0

2.3.8. jc_0245: Length restrictions for signal and bus names

ID: Title	**jc_0245:** Length restrictions for signals and bus names
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	Signal and bus name lengths should be 3 to 63 characters.

	 Signal and bus name
	Bus signals can be layered. It is better to restrict the overall number of characters (full path).  Overall number of characters. 
See Also	
Last Change	V4.0

2.3.9. jc_0246: Length restrictions for parameter names

ID: Title	**jc_0246: Length restrictions for parameter names**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB	ALL

Version	
Prerequisites	
Description	Parameter name lengths should be 3 to 63 characters.
See Also	
Last Change	V4.0

2.3.10. jc_0247: Length restrictions for block names

ID: Title	**jc_0247: Length restrictions for block names**
Priority	recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	Block name lengths should be 3-63 characters.
See Also	
Last Change	V4.0

2.4. Notes on other used characters

2.4.1. na_0035: Adoption of naming conventions

ID: Title	**na_0035: Adoption of naming conventions**
Priority	Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Adoption of a naming convention is recommended. A naming convention provides guidance for naming blocks, signals, parameters and data types. Naming conventions frequently cover issues such as: Readability: • Use of underscores • Use of capitalization Encoding information: • Use of meaningful names • Standard abbreviations and acronyms • Data type • Engineering units (system of units) • Data ownership • Memory type
Notes	This is an example of a rule relating to readable capitalization. ➢ All-capital parameters should define storage class.. ➢ All-capital signal (Simulink, mpt objects) names should not be used. Names are defined for signal lines (label names), but signal line names that only have an annotative significance without defining Simulink or mpt objects, are given in all capitals to distinguish them from global signals.

	Acronym is a kind of abbreviations mainly used in European languages. It is created from initial characters of compound word which consists of several words.
See Also	
Last Change	V4.0

2.4.2. jc_0251: Naming restrictions for signals and parameters.

ID: Title	jc_0251: Naming restrictions for signals and parameters.
Priority	Mandatory
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	There are 2 constraints on signal names and parameter names inside a model. 1. Please do not use any reserved words, function names or operator names used by MATLAB such as pi, true and false . 2. Please do not use any words reserved in MATLAB auto coding . Even when a simulation has been run, problems may on occasions arise when automatically generating code or when integrating it. Parameters that may not be used can be checked using iskeyword, but this function only checks the names that have been registered as MATLAB keywords. Function names and operator names cannot be checked with this function. A number of examples is listed below, but care must be taken as there are numerous examples apart from these. ● MATLAB keywords 'break', 'case', 'catch', 'classdef', 'continue', 'else', 'elseif', 'end', 'for','function', 'global', 'if', 'otherwise', 'parfor', 'persistent', 'return', 'spmd', 'switch', 'try', 'while' ● Function names, constant names, operator names 'eps','Enf','intmax','intmin','NaN','pi','realmax','realmin','true','false','inf' The following are reserved by MATLAB for auto coding. • const、TRUE、FALSE、infinity, nil, double, single, or, enum
Notes	Reserved words are defined in the Simulink Coder documentation. http: //www.mathworks.co.jp/jp/help/symbolic/reserved-variable-and-function-names.html
See Also	
Last Change	V4.0

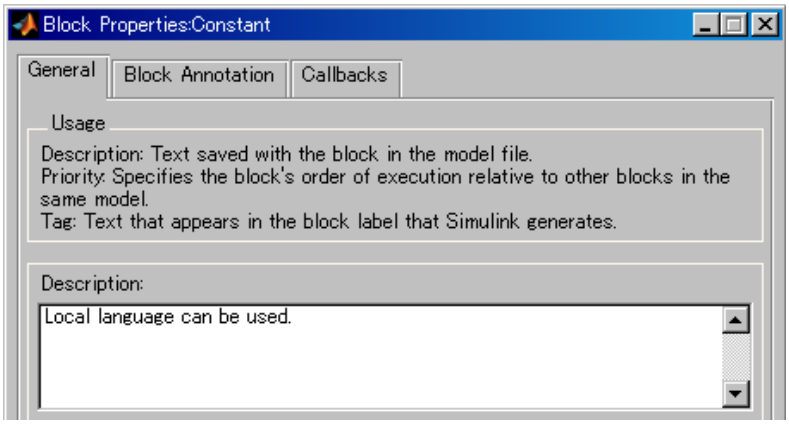
2.4.3. na_0014: Use of local language in Simulink and Stateflow

ID: Title	**na_0014: Use of local language in Simulink and Stateflow**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	The local language should be used only in descriptive fields. Descriptive fields are text entry points that do not affect code generation or simulation.

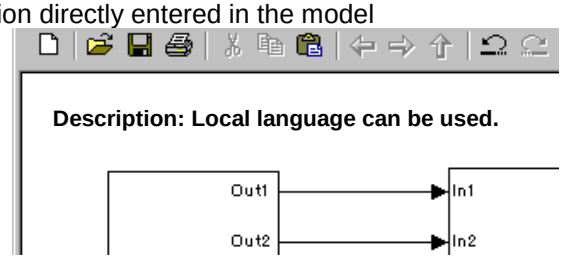
Examples of descriptive fields include the [Description] field in the Block Properties dialog box.

Simulink Example:

- The description field in the Block Properties dialog box.

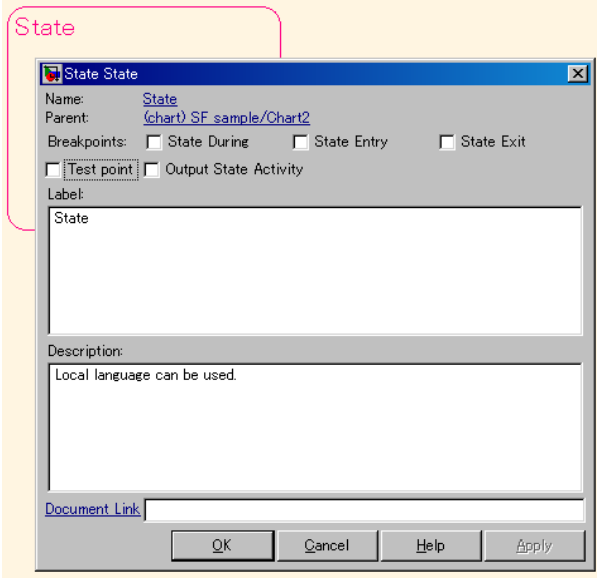


- Text annotation directly entered in the model

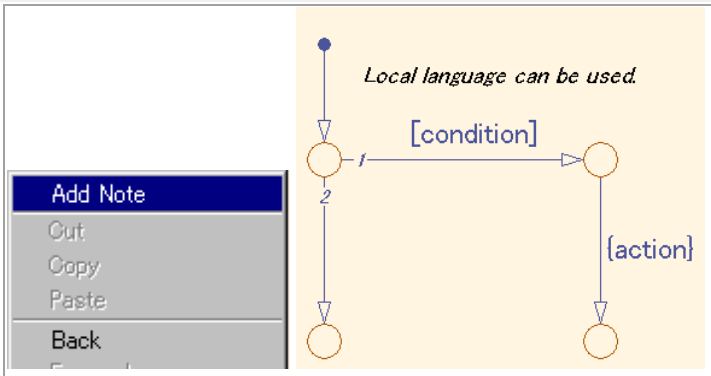


Stateflow Example:

- The Description field in chart or state Properties

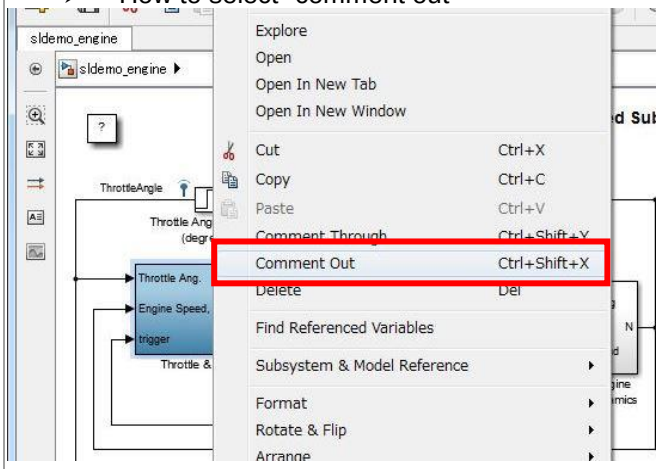


- Annotation description added using Add Note



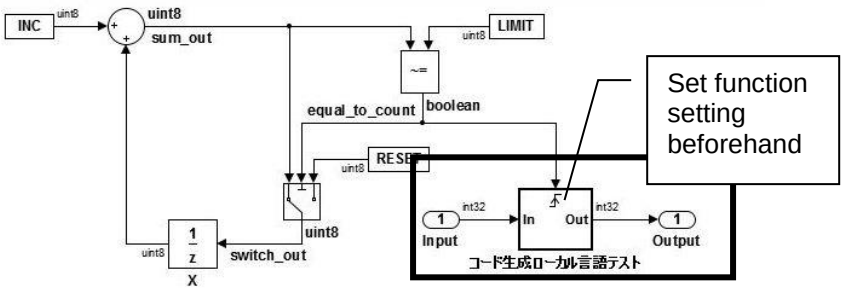
There are also many other places in masked subsystem Disp that correspond to Description fields, such as user tags, inside block annotations and commented out subsystems.

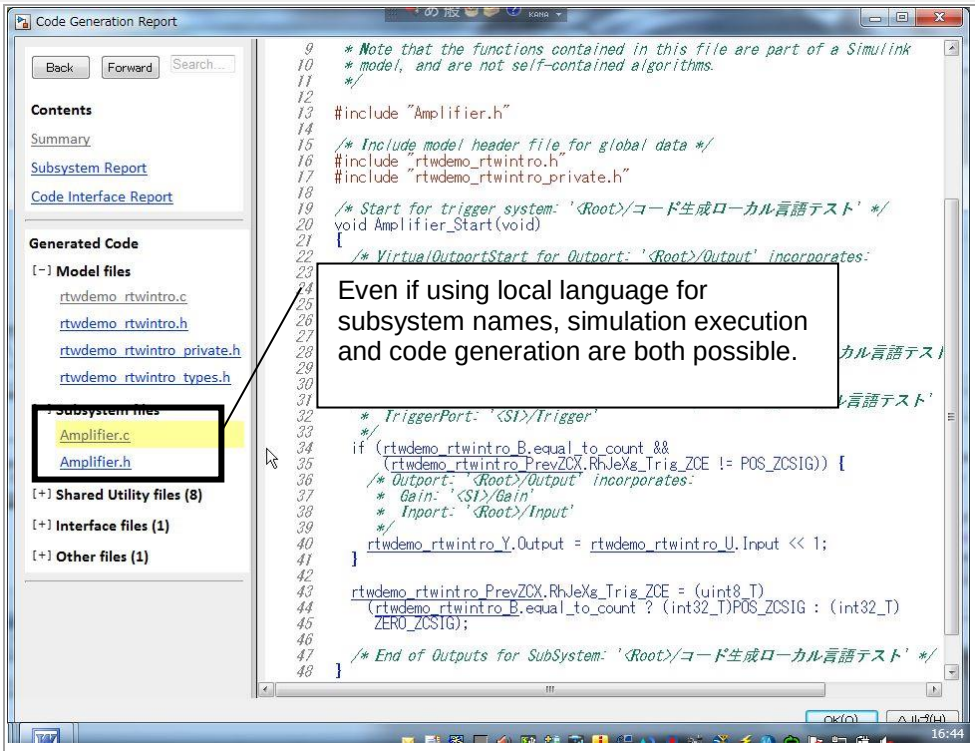
- How to select “comment out”



Notes

Description fields may vary between versions.
In recent Simulink versions the use of local language is allowed for subsystem names and block names.
Both simulation and code generation are possible if only characters that allow for code generation are designated in the function setting.



	
See Also	
Last Change	V2.0

3. Model Architecture

3.1.1. na_0006: Guidelines for mixed use of Simulink and Stateflow

ID: Title	**na_0006: Guidelines for mixed use of Simulink and Stateflow**
Priority	Strongly Recommended
Scope	NAMAAB
MATLAB Version	ALL
Prerequisites	
Description	The choice of whether to use Simulink or Stateflow to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled.
Notes	The details are this.「10.1 The roles of Simulink and Stateflow」
Last Change	V4.0

3.1.2. na_0007: Guidelines for use of Flowcharts, Truth Tables and State Machines

ID: Title	**na_0007: Guidelines for use of Flowcharts, Truth Tables and State Machines**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	na_0006: Guidelines for Mixed use of Simulink and Stateflow
Description	Within Stateflow, the choice of whether to use a Flowchart or a state chart to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled. ● If the primary nature of the function segment is to calculate modes of operation or discrete-valued states, use state charts. Some examples are: ・ Diagnostic model with pass, fail, abort, and conflict states ・ Model that calculates different modes of operation for a control algorithm ● If the primary nature of the function segment involves if-then-else statements, use Flowcharts or Truth Tables. Specifics: If the primary nature of the function segment is to calculate modes or states, but if-then-else statements are required, add a Flowchart to a state within the state chart. (See 5.7Flow Chart foundation)
Last Change	V2.0

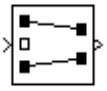
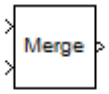
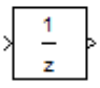
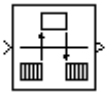
3.1.3. db_0143: Similar block types on the model levels

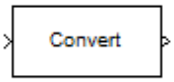
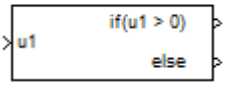
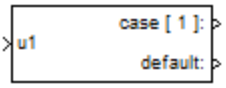
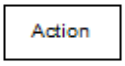
ID: Title	**db_0143: Similar block types on the model levels**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	A structure layer(Function or Schedule layer) and data flow layer shall not be mixed at the same layer. Usable block types in the model layers should be restricted and the list of

usable block types according to model layers should be prepared and agreed upon. To understand layer concept, refer 10.2 Hierarchical structure of a controller model.

However, the following blocks are not limited to a layer, and can be used on all levels.

Blocks which can be placed on every model level (blocks that can be used on all levels)

Block types	Examples of block icons
Inport	
Outport	
Mux	
Demux	
Bus Selector	
Bus Creator	
Select or	
Ground	
Terminator	
From	
Goto	
Merge	
Unit Delay(1)	
Rate Transition	

	Data Type Conversion 
	Data Store Memory 
	If 
	Case 
	Function-Call Generator 
	Function-Call Split 
	Trigger[2] 
	Enable[3] 
	Action port[4] 
Notes	4) Not only the Unit Delay block but all similar blocks like the Delay block are treated in the same manner. 2) In R2011a and earlier, Enable block is not allowed at the root level of the model. 3) In R2008b and earlier, Trigger block is not allowed at the root level of the model. Note: If the Trigger or Enable blocks are placed at the root level of the model, then the model will not simulate in a standalone mode. The model must be referenced using the Model block. 4) Action port is allowed at the root level of the model. Regarding kinds of laysers, please see appendix. Establish standards for each project on whether to include libraries or virtual subsystems within the scope of "Subsystems only".
Last Change	V4.0

3.1.4. db_0144: Use of Subsystems

ID: Title	**db_0144: Use of Subsystems**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Blocks in a Simulink diagram should 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 block diagram should represent a unit of functionality required to accomplish the purpose of the model or submodel. Blocks can also be grouped together based on behavioral variants or timing.

	If creation of subsystems is required for readability issues, then a virtual subsystem should be used.
Last Change	V2.2

4. Simulink

4.1. Diagram appearance

4.1.1. na_0004: Simulink model appearance

ID: Title	**na_0004: Simulink model appearance**
Priority	Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	The model appearance settings should conform to the following guidelines when the model is released.

The user is free to change the settings during the	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 Indicator	unchecked
Library Link Display	none
Linearization Indicators	checked
Model/Block I/O Mismatch	unchecked
Model Block version	unchecked
Sample Time Colors	unchecked
Sorted Order	unchecked

Signal Display Options	Setting
Port Data Types	unchecked
Signal Dimensions	unchecked
Storage Class	unchecked
Test point Indicators	checked
Viewer Indicators	checked

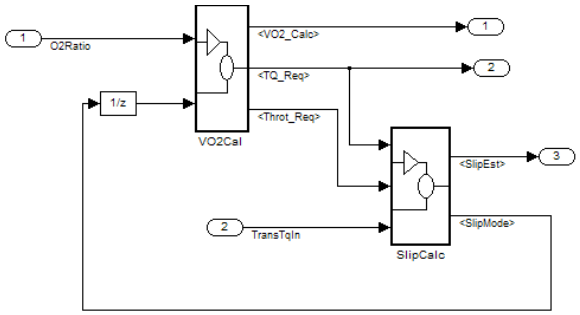
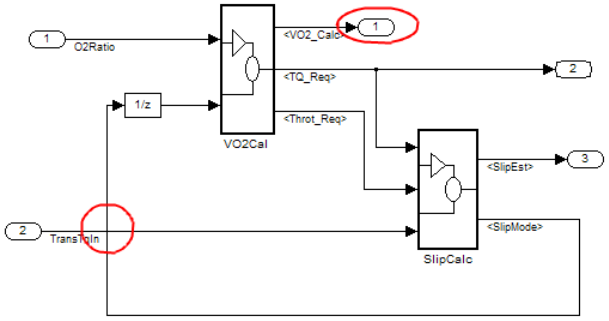
	Wide Non-scalar Lines	checked
Notes	These are an example. Please set standards for each project.	
See Also	MISRA AC SLSF 023A	
Last Change	V2.0	

4.1.2. db_0043: Simulink font and font size

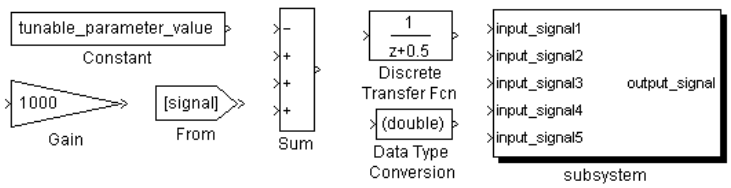
ID: Title	**db_0043: Simulink font and font size**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	All text elements (block names, block annotations and signal labels) except text annotations within a model must have the same font style and font size. Fonts and font size should be selected for legibility.
Notes	The selected font sould be directly portable (e.g. Simulink/Stateflow default font) or convertible between platforms (e.g. Arial/Helvetica 12pt).
Last Change	V2.0

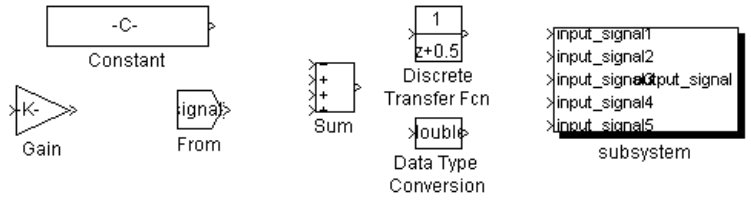
4.1.3. db_0042: Port block in Simulink models

ID: Title	**db_0042: Port block in Simulink models**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	In a Simulink model, the ports comply with the following rules: • Inports should be placed on the left side of the diagram, but they can be moved in to prevent signal crossings. • Outports should be placed on the right side, but they can be moved in to prevent signal crossings. • Duplicate Inports can be used at the subsystem level if required but should be avoided if possible. ○ Duplicate Inports cannot be used at the root level.

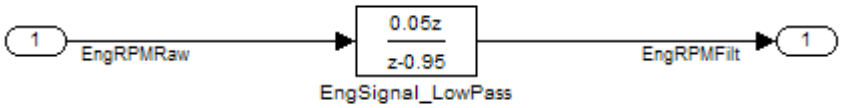
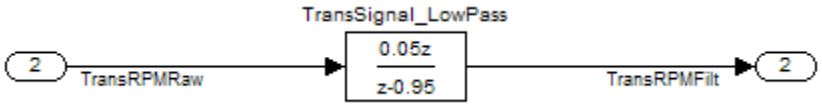
	Correct:  **Incorrect:**  Notes on the incorrect model • Inport 2 should be moved in so it does not cross the feed back loop lines. • Outport 1 should be moved to the right hand side of the diagram .
Last Change	V2.0

4.1.4. jm_0002: Block resizing

ID: Title	**jm_0002: Block resizing**
Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	All blocks in a model must be sized such that their icon is completely visible and recognizable. In particular, any text displayed (e.g. tunable parameters, filenames, equations) in the icon must be readable. However, when it is difficult to resize subsystems with many inputs and outputs, the content of the icon should be made visible in an alternative way. **Correct:**  **Incorrect:**

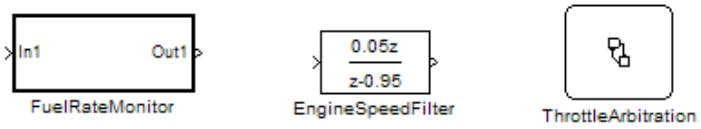
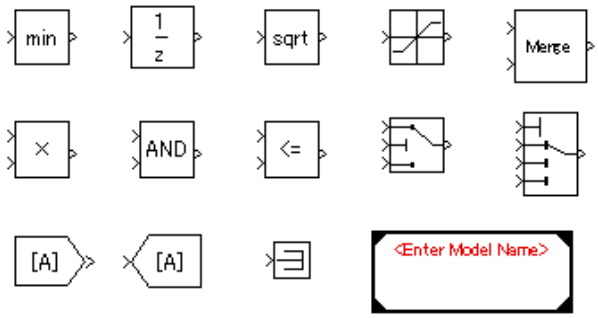
	
Notes	This guideline requires resizing of blocks with variable icons relying on option settings or blocks with variable number of inputs and outputs. However, in some cases, it may not be practical or desirable to resize the block icon of a subsystem block so that all of the input and output names within it are readable. In such cases, you may hide the names in the icon by using a mask or by hiding the names in the subsystem associated with the icon. If you do this, the signal lines coming into and out of the subsystem block should be clearly labeled in close proximity to the block.
Last Change	V2.0

4.1.5. db_0142: Position of block names

ID: Title	**db_0142: Position of block names**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	If shown the name of each block should be placed below the block. Correct:  Incorrect: 
Last Change	V2.0

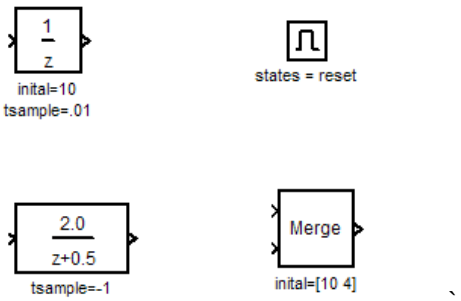
4.1.6. jc_0061: Display of block names

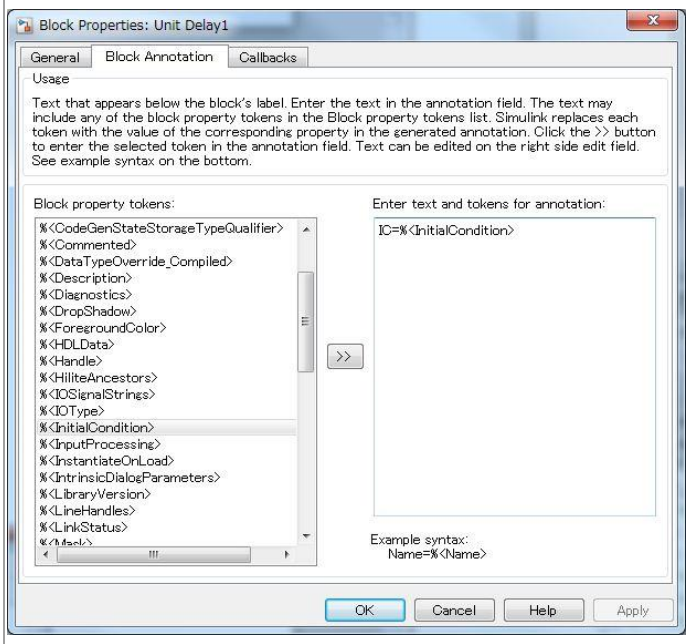
ID: Title	**jc_0061: Display of block names**
Priority	Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	● Display block names for blocks which have a functional requirement to have the name displayed, and for blocks with names that have significance. Examples of blocks with function names instead of block names.

	 ● No block names are displayed for blocks to which all of the following applies. * Its function is understood from its appearance. (actual blocks are defined for each development project) * No changes to default block names apart from the number at the end. Examples of blocks where names are not displayed 
Notes	Through sldiagnostics (model name), the block classification and numbers used in the model that is used are known. Based on the results of this command we can infer which blocks are well-know and which ones aren't. In line with our own training curriculum, it would be better to display the block names for blocks whose function is not that well-known.
See Also	MISRA AC SLSF 026A
Last Change	V4.0

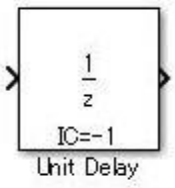
4.1.7. db_0140: Display of block parameters

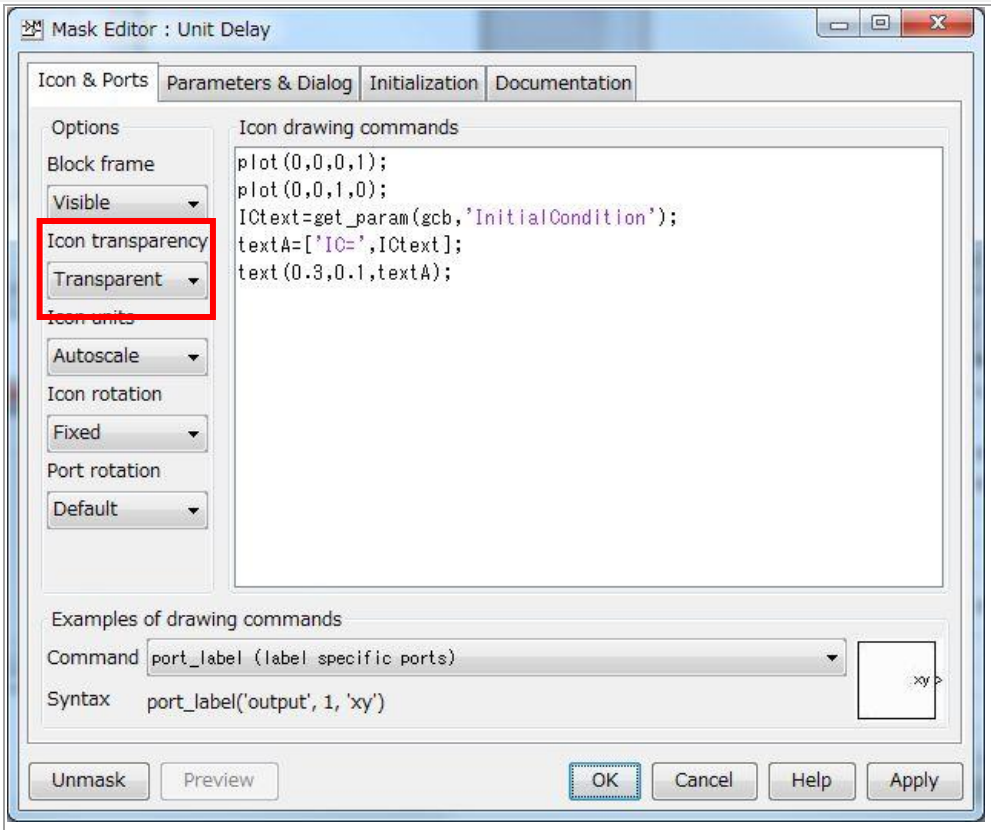
ID: Title	**db_0140: Display of block parameters**
Priority	Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Important block parameters must be displayed. In R2011b and later, masking basic blocks is a supported method for displaying the information. This method is allowed if the base icon is distinguishable. **Correct:**



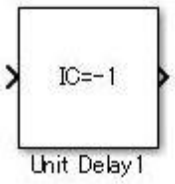


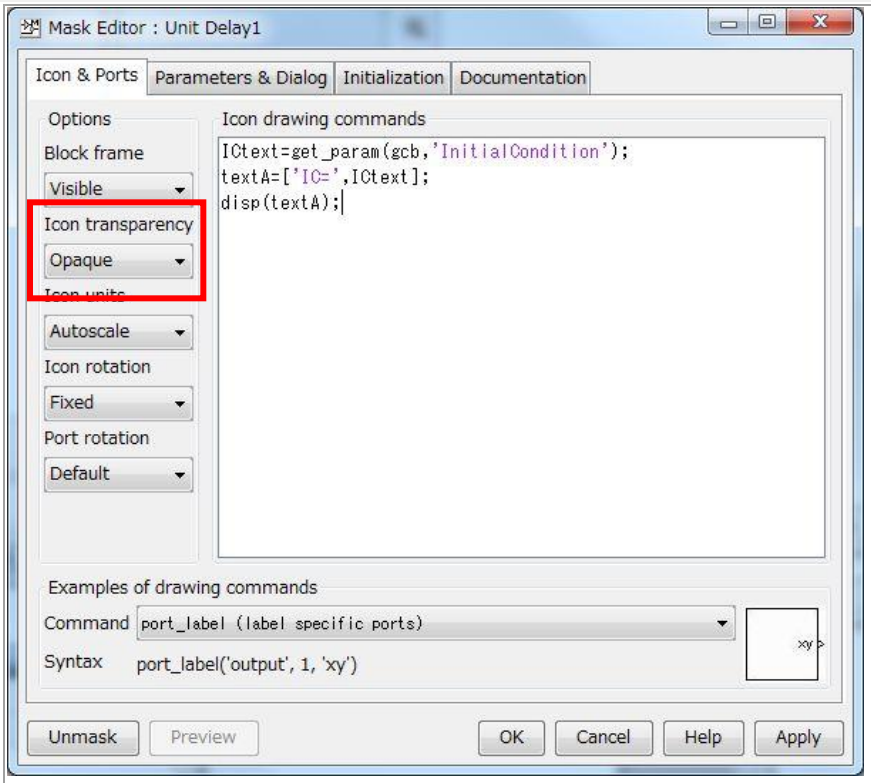
Correct: Masked block
Use the display function by masking the basic block





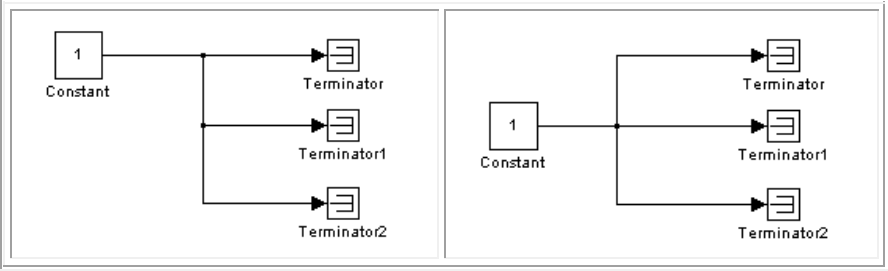
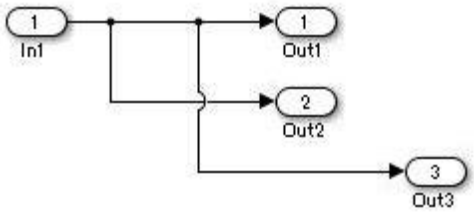
Incorrect: Because of mask,base icon of masked block cannot be seen.



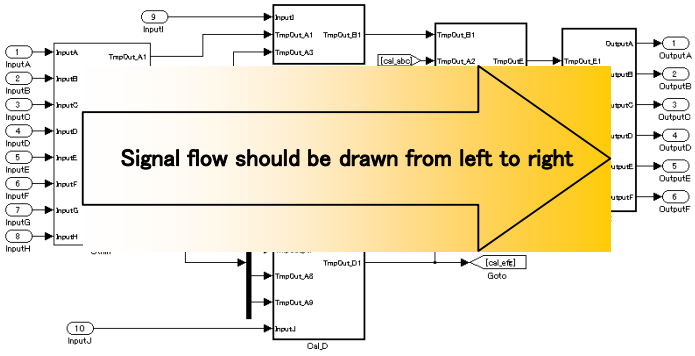
	
Notes	Displaying properties is a way to realize to show block parameters. Necessary property information can be added on [Block Annotation] tab. The block parameters that must be displayed will change depending on the process. Please change the required information for each process. The parameters considered to be important vary depending on the used Simulink version.
See Also	MISRA AC SLSF 026E
Last Change	V4.0

4.1.8. db_0032: Simulink signal appearance

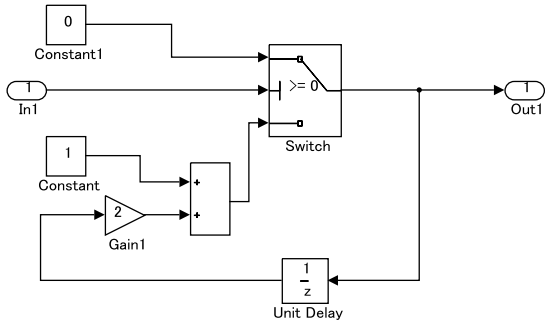
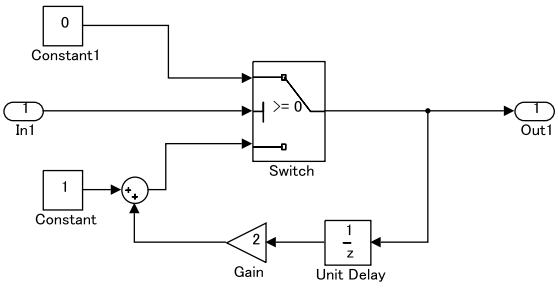
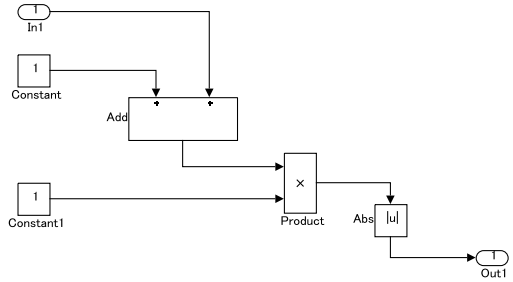
ID: Title	**db_0032: Simulink signal appearance**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Please adhere to the following rules for signal lines. • Should not cross each other, if possible. • Should bend at right angles (only use vertical and horizontal lines, do not draw them diagonally) • Are not drawn one upon the other. • Should not cross any blocks. • Should not split into more than two sub lines at a single branching point (cross-shaped connections are not permitted). **Correct:** **Incorrect:**

	
Notes	Vertical line that is crossed is now get off holizontal line.  As a result, the cross, the difference of the branch is now clear. • Should not cross each other, if possible.. • Should not split into more than two sub lines at a single branching point. Above two rules were made because the branch and cross is hard to recognize. You can be less restrictive in R2014a and later. Please determine whether adopt it or not based on the version you use.
Last Change	V2.0

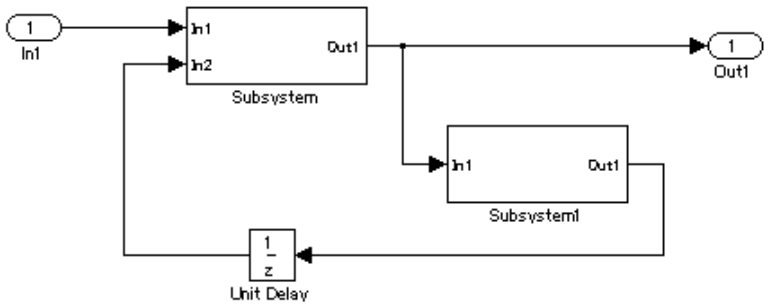
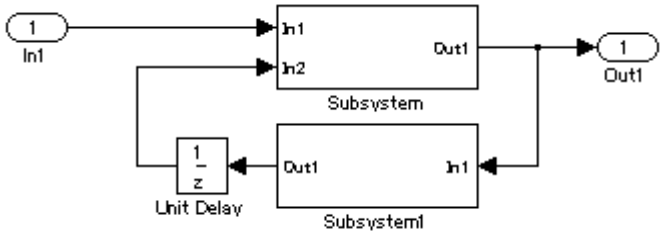
4.1.9. db_0141: Signal flow in Simulink models

ID: Title	**db_0141: Signal flow in Simulink models**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	• Signal flow in a model is from left to right. (Exception: Feedback loops) • Sequential blocks or subsystems are arranged from left to right. (Exception: Feedback loops) • Parallel blocks or subsystems are arranged from top to bottom. 
Last Change	V2.0

4.1.10. jc_0110: Direction of block

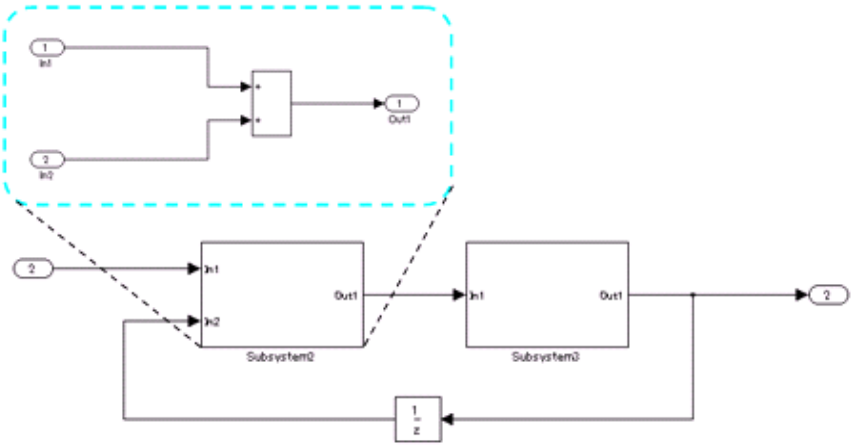
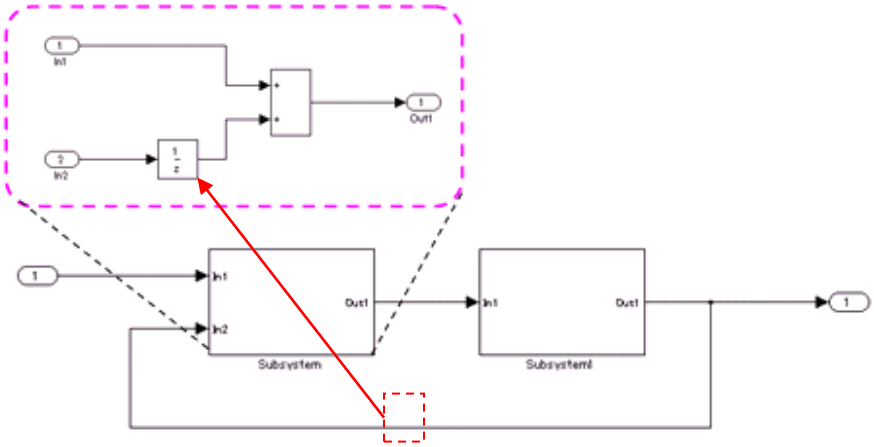
ID: Title	**jc_0110: Direction of block**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	**db_0141: Signal flow in Simulink models**
Description	Any blocks other than blocks with Delay blocks (e.g. Unit Delay) should not be rotated or reversed. **Correct:**  Only the Unit Delay block is reversed. **Incorrect:**  The Gain block is also reversed. **Incorrect:**  The signal flow is drawn from left to right, but the blocks are used vertically.
See Also	
Last Change	V4.0

4.1.11. jc_0111: Direction of Subsystem

ID: Title	**jc_0111: Direction of Subsystem**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	jc_0110: Direction of block
Description	The direction of the subsystem must not be rotated or reversed. **Correct:**  **Incorrect:** 
Last Change	V2.0

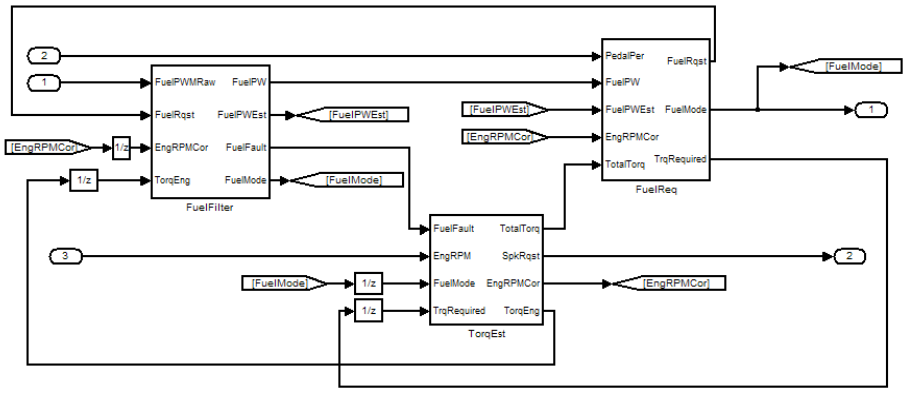
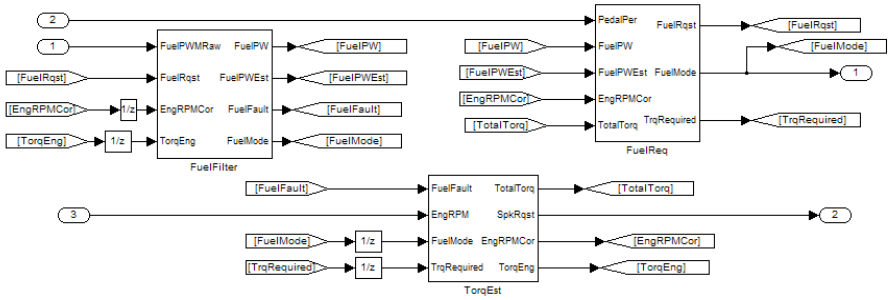
4.1.12. jc_0653: Guidelines for avoiding algebraic loops between subsystems

ID: Title	**jc_0653: Guidelines for avoiding algebraic loops between subsystems**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	When using Delay blocks (e.g. Unit Delay blocks) with the purpose of preventing algebraic loops in feedback loops across subsystems, they must be placed on the outside of the subsystem. Rationale: ● If a Delay block is placed inside a subsystem, it is difficult to know where it has been placed, and the Delay may be duplicated. Placing it on the outside makes it explicit. ● Delay blocks inside a subsystem decrease its reusability. ● Inspection times will be longer due to the dependence on past values.

	Correct: The Delay block is placed outside the subsystem  Incorrect: The Delay block is placed inside the subsystem 
See Also	
Last Change	V4.0

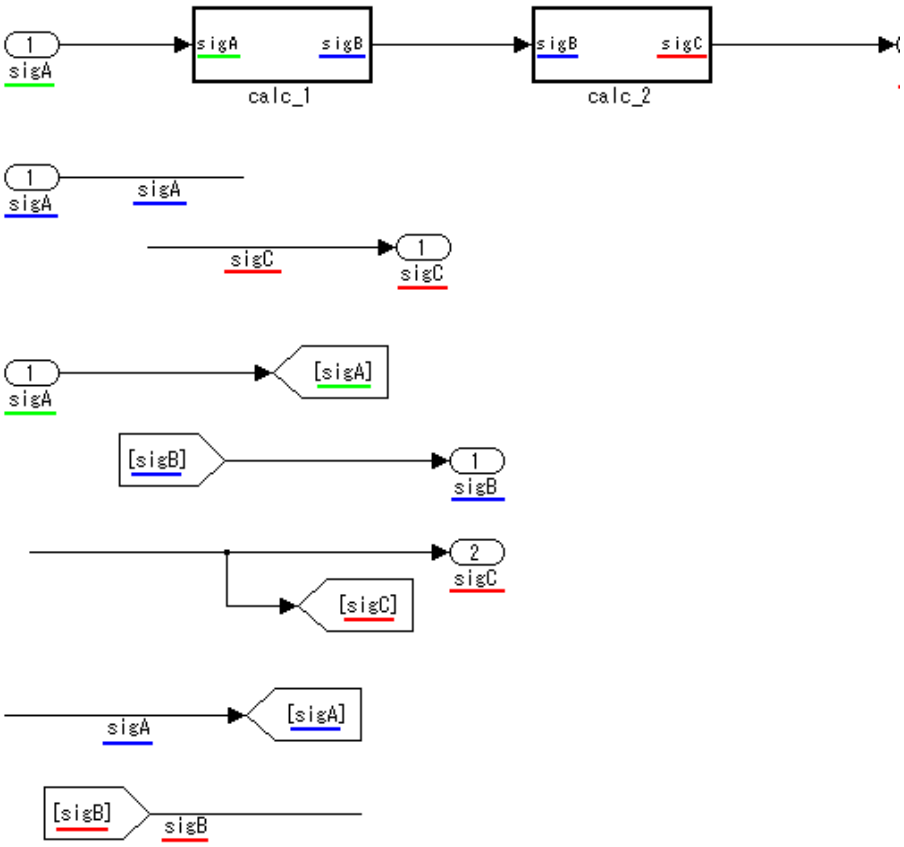
4.1.13. jc_0171: Maintaining signal flow when using Goto and From blocks

ID: Title	**jc_0171: Maintaining signal flow when using Goto and From blocks**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Visual depiction of signal flow must be maintained between subsystems. ● Use of Goto and From blocks is allowed in the following cases: ➢ At least one signal line is used between connected systems. ➢ Subsystems connected in a feed-forward and feedback loop have at least one signal line for each direction.

	Correct:  **Incorrect:** 
Notes	This rule is to visually clarify the connection between subsystems. Using Goto and From blocks to create buses or connect inputs to merge blocks are exceptions to this rule.
Last Change	V4.0
	Rule for bus added by NAMAAB is unclear. Since this rule mentions about connection between subsystems, bus has no relation.

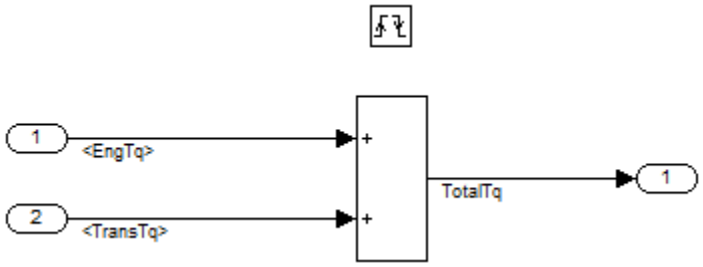
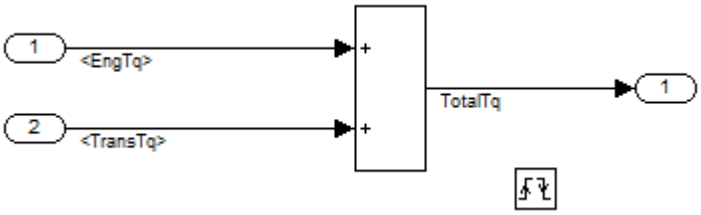
4.1.14. jc_0602: Consistency in model element names

ID: Title	**jc_0602: Consistency in model element names**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	db_0042**: Port block in Simulink models** na_0005**: Display of Inport and Outport block names** db_0123**: Stateflow port names**
Description	Names (characters) for the following model elements directly connected to the same signal should be consistent. ・ Inport block: block name (if the block name is displayed) ・ Outport block: block name (if the block name is displayed) ・ Goto block: tag name (not the block name) ・ From block: tag name (not the block name) ・ Signal line: signal name (including legacy signal names)

	・ Subsystem: masked port label names (if the port name is visible from above) Label name when the port is displaying the label name ・ Inport, Outport prioritize rule na_0005. However, for signals connected to the following subsystems, the connected boundaries are regarded as an exception. ➢ Subsystems linked to a library ➢ Reusable subsystems 
Notes	If a combination of Inport blocks, Outport blocks and other blocks has the same block name, use a suffix or prefix for the Inport and Outport blocks. Often used suffixes and prefixes are "in" for Inport blocks and "out" for Outport blocks. Any prefix or suffix can be used for ports, but consistent prefixes must be selected.
See Also	MISRA AC SLSF 036-C
Last Change	V4.0

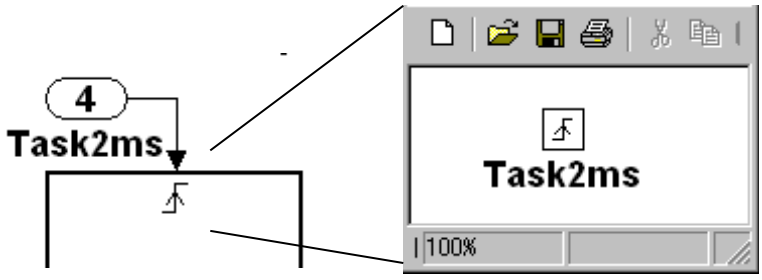
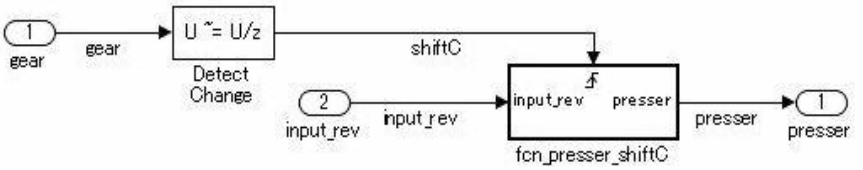
4.1.15. db_0146: Triggered, enabled, conditional Subsystems

ID: Title	**db_0146: Triggered, enabled, conditional Subsystems**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Conditional input blocks should be located at the tope of the subsystem. 1.Conditional input blocks ● Enable ● For Iterator

	• Action Port • Switch Case Action • Trigger • While Iterator Following blocks also should be uniformely located. 2.Blocks treated as nealy same as conditional input blocks. • For Each • For Iterator **Correct:**  **Incorrect:** 
Notes	• This guideline intends to improve readability by unifying outer shape of subsystem and internal location. Regarding For Each block, For Iterator block and While Iterator block, locations should be unified. However, regarding While Iterator block, it should be careful since it is difficult to fix the location. • It is necessary to clarify the positions, when the model information of jc_0603 is described.
Last Change	V4.0

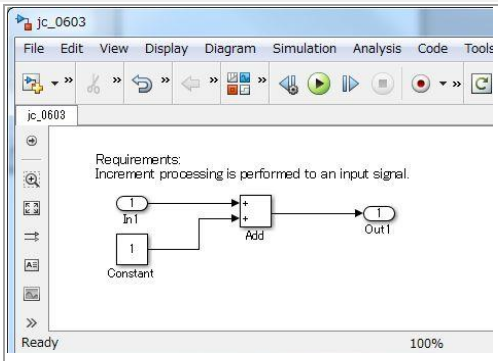
4.1.16. jc_0281: Naming of Trigger Port block and Enable Port block

ID: Title	**jc_0281: Naming of Trigger Port block and Enable Port block**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	For conditional subsystems including conditional input blocks such as Trigger Port blocks, the effect of the signal that triggers the subsystem is recorded both in the source and the destination. At the source, names that indicate the effect are given to either of the following: • Block name • Subsystem's block name（Part） • Signal name At the destination, these names are added to either of the following: • Block name of the conditional input block (Trigger, Enable Port)

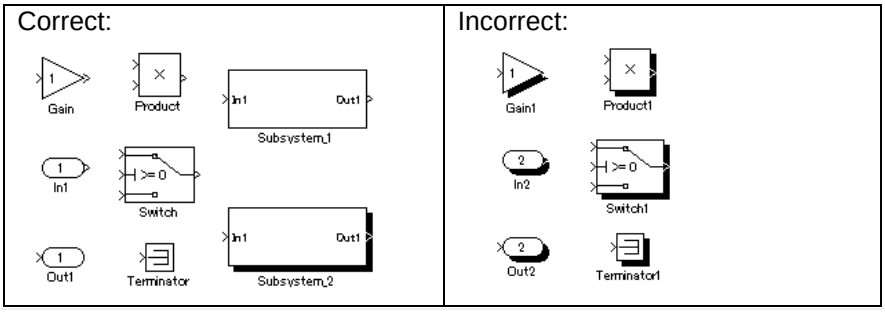
	• Part of the subsystem name of the conditional subsystem Correct: example of a matching block name at the source and block name of the conditional input block at the destination.  Correct: example of a matching signal name and suffix for the connected subsystem.  Exception: • In the case of library blocks that encapsulate generic functionality or reusable subsystems, generic names for the signal should be used.
Notes	The purpose of this rule is to improve readability while also considering the prevention of connection errors where the automatic checker is checking for connection errors. A simple name inheritance rule that can be generally interpreted should be established for the purpose of automatic checks by the checker.
See Also	MISRA AC SLSF 026C
Last Change	V4.0

4.1.17. jc_0603: Model description

ID: Title	**jc_0603: Model description**
Priority	Recommended
Scope	JMAAB
MATLAB Version	ALL
Description	Define functional units where a model description will be added, and supply a model description for each functional unit using annotations or ModelInfo blocks. Use a common format for the model description in the entire model. For instance, use explicitly understood fixed headings (e.g. "Requirements", "Summary"). Example:

	
Notes	Use the example above to determine the notation, placement and headings for the description.
See Also	MISRA AC SLSF 022
Last Change	V4.0

4.1.18. jc_0604: Block shading

ID: Title	**jc_0604: Block shading**
Priority	Recommended
Scope	JMAAB
MATLAB Version	ALL
Description	Block shading should not be used to show that signal lines are not connected, except in the following cases: ● Subsystems without an Output Port ● Subsystems with displayed signal name 
Notes	If the signal name is noted in the subsystem, it is explicit that it has an Output Port. As it will be immediately clear that it is not connected, it will not fall within the restrictions of this rule.
See Also	MISRA AC SLSF 024A
Last Change	**V4.0**

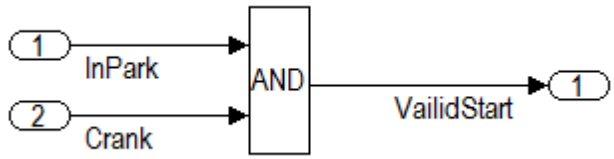
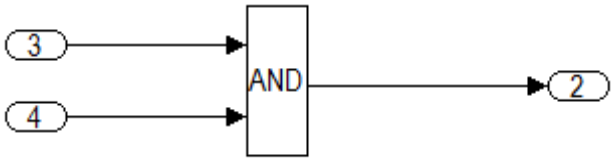
4.2. Signals

4.2.1. na_0010: Grouping data flows into signals

ID: Title	**na_0010: Grouping data flows into signals**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	**Restrictions on use of busses and vectors** ・ Mux and Demux blocks must only be used in generating and decomposing vectors. ・ Scalars and vectors must be used for Mux input. ・ BusCreator and BusSelector must only be used in generating and decomposing busses. ・ To avoid the problem of mixing Mux and busses, connect busses to bus-supported blocks.
See Also	MISRA AC SLSF 015A,B,C,016A,B,C,D,E
Last Change	V4.0

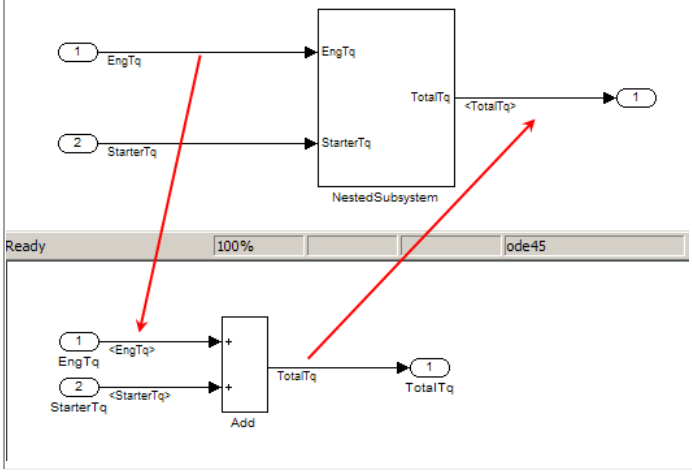
4.2.2. na_0008: Display of labels on signals

ID: Title	**na_0008: Display of labels on signals**
Priority	recommended
Scope	NAMAAB
MATLAB Version	All
Prerequisites	
Description	A label must be displayed on a signal originating from the following blocks: • Inport block • From block (block icon exception applies – see Note below) • Subsystem block or Stateflow chart block (block icon exception applies) • Bus Selector block (the tool forces this to happen) • Demux block • Selector block • Data Store Read block (block icon exception applies) • Constant block (block icon exception applies) A label must be displayed on any signal connected to the following destination blocks (directly or by way of a basic block that performs a non transformative operation): • Outport block • Goto block • Data Store Write block • Bus Creator block • Mux block • Subsystem block • Chart block

	Note: Block icon exception (applicable only where called out above): If the signal label is visible in the originating block icon display, the connected signal does not need not to have the label displayed, *unless* the signal label is needed elsewhere due to a destination-based rule. **Correct**  **Incorrect** 
Rationale	☑ Readability ☑ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.2

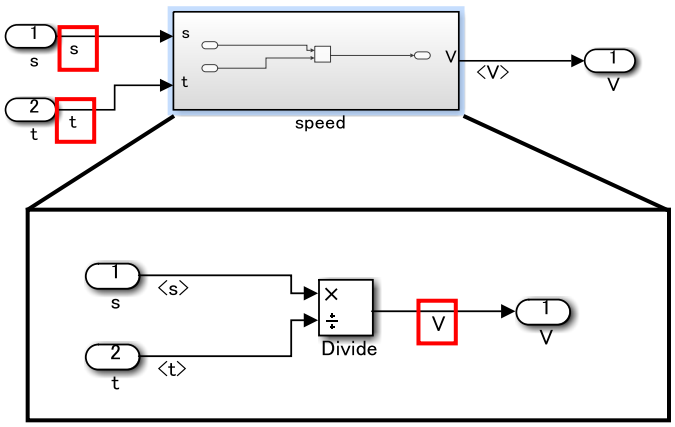
4.2.3. na_0009: Entry versus propagation of signal labels

ID: Title	**na_0009: Entry versus propagation of signal labels**
Priority	Strongly Recommended
Scope	NAMAAB
MATLAB Version	All
Prerequisites	na_0008: Display of labels on signals
Description	If a label is present on a signal, the following rules define whether that label shall be created there (entered directly on the signal) or propagated from its true source (inherited from elsewhere in the model by using the '<' character). 1. Any displayed signal label must be *entered* for signals that: a. Originate from an Inport at the Root (top) Level of a model b. Originate from a basic block that performs a transformative operation (For the purpose of interpreting this rule only, the Bus Creator block, Mux block, and Selector block shall be considered to be included among the blocks that perform transformative operations.) 2. Any displayed signal label must be *propagated* for signals that: a. Originate from an Inport block in a nested subsystem **Exception:** If the nested subsystem is a library subsystem, a label may be *entered* on the signal coming from the Inport to accommodate reuse of the library block. b. Originate from a basic block that performs a non-transformative operation c. Originate from a Subsystem or Stateflow chart block **Exception:** If the connection originates from the output of a library subsystem block instance, a new label may be *entered* on the signal to accommodate reuse of the library block.

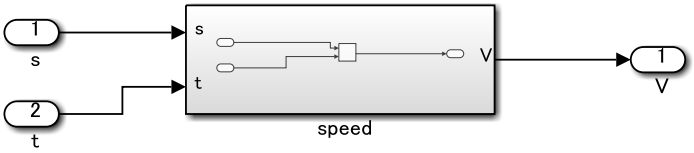
	
Rationale	☑ Readability ☑ Workflow ☑ Simulation ☑ Verification and Validation ☑ Code Generation
Last Change	V2.0

4.2.4. jc_0008 : Definition of a Signal labels.

ID: Title	**jc_0008 : Definition of a Signal labels.**
Priority	recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	Defines signal name to input port which is located at system top layer or signal lines which is output from important block. ● Labels has to be displayed when signal name is defined. ● Signal name needs to be input only once (to the place where signal is occurred). An important block is a block which outputs the result which is not decided by the kind of block but is meaningful. **Correct:** Signal name is settled at necessary location, and is displayed. **Correct:** Signal name is settled at necessary location, and is displayed.



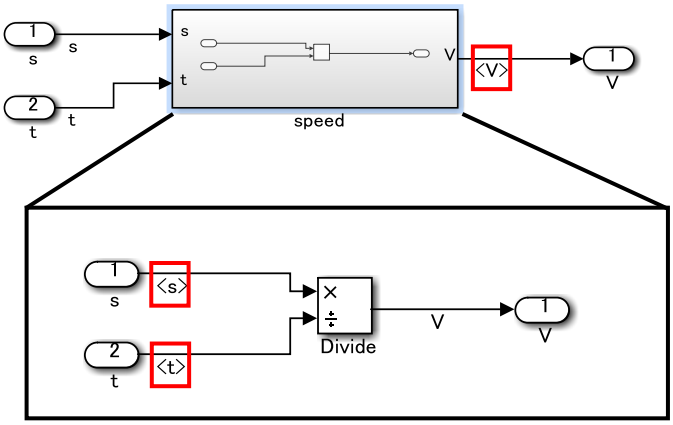
Incorrect:
Signal name is not settled.



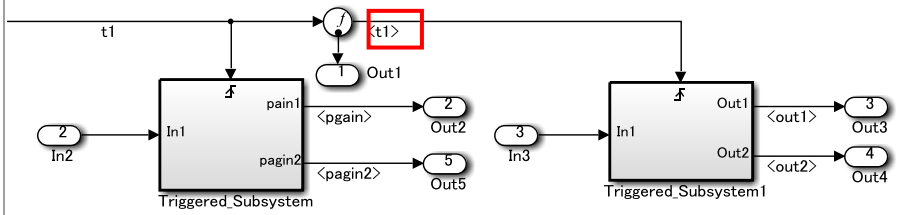
See Also	MISRA AC SLSF 027C,027D,027F,027G,027I,027J
Last Change	V4.0

4.2.5. jc_0009 : Propagation of signal label

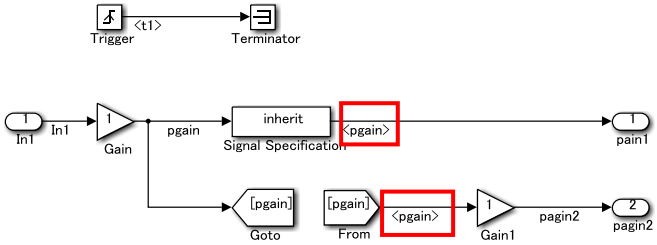
ID: Title	**jc_0009 : Propagation of signal label**
Priority	**Recommended**
Scope	**JMAAB**
MATLAB Version	**ALL**
Prerequisites	**jc_0008 : Definition of Signal labels**
Description	**If the signal name is propagated (with signal name), turn propagation signal ON, and display signal name.** **When signal name is defined at different layer, signal name is displayed with propagation signal ON.** **However, in the following cases, without cross the hierarchy, propagation of the signal display name is required** ➢ **Target block: Signal output from the basic block to perform a non-conversion operation** ・ **from,goto** ・ **Bus Creator ,Bus Selector** ・ **Signal Specification** ・ **Function Call Split** **Propagation signal display example1: Propagation display step over the hierarchy**



Propagation signal display example2: Propagation display at same hierarchy



Propagation signal display example2: Propagation display at same hierarchy

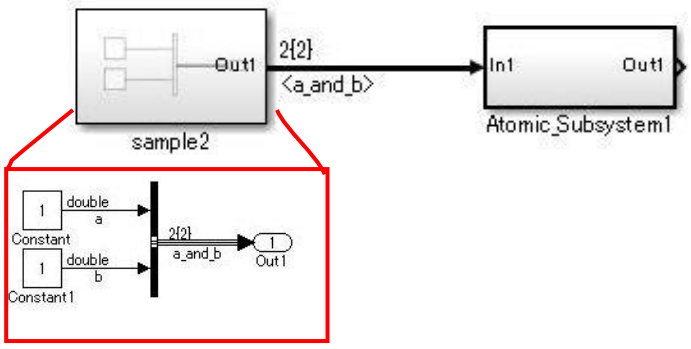


Exception to ON the display of the signal propagation

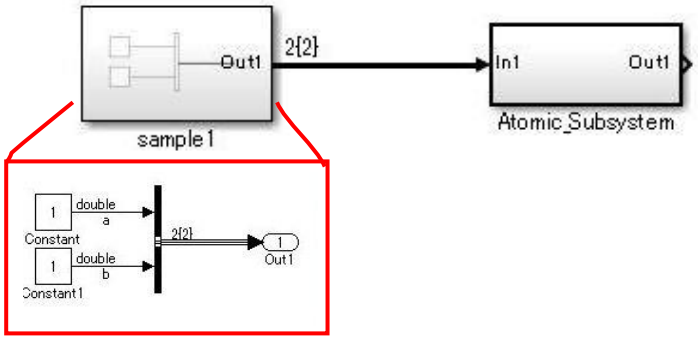
1. **Subsystem inside that library and reusable function is set.**
2. **No signal name is set at Bus Creator outport.**

In case there is a signal name at Bus Creator outport, propagation signal is ON for this signal. However, in case there is no signal name on Bus Creator outport, propagated signal is transmitted in a state in which all of the bus signal names was degraded in the past MATLAB. It is also transmitted in empty in the latest MATLAB. This case, propagation signal is not ON.

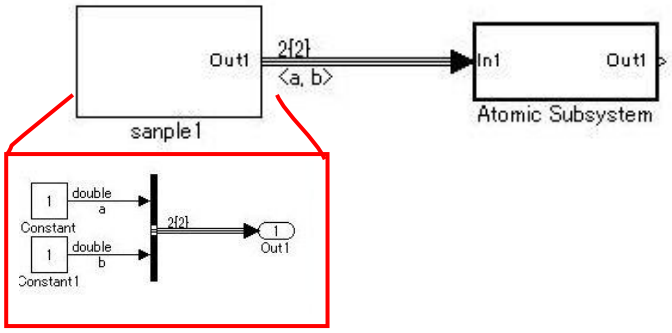
Correct
If there is signal name on Bus Creator outport, propagation should be ON.



Correct
If there is no signal name on Bus Creator outport, propagation should be OFF.



Incorrect
In case no signal name is put on Bus. (R2010b)



Last change	**V4.0**
See also	**hisl_0013: Guideline for using the Data Store block** **MISRA AC SLSF 005C**
Last change	**V4.0**

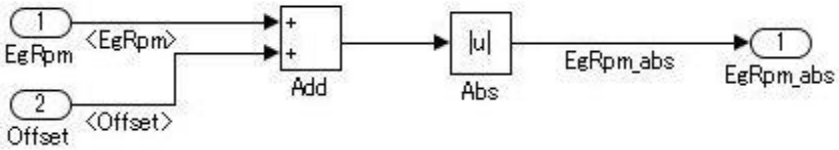
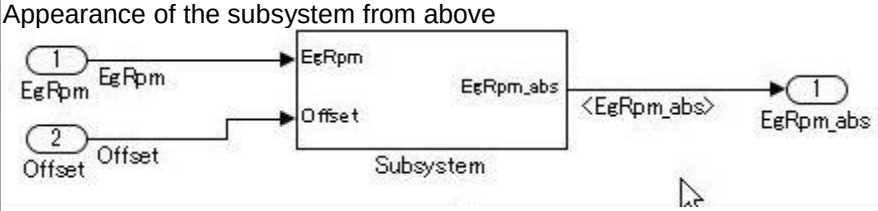
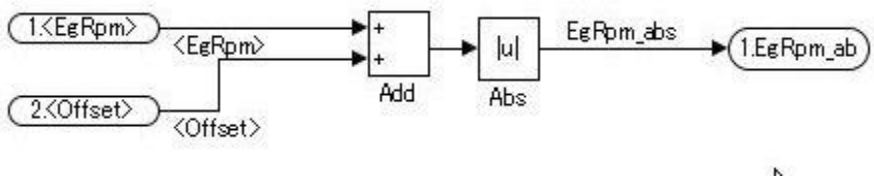
4.2.6. na_0005: Port block name visibility in Simulink models

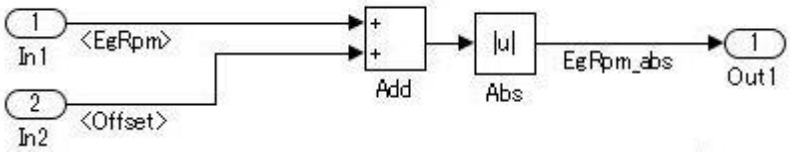
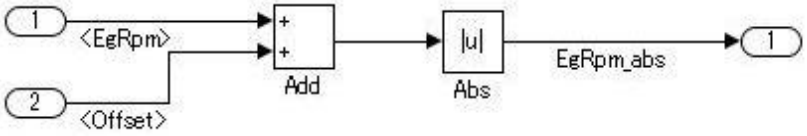
ID: Title	**na_0005:** Port block name visibility in Simulink models
Priority	Strongly Recommended
Scope	NAMAAB
MATLAB Version	ALL

Prerequisites	
Description	For the display of Inport and Outport block names, select either jc_0082 or jc_0083 and apply uniformly. However, understanding the benefits of each of the rules outlined below, they can also both be used depending on the process or the layer and subsystem type. (It is important to clearly define the rule for the separate usages) For instance, they could be used based on rules like the following. ● When creating on the premise of RCP which extends the functionality of the model at the beginning of the process. ➢ In the case of Atomic Subsystem + function, jc_0082 ➢ Virtual Subsystems allow jc_0083 ➢ Allow use of either for Atomic Subsystem＋auto, inline. ● Implementation code stage ➢ Make all conform to jc_0082. These are some examples of what is possible. ● Benefits of each rule ◆ jc_0082 These rules are in order to avoid connection errors in layered subsystems. When connecting subsystems after designing functions for individual subsystems separately, this is effective in avoiding connection errors, by connecting signals and ports so that their names match. ◆ jc_0083 The purpose of this rule is to reduce man-hours. The advantage of this rule becomes apparent when used for layering by building subsystems through the selection of already existing specific blocks. In this case connection errors do not occur as only a layer of existing blocks is dropped. As the subsystem creation function has no function that automatically copies signal names to block names, unification of block names and signal names requires man hours. Moreover, as there is the option of changing signal names in the initial stage of the process, the concern is that complying with jc_0082 will lead to errors because of an increase in subsequent hours needed for correction and correction oversights. Also, the subsystem creation function does not have a correction function for port icon display or a function for automatically replacing block names with signal names. Whether using jc_0082 or jc_0083, man-hours will be required for that. For work that is done this frequently, an automatic correction function using an API should be used.
Notes	The following 3 rules were highly related. ● **na_0005:** Port block name visibility in Simulink models ● **jm_0010: Inport and Outport block names** ● **jc_0081: Display of Inport and Outport block icons** Furthermore, 2 techniques were described in na_0005. As it was difficult to know how these techniques should be described, this rule imposes a choice of one of the two techniques, as is indicated in the old na_0005. We have then extracted 2 separate new rules, jc-0082 and jc-0083, to describe these 2 techniques separately. **Rules jm_0010 and jc_0081 have been deleted as they have been combined with this rule and split between jc_0082 and jc_0083.**
See Also	MISRA AC SLSF 036-C
Last Change	V4.0

4.2.7. jc_0082: Display of Inport and Outport block names 1

ID: Title	**jc_0082: Display of Inport and Outport block names 1**
Priority	Strongly Recommended

Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Please set the port block name, the signal name and the icon name as a set. If the Inport or the Outport block has a signal name, ● the name of the signal line connected to the Inport or the Outport should be the same as the block name. ● The Inport or the Outport block name should be displayed. ("Format", "Block name not displayed" not possible). ● "Port number" should be selected to display the icon for the Inport or Outport block name. Signal names refer to both attaching a label name or the existence of a legacy name. The rule above is the same for scalars, vectors and busses. Exception: ● The rule does not need to be adhered to inside library subsystems, masked subsystems or subsystems where reusable functions have been set. ● Block names do not have to conform completely to the signal name. Please register any differences in numeric characters used for suffixes or prefixes as specific characters. Often used suffixes and prefixes are _in for Inport blocks and _out for Outport blocks. Any prefix or suffix can be used for ports, but consistent prefixes must be selected. Correct: Selecting a port number  Appearance of the subsystem from above  Incorrect: "Port number and signal name" is selected for the display of the port block icon.  Incorrect: The port block name and the signal name are different.

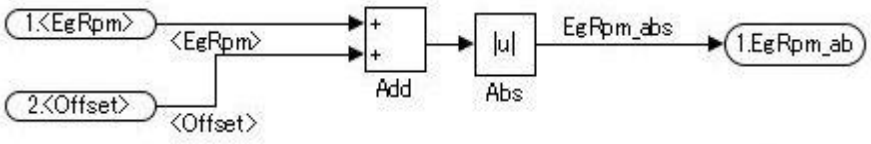
	 Incorrect: The port block name is not displayed. 
Notes	To match it with the checking content of the current guideline checker na_0005 - 1 has been formulated independently. The purpose of this rule is to avoid connection errors in layered subsystems. Function layers in controller models sometimes connect tens of signal lines, and it is important to build a model where connection errors can be seen at one glance. It is important, when building a safe system that is an automobile, that its design allows to discover simple errors at a glance. When this rule is applied, it is difficult to automatically judge whether all signal names or block names are correct after they have been automatically replaced. Signal names, even when they are legacy names, cannot be automatically judged as to whether the block name may be replaced from the signal name or whether there are any connection errors. While extracting the differing parts of the names and confirming them one by one, a decision needs to be taken on whether to employ the block name or the signal name. If either is unilaterally changed, they can be made uniform with the automation tool.
See Also	MISRA AC SLSF 036-C
Last Change	V4.0

4.2.8. jc_0083: Display of Inport and Outport block names 2

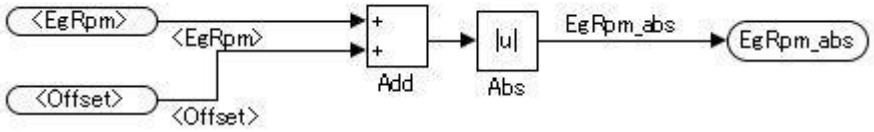
ID: Title	**jc_0083: Display of Inport and Outport block names 2**
Priority	recommended
Scope	NAMAAB
MATLAB Version	ALL
Prerequisites	
Description	Please set the port block name, the signal name and the icon name as a set. If the Inport or the Outport block has a signal name, ● the signal name should not be the same as the block name in the Inport or the Outport block. The block name should conform to a number of specific fixed block names. ・ For instance, in value and out value Inport standardly named in Simulink ● Icon display of Inport or Outport block should select "Signal name" or "Port number and signal name". ● Block names for Inport or Outport blocks should be set to not displayed (when changing the icon display settings described above, the default setting for block name display changes to OFF, and the user does not need to perform any specific operation). Signal names refer to both attaching a label name or the existence of a legacy name. The rule above is the same for scalars, vectors and busses.

Exception: Names cannot be set to non-display inside library subsystem blocks.
This is used when the signal name is prioritized and no meaningful name is attached to the block name.

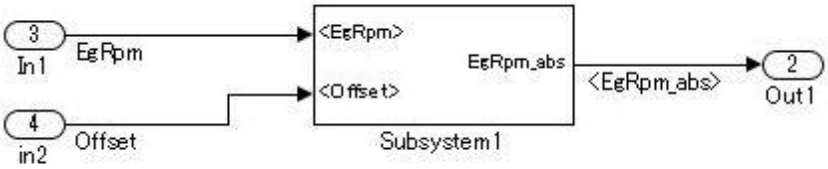
Correct: Use the port number and the signal name display for the icon label
The icon display for the Inport of the Outport block is selected as the signal name



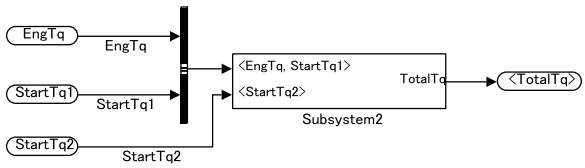
Correct: The signal name display is used for the icon label



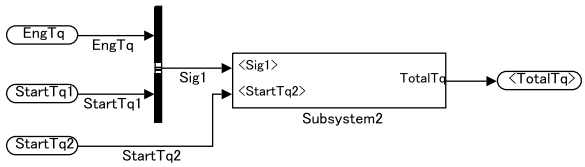
Appearance of the subsystem from above



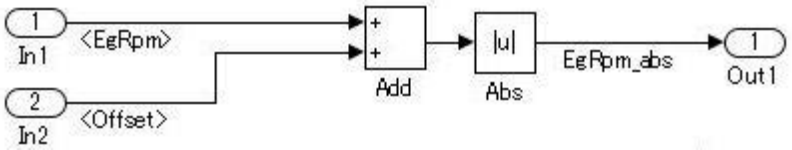
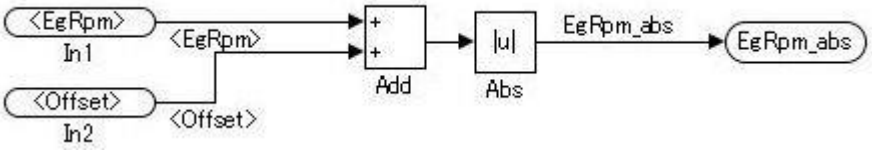
When the "signal name" is given for the icon display, the signal name for inherited signals is given between < >, but no < > are used for signal names if a direct label has been entered. Please note that, if entering a list of signals at a bus, extremely long names will be displayed if no name is given to the bus.
Reference 1: No name given to bus



Reference 2: Name given to bus



Incorrect:
If the signal line of the Inport or Outport block has a name, the icon display only has a port number, and the block name is displayed.

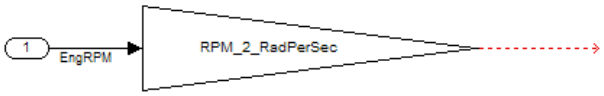
	 Incorrect: After using a port number and signal name display on the icon label, this has been changed to block name display for the port block. 
Notes	Manually changing the icon display takes time and effort. Automatic correction using a Simulink API surely is desirable. In that event, an API for automatically conforming the specifically defined port block name to signal name should also be used.
Last Change	V4.0

4.2.9. db_0097: Position of labels for signals and busses

ID: Title	**db_0097: Position of labels for signals and busses**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	The labels must be visually associated with the corresponding signal, and not overlap other labels, signals or blocks. Labels should be located close to the corresponding source or destination block below the signal line.
See Also	MISRA AC SLSF 027A
Last Change	V2.0

4.2.10. db_0081: Unconnected signals, block inputs and block outputs

ID: Title	**db_0081: Unconnected signals, block inputs and block outputs**
Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	A model must not include the following. ● Subsystems or basic blocks with unconnected inputs ● Subsystems or basic blocks with unconnected outputs ● Unconnected signal lines If unconnected blocks/signal lines are required, they must conform to the following. ● Unconnected inputs should be connected to a ground block.

	• Unconnected outputs should be connected to a terminator block. **Correct:**  **Incorrect:** 
Notes	By using addterm('sys') command, Terminator blocks and Ground blocks are added to the terminal which is not connected to sys in Simulink block diagram. By executing this operation, the model compliant to guidelines can be realized easily. However, distinguishment of intended block or the block which is forgotten to connect becomes impossible. To enable identification of them after this operation, please add annotations near those blocks or change give identifiable names to those blocks. Or please change size of these blocks so that it is clear these blocks were intentionally added.
Last Change	V2.0

4.3. Use of of Blocks

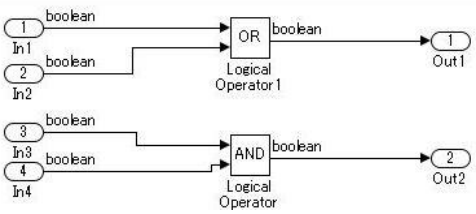
4.3.1. na_0003: Simple logical expressions for If condition blocks

ID: Title	**na_0003: Simple logical expressions for If condition blocks**
Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	A logical expression may be implemented within an If condition block instead of building it up with logical operation blocks, if the expression contains two or fewer primary expressions. A primary expression is defined as one of the following: • An input • A constant • A constant parameter • A parenthesized expression. Except for zero or 1 instances of: < , <=, >, >=, ~=, ==, ~, no operator is included. (See examples below) Exception: A logical expression may contain 3 or more primary expressions if both of the following are true: • The primary expression are all inputs • Only one type of logical operator is present Examples of acceptable exceptions: • u1 | u2 | u3 |u4 | u5 • u1 & u2 & u3 & u4 Examples of primary expressions: • u1 • (u1 > 0) • (u1 <= G)

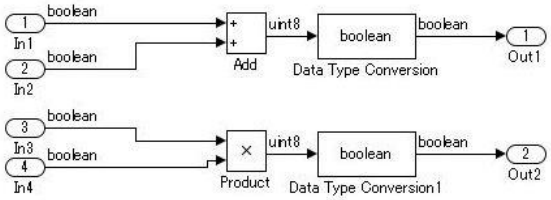
	• (u1 > U2) • (~u1) Examples of acceptable logical expressions: • u1 | u2 • (u1 > 0) & (u1 < 20) • (u1 > 0) & (u2 < u3) • (u1 > 0) & (~u2) Examples of unacceptable logical expressions: u1 & u2 | u3 — Too many primary expressions. Two kinds of operators exist. u1 & (u2 | u3) — Unacceptable operator within the primary expression. In parenthesized expression,only relational operators can be used. (u1 > 0) & (u1 < 20) & (u2 > 5) — Too many primary expressions that are not inputs. Allowed number of primary expressions is two or less. (u1 > 0) & ((2*u2) > 6) — Unacceptable operator within the primary expression Multiplication is executed in parenthesized expression. In these cases, the primary expression must be computed and entered outside the If Condition block.
Last Change	V2.2

4.3.2. na_0002: Appropriate implementation of fundamental logical and numerical operations

ID: Title	**na_0002: Appropriate implementation of fundamental logical and numerical operations**
Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Operations must be performed using the appropriate blocks for logical and numerical operations. 1. No numerical values may be input on blocks that are awaiting logical values. 2. No logical values may be input on blocks that are awaiting numerical values. Detailed explanation ● A logical output should not be directly connected to the inputs of blocks that process numerical inputs. ● The result of a logical expression parameter should not be processed with a numerical operator. ● This guideline for logical operations also applies to enumerated data types. **Correct:**

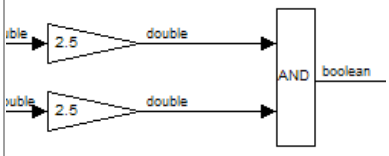


Incorrect:



- Blocks for performing logical operations may not be used for performing numerical operations.
- A numerical output should not be connected to the inputs of blocks that process logical inputs.

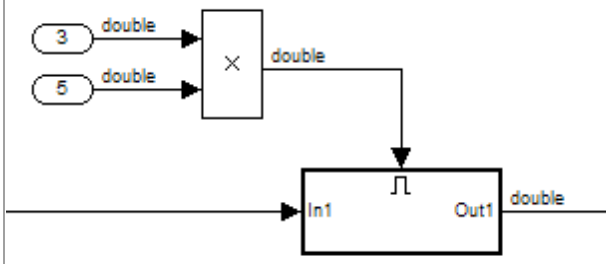
Incorrect:



- Blocks for performing numerical operations may not be used for performing logical operations.

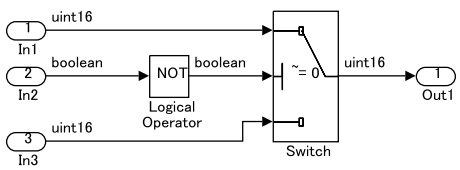
Incorrect:

Although Inputs other than logical values can be made, the Enable Port is a block that awaits logical signals for which only On/Off exists.
Product blocks perform double and double operations, but as it connects the numerical operations result to the block that awaits the logical value called Enable Port, the Product block performs the logical operation.

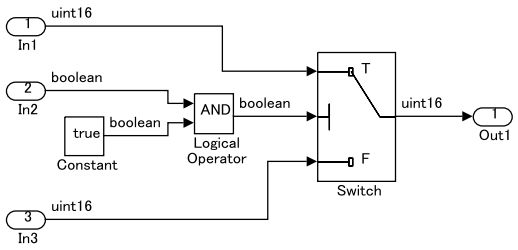
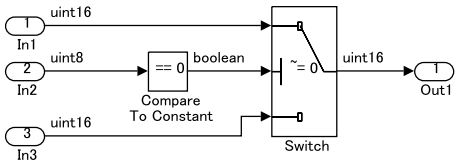
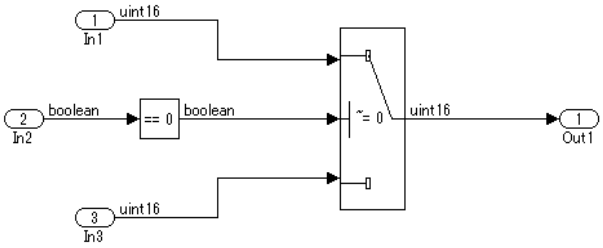
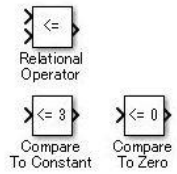


- Boolean should not be applied relational operation.(Boolean signal should not be compared with numerical value(0,1,～) or logical value(true, false))
- To invert boolean value, logical operation NOT should be used.

Correct: Boolean signal is inverted by using logical operation.



Correct: Boolean signal is judged by using logical operation.

	 Correct: Equality of a numerical value and another numerical value is judged.  Signals which are not boolean can be compared with true, false. Incorrect: Boolean signal is compared with numerical value. 
Notes	“Relational Operator”, ”Compare To Constant” and “Compare To Zero”are the blocks that expect numerical input.  Although true, false tend to be considered as equal to numerical 0,1, they mean 0, other than 0. Therefore relational operator should not be applied to boolean signal.
See Also	
Last Change	V4.0

4.3.3. jm_0001: Prohibited Simulink standard blocks inside controllers

ID: Title	**jm_0001: Prohibited Simulink standard blocks inside controllers**
Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	● Controller models must be designed from discrete blocks. The MathWorks "Simulink Block Data Type Support" table provides a list of blocks that support code generation. >The table will be displayed by entering the command showblockdatatypetable.

Please use the blocks that are listed as "Code generation support" in your design. Even if the blocks are subject to code generation support, do not use them for mass production code in the following cases.

- It is dependent to continuous time
- It refers to non-finite values (Inf, -Inf, NaN)
- It includes measuring code that is only suitable for rapid prototyping

- In addition to the blocks defined by the rule above, please do not use the following blocks.
  - Use of the following Sources blocks is prohibited.

Sources are not allowed:	
Sine Wave	
Pulse Generator	
Random Number	
Uniform Random Number	
Band-Limited White Noise	

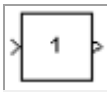
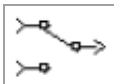
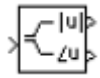
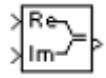
- Sources blocks that are allowed

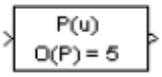
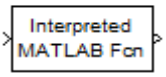
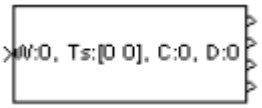
The Sources block group is formed by blocks that can all generate code, but the blocks that can generate mass production code are limited to the following.

- Constant
- Enumerated Constant
- Ground
- Inport

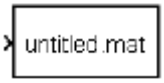
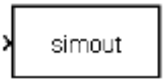
- Use of the following additional blocks is prohibited.

The MAAB Style Guide does not recommend the use of the following blocks. This list can be extended by individual companies.

Slider Gain	
Manual Switch	
Complex to Magnitude-Angle	
Magnitude-Angle to Complex	
Complex to Real-Imag	
Real-Imag to Complex	

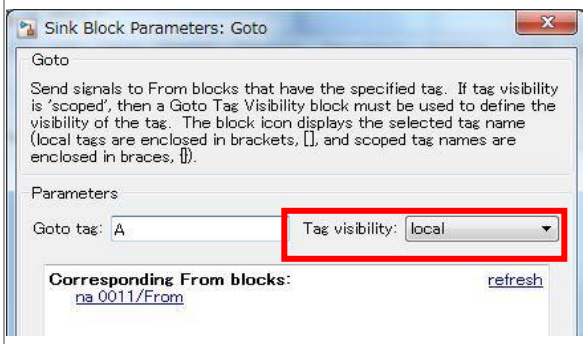
	Polynomial	
	MATLAB Fcn(1)	
	Goto Tag Visibility	
	Probe	
Notes	In (1)R2011a, the block name "MATLAB Fcn" was renamed to the block name "Interpreted MATLAB Function".	
Last Change	V2.2	

4.3.4. hd_0001: Prohibited Simulink sinks

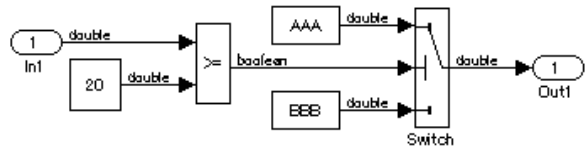
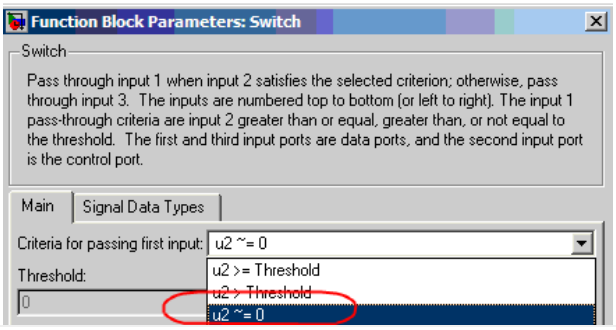
ID: Title	**hd_0001: Prohibited Simulink sinks**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	**Controller models must be designed from discrete blocks.** **Use of the following Sink blocks is "prohibited".** To File  To Workspace  Stop Simulation 
Notes	Simulink Scope blocks and Display blocks can be used in the model diagram. Please consider using Simulink Signal logging and Signal and Scope Manager for data logging and reference requirements.
Last Change	V2.2

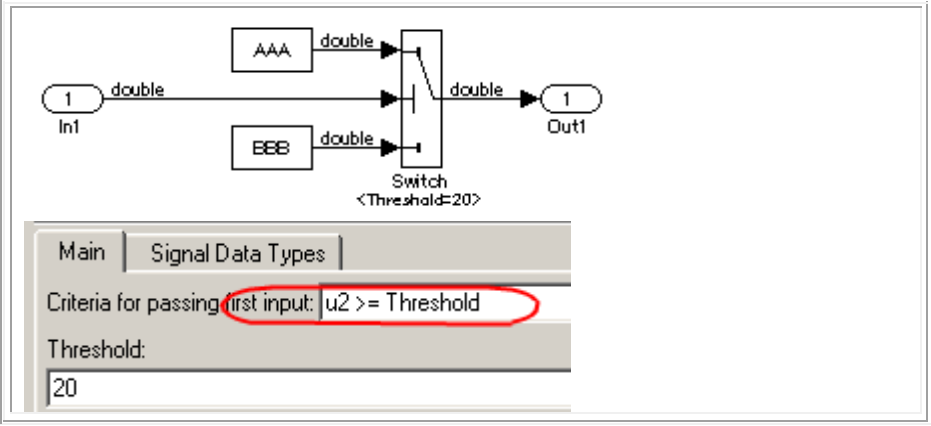
4.3.5. na_0011: Scope of Goto and From blocks

ID: Title	**na_0011: Scope of Goto and From blocks**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	● Goto blocks must use local scope.

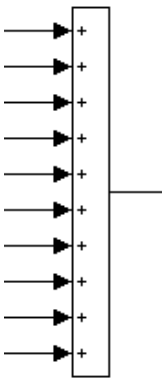
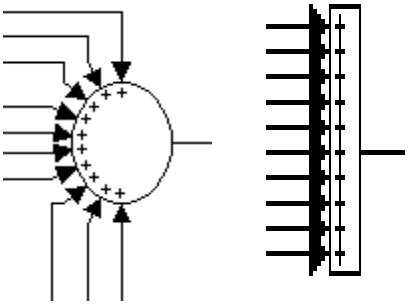
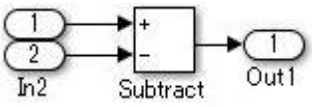
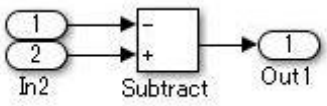
	 Setting tag visualization to global sometimes inhibits subsequent changes from virtual to non-virtual subsystem. Not using them inside a controller model is therefore preferable.
Notes	Goto and From global tags can only be used outside the Atomic Subsystem. When Goto and From are used globally, no Atomic Subsystem is present in the layers above. Case of using From Goto global tag at outside of controller for the connection of controller and plant model is not subject to this rule. **Same As jc_0161: Use of Data Store Read/Write/Memory blocks**
Last Change	V4.0

4.3.6. jc_0141: Use of the Switch block

ID: Title	**jc_0141: Use of the Switch block**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	The Switch block must be used under the following conditions ● The Switch condition, input 2, must be a Boolean type. ● The block parameter "Conditions for the passing through of the first input" should be set to u2~=0. **Correct:**   **Incorrect:**

	
Last Change	V2.2

4.3.7. jc_0121: Use of the Sum block

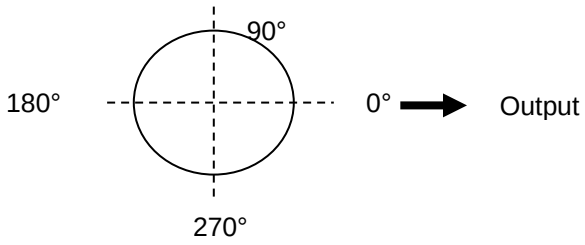
ID: Title	**jc_0121: Use of the Sum block**
Priority	Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	Use conditions of the Sum block ● A rectangular shape should be used. ● The size should be adjusted to ensure there is no input signal overlap. ● Use the + mark for the first input. **Correct:**  **Incorrect:** 1) 2)  1) When using a round shape, the input cannot use any angles other than 90 degrees, 180 degrees, and 270 degree. 2) Since the mark has overlapped, it cannot distinguish. **Correct:**  **Incorrect:**  The 1st input is using the mark of －.

	This spoils readability.

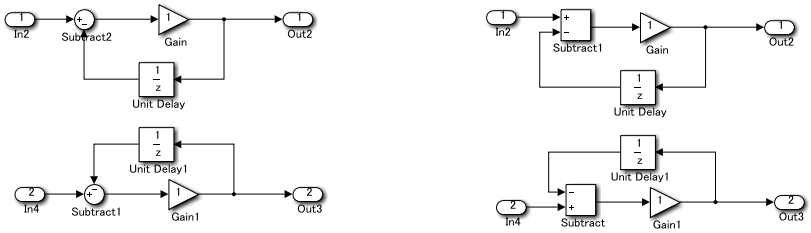
Circular shapes can be used for feedback loops. The following 3 conditions must be adhered to when this is used.

- Please keep the number of inputs up to 2-3.
- The inputs should be positioned at 90°, 180°, 270°.
- The output should be positioned at 0°.

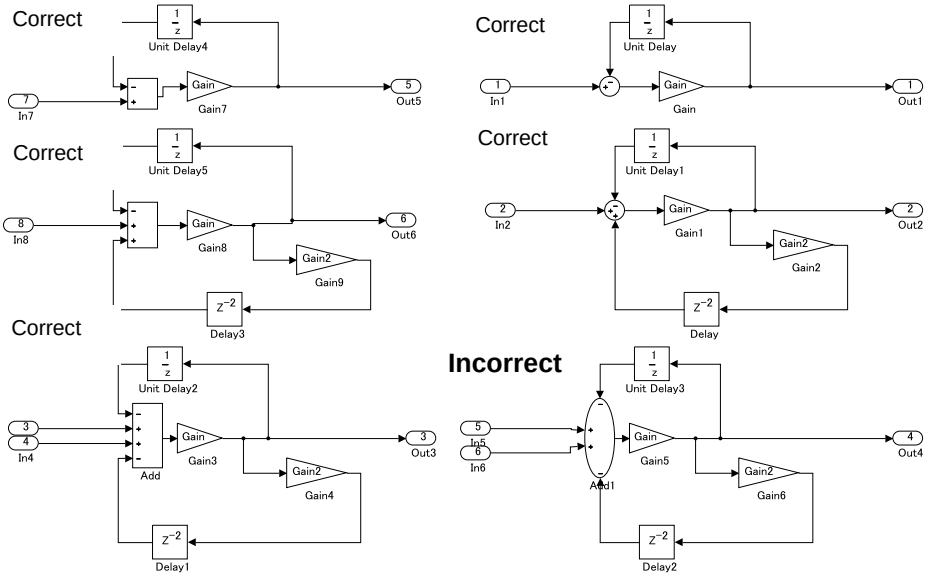
Whether feedback loops are rectangular or circular, the - mark may be used for the first input.



Correct:



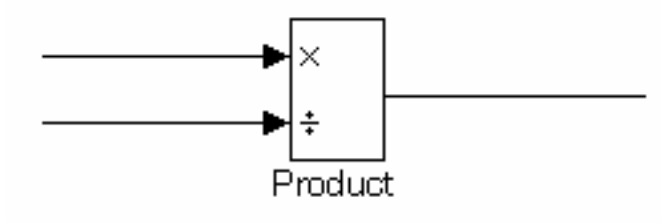
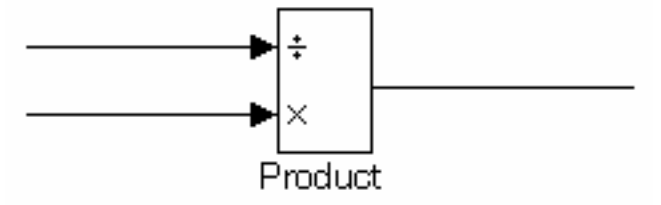
Other notation examples:



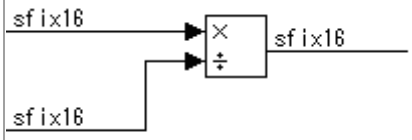
Incorrect:
When using a round shape, the input cannot use any angles other than 90 degrees, 180 degrees, and 270 degree.

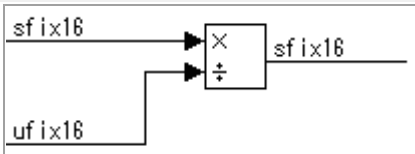
See Also	MISRA AC SLSF 010A
Last Change	V4.0

4.3.8. jc_0610: Operator order for Product block

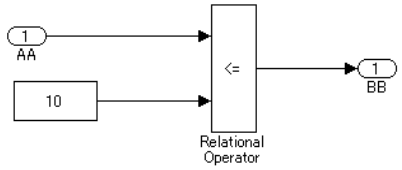
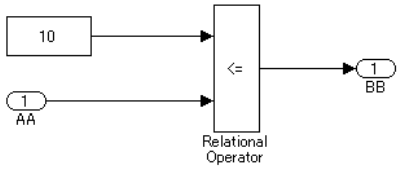
ID: Title	**jc_0610: Operator order for Product block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	ALL
Prerequisites	
Description	If a block is set as a divisor, the first input should be multiplied (*). Correct:  Incorrect: 
Notes	As for jc_0121, it is assumed that the reason that there is no mention of a feedback group is that there are no cases where the return destination is directly the Product block.
See Also	MISRA AC SLSF 010B
Last Change	V4.0

4.3.9. jc_0611: Input signal sign during product block division

ID: Title	**jc_0611: Input signal sign during product block division**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	In the fixed-point model, if division is incorporated into the arithmetic expression, the sign is the same as the input signal type. Correct: The input signal sign is the same.  Incorrect: The input signal sign is different.

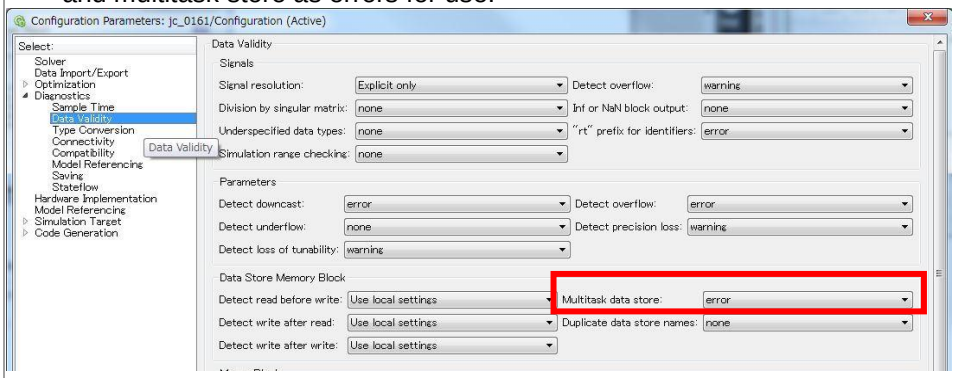
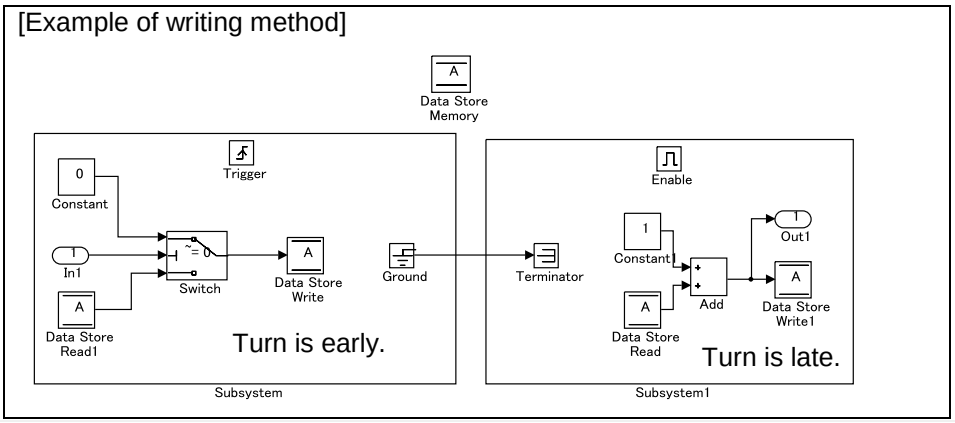
	
See Also	
Notes	In division arithmetic, various utility functions are created when a fixed-point code is generated. While a utility function is created for each LSB, the problem of LSB precision may make it difficult to suppress the number. In addition, if the type is different, the number can easily double in size. Unification of types used can be expected to suppress the number of utility functions, to improve ROM efficiency, and to cut down on testing manhours.
Last Change	V4.0

4.3.10. jc_0131: Use of Relational Operator block

ID: Title	**jc_0131: Use of Relational Operator block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	If using the relational operator for comparison of signals and constants, set the constant input to the second (bottom) input. **Correct**  **Incorrect** 
Last Change	V2.0

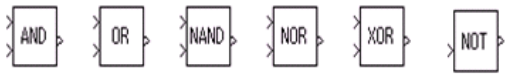
4.3.11. jc_0161: Use of Data Store Read/Write/Memory blocks

ID: Title	**jc_0161: Use of Data Store Read/Write/Memory blocks**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Description	● The use case of data Store Read/Write/Memory is determined. When using it as a memory which memorizes the past value, you should use UnitDelay , Delay block, etc. If UnitDelay is used, when readability will fall, Data Store Read/Write/Memory can be used. Please determine the case used in a project and use the use part of Data Store Read/Write/Memory, limiting. ● Arrangement of Data Store Read Memory To explicitly show the Read and Write scope, position the DSM block in as low a layer as possible. Do not position the DSM in the top layer for no reason

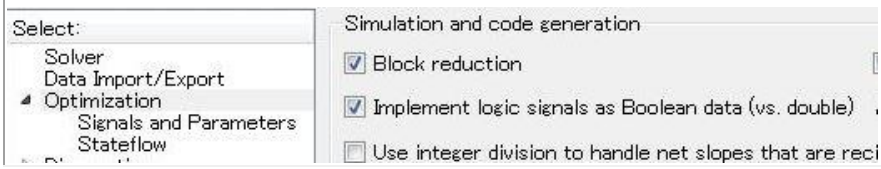
	● diagnosis If using between subsystems running at different rates, set diagnosis, data validity, and multitask store as errors for use. 
Notes	Object block  Know-how for improving readability If Read and Write are positioned in differing subsystems, and the subsystems are not directly wired, using a Ground and Terminator to create a dummy line that directly wires the subsystems can enable visualization of the relationship from a higher level, improving readability. Priority order descriptions are necessary for these subsystems(and blocks). Dummy connection does not bind the turn order. Dummy connection should be drawn based on its priority. 
See Also	hisl_0013: Guideline for using the Data Store block MISRA AC SLSF 005C
Last Change	V4.0

4.3.12. Guideline for using the Logical Operator block

ID: Title	**jc_0621: Guideline for using the Logical Operator block**
Priority	Strongly Recommended

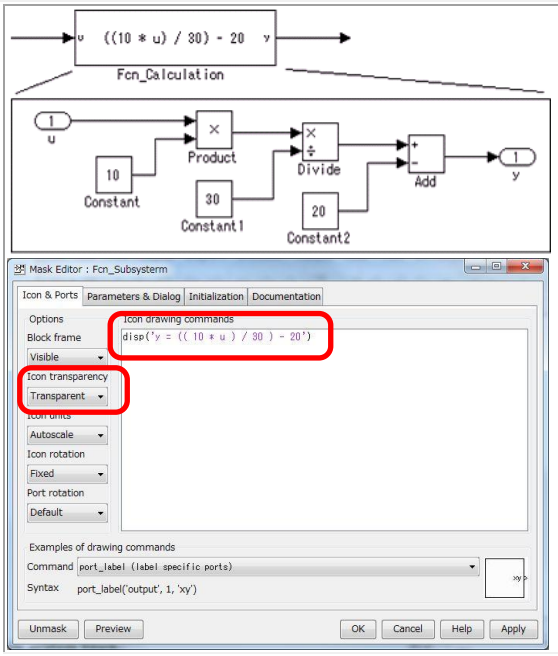
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Unify the Logical Operator block icon shape to either "square" or "characteristics". Unless there is otherwise a particular reason, set to "square". Icon shape: Square  Icon shape: Characteristics 
See Also	
Last Change	V4.0

4.3.13. jc_0011: Optimization parameters for Boolean data types

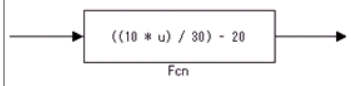
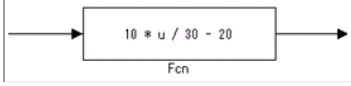
ID: Title	**jc_0011: Optimization parameters for Boolean data types**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	**na_0002: Appropriate implementation of fundamental logical and numerical operations**
Description	The optimization parameter for Boolean signals must be enabled. In the Configuration Parameter Dialog Box, select Use Logic Signal as Boolean Data (vs double) under Simulation and Code Generation of Optimization. 
Last Change	V2.2

4.3.14. jc_0629: Fcn block use limits

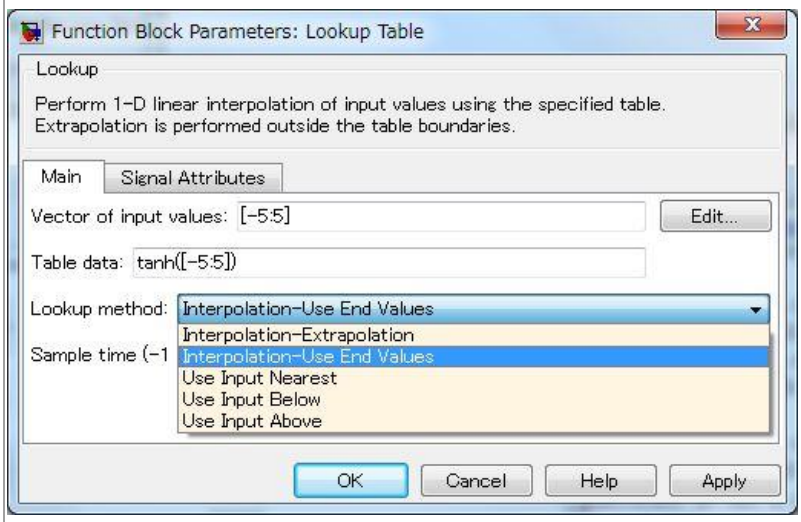
ID: Title	**jc_0629: Fcn block use limits**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	The Fcn block is not used in the Controller Model for the purpose of code generation. If using the Fcn block, use the MathOperation block within the subsystem, and build an expression. Example

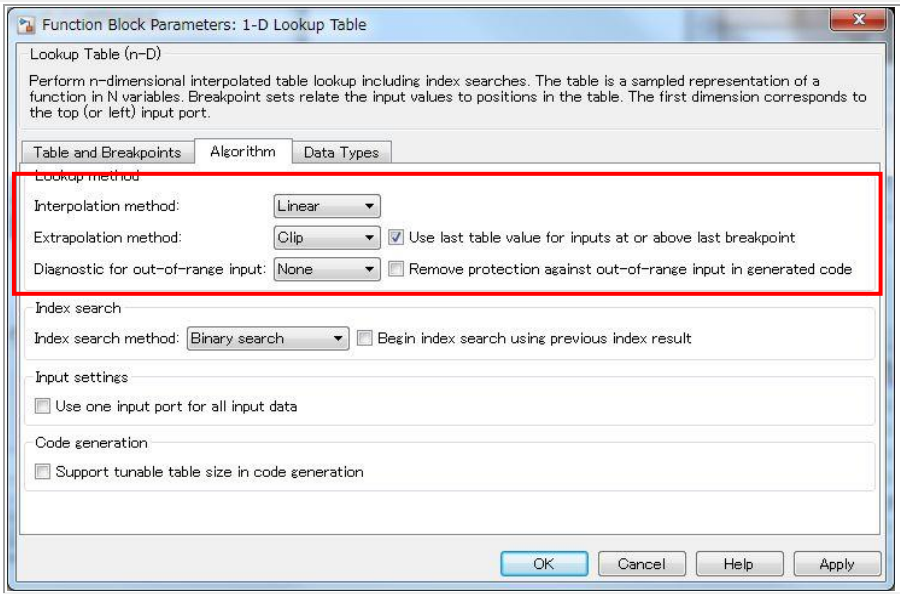
	
Notes	If using an Fcn block, it is advantageous in terms of readability because the numerical expression is displayed from the top. If a subsystem consisting of numerical expressions only has been designed, implementing subsystem masking, and displaying the numerical expression within the disp command, makes it appear equivalent to Fcn, and improves readability from the upper layer.
See Also	MISRA AC SLSF 005B
Update History	V4.0

4.3.15. jc_0622: Guideline for using the Fcn block

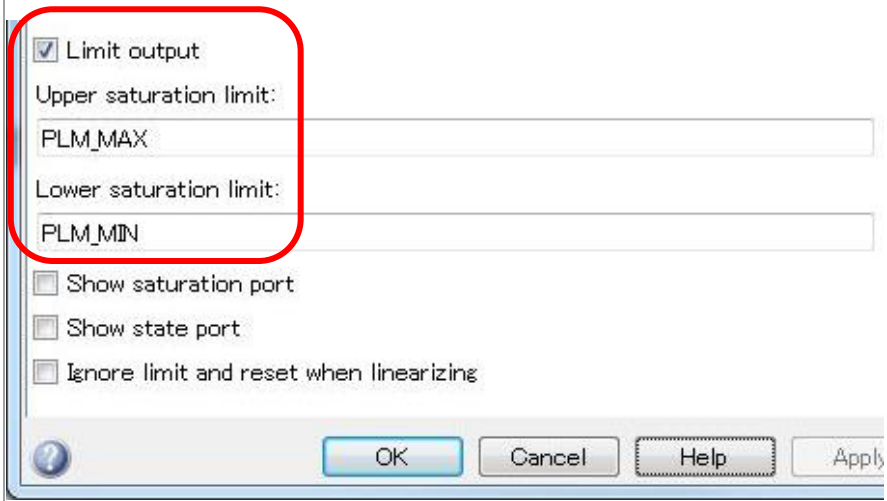
ID: Title	**jc_0622: Guideline for using the Fcn block**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	If using the Fcn block, always enclose in parentheses in arithmetic with priority order. (Rather than blindly rely on the priority order, use parentheses for clarification.) Correct: Since there is a priority order in the Fcn block operation, parentheses are attached.  Incorrect: Even though there is a priority order in the Fcn block operation, parentheses are not attached. 
See Also	
Last Change	V4.0

4.3.16. jc_0626: Guideline for using the Lookup Table system block

ID: Title	**jc_0626: Guideline for using the Lookup Table system block**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	For the lookup manual option in the Lookup Table, Lookup Table 2D, Lookup Table nD, and Lookup Table Dynamic, use "Interpolation - Use Final Value". (R2011a) However, exclude cases when all input and output are a real number (double, single). This rule is not merely for the purpose of preventing overflow of the Lookup Table output (if that is the purpose, use the saturation at integer overflow in the Lookup Table block), it is for the purpose of clearly defining the Lookup Table maximum and minimum values to prevent unexpected results in other operation blocks using the Lookup Table output. ■ Lookup Table block up to R2011a  ■ Lookup Table R2011b and later (same as n-d Lookup Table) Interpolation method: Prohibit 3D spline, and use linear shape. Extrapolation method: Prohibit linear shape and 3D spline, and use clip. Extrapolation option: Check "Use the final break point, or the final table value for input based on it".

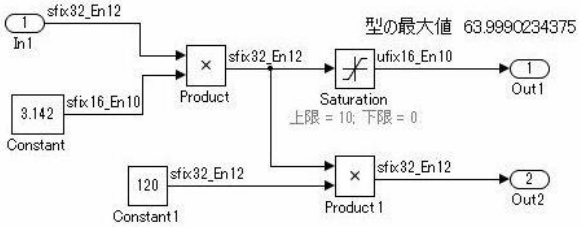
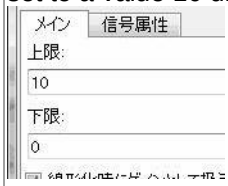
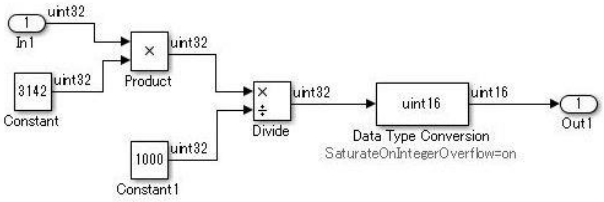
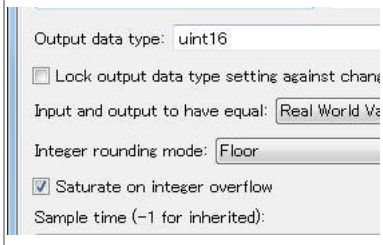
	
See Also	
Notes	The options shown below usable in versions R2011a or earlier do not have upward compatibility with versions R2011b or later. As a result, in these Guidelines it is limited to "Interpolation - Use Final Value". Option name with no upward compatibility ・ Use nearest input ・ Use bottom input value ・ Use top input value
Last Change	V4.0

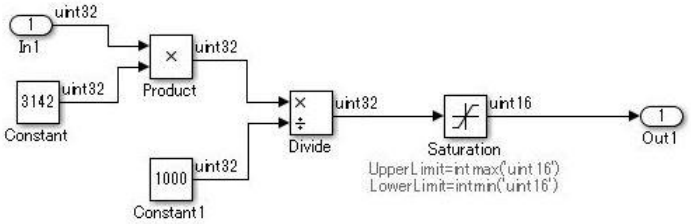
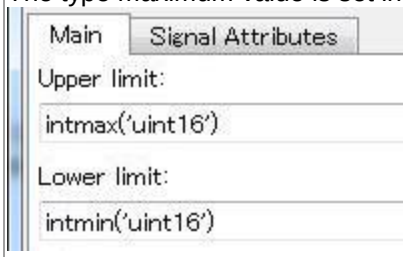
4.3.17. jc_0627: Guideline for using the Discrete-Time Integrator block

ID: Title	**jc_0627: Guideline for using the Discrete-Time Integrator block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	For the Discrete-Time Integrator, set the saturation upper limit and lower limit.  If performing settings for generation of mpt.Parameter and other codes in the parameters,

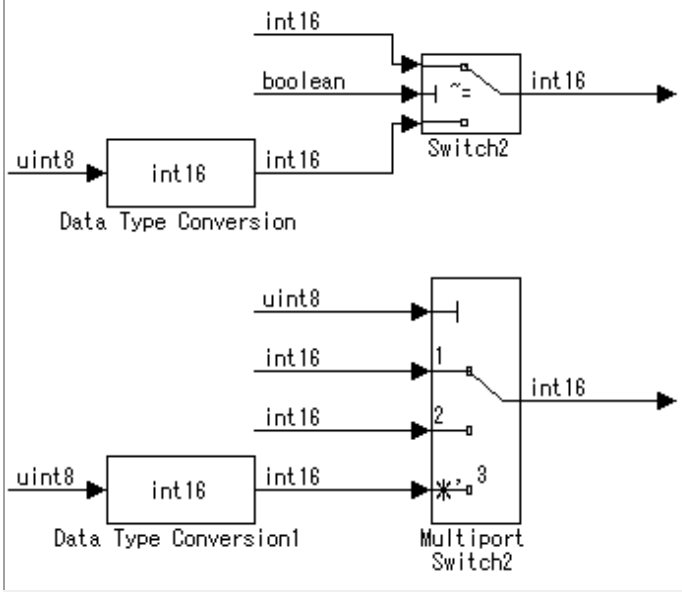
	the data type should be set to auto.
See Also	
Last Change	V4.0

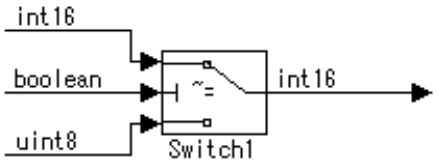
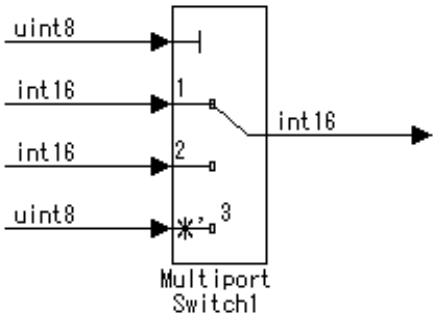
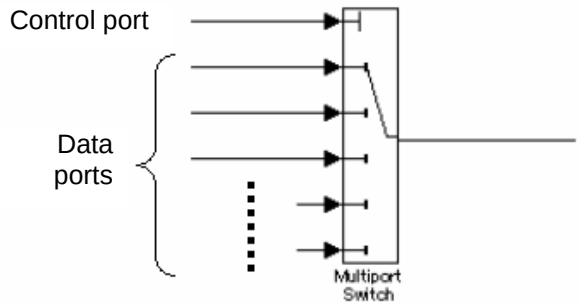
4.3.18. jc_0628: Guideline for using the Saturation Block

ID: Title	**jc_0628: Guideline for using the Saturation block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	For the maximum value and minimum value of Saturation or Dynamic Saturation blocks, use should be limited to to significant values within the maximum and minimum range. If setting the type maximum and minimum for both the Saturation or Dynamic Saturation blocks, use the "Saturation at Integer Overflow" in the Data Type Conversion block. (For details, see **jc_0651**) Correct: A significant value should be used for the Saturation limit value.  In regards to the type maximum value 63.9990234375, the Saturation upper limit value is set to a value 10 differing from the type maximum value.  Correct: If limiting the type maximum and minimum values, use the Data Type Conversion block.  

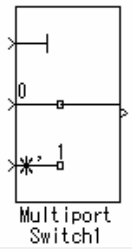
	Incorrect: In the Saturation Block, upper and lower limit processing is performed within the type maximum and minimum ranges after downcasting.  The type maximum value is set in Saturation. 
See Also	MISRA SLSF0002A
Last Change	V4.0

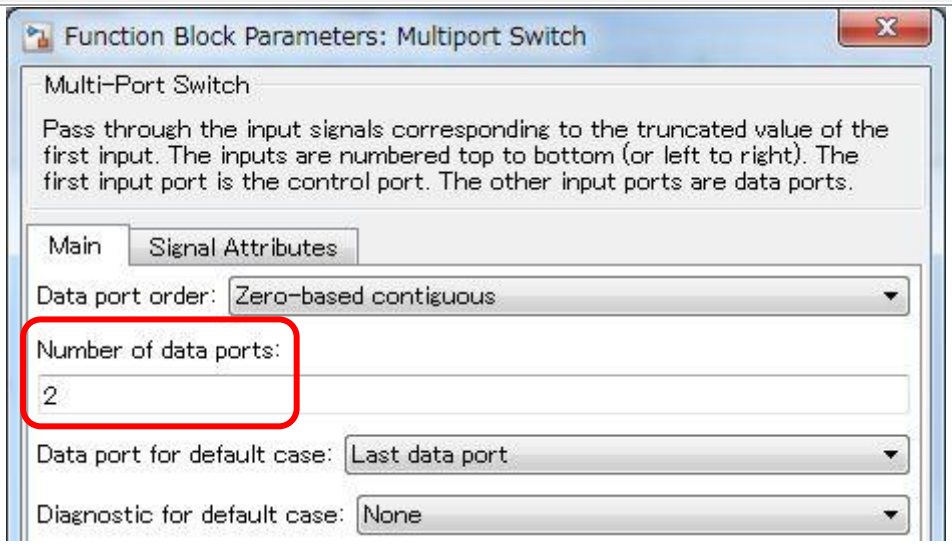
4.3.19. jc_0650: Block input/output data type with switching function

ID: Title	**jc_0650: Block input/output data type with switching function**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	For blocks (Switch, Multiport Switch, Index Vector) with switching functions, use the same data type for data ports and output port. Correct: 

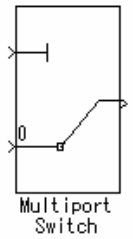
	Incorrect:  
Notes	Signal flow switching port is described as control port, other input ports are described as data ports. 
See Also	
Last Change	V4.0

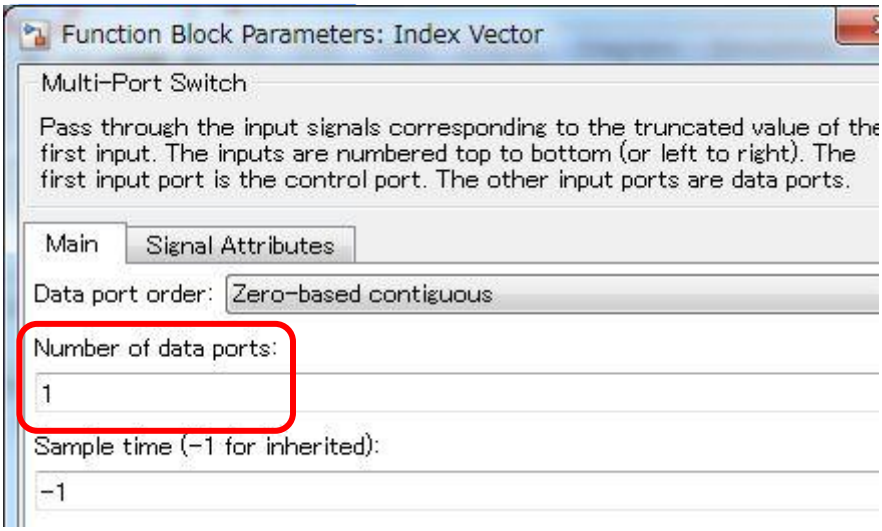
4.3.20. jc_0630: Number of data ports in Multiport Switch block

ID: Title	**jc_0630: Number of data ports in Multiport Switch block**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Set the "data port number" in the "Multiport Switch" block" to two or more. Correct: 



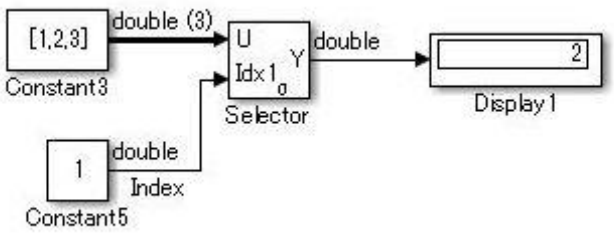
Incorrect:

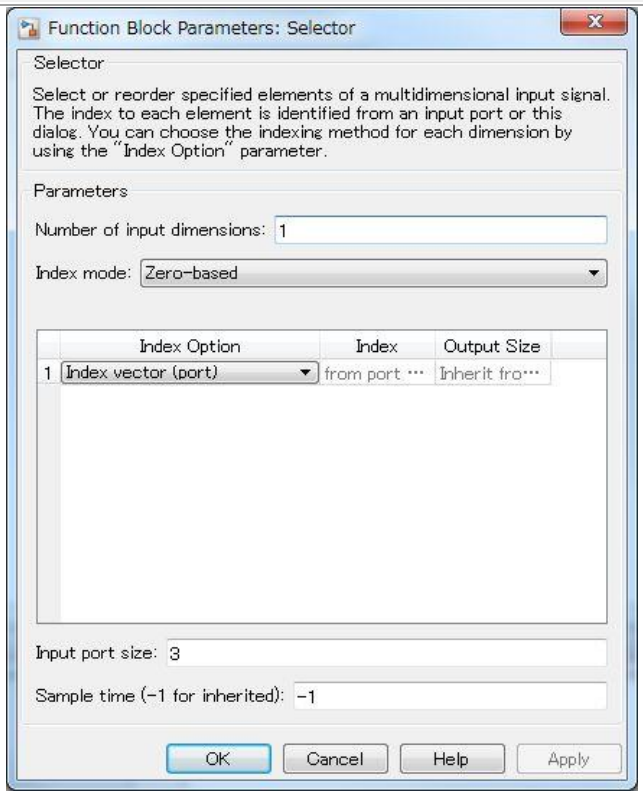




Correct:

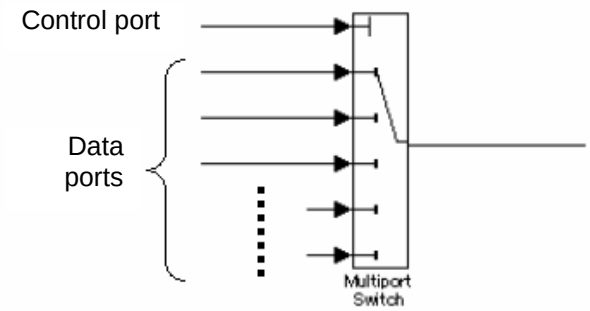
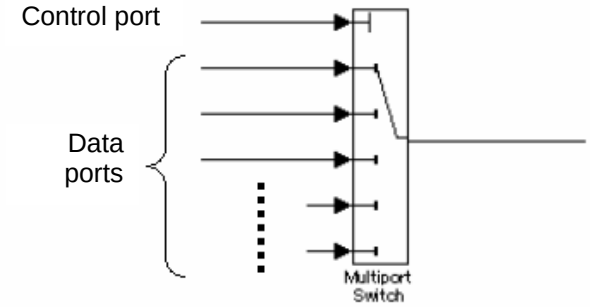
If extracting index elements from the array, use the Selector block.





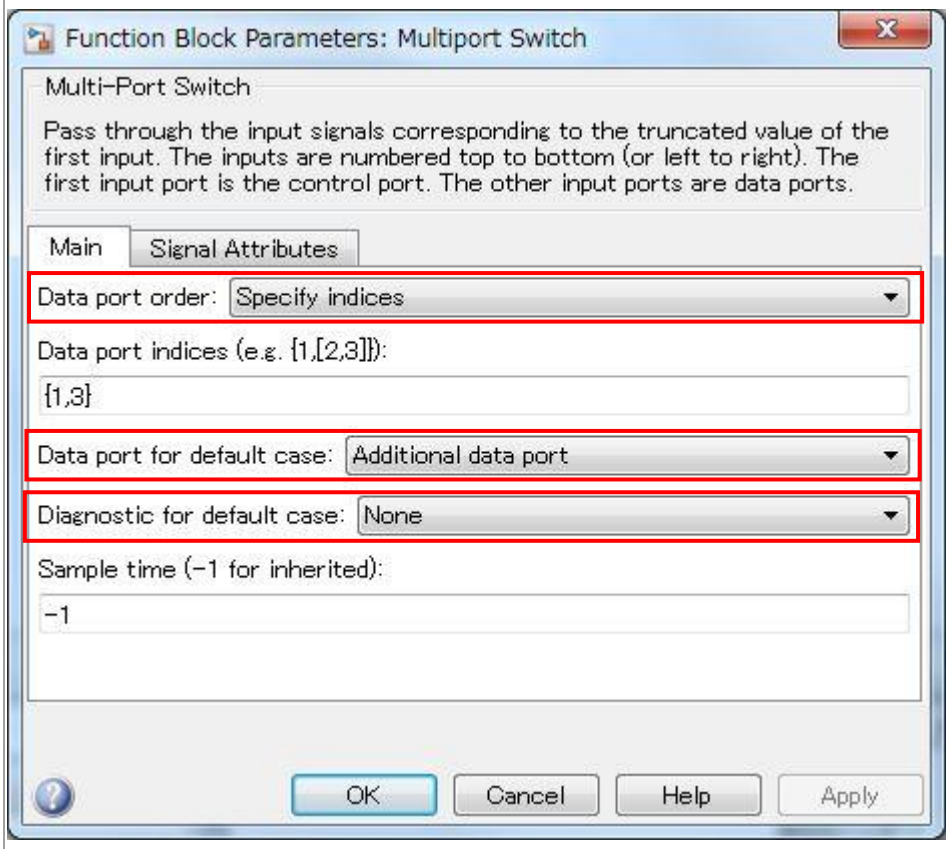
See Also	MISRA AC SLSF 013A
Notes	Only the Index Vector and Multiport Switch option settings differ, and both are blocks that have the same functions. If there have been multiple inputs of the vector signal, output the vector in accordance with the index number. If the number of data ports is one, it will change to a block that extracts scalar from inside the vector. If, without knowing this, the input pattern of the index portion has been reduced to just one, the block should in fact be cut back. However, if the block role has not been recognized, there is a possibility that reducing the port number will be acceptable. In this case, the intended action will not occur. To confirm whether the design intentions were intentionally prepared or unintentionally used, use the Selector block in the block that extracts any single desired element from the vector. In addition, if extracting a specific fixed scalar from the vector, it should be considered that there is a possibility that a path should be used rather than a vector.
Last Change	V4.0

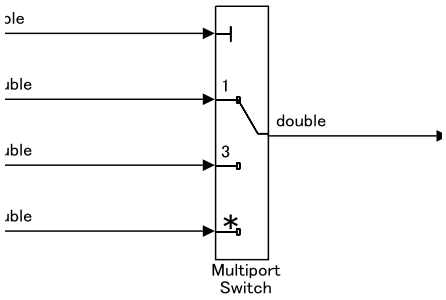
4.3.21. jc_0631: Input of Multiport Switch block to control port

ID: Title	**jc_0631: Input of Multiport Switch block to control port**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Set the input to the "Multiport Switch" block control port to an unsigned integer. Usable data type • uint8, uint16, uint32 • Enumerated data type (does not literally use negative values) 
Notes	
See Also	hisl_0022: Selection of index signal data type MISRA AC SLSF 013B
Last Change	V4.0

4.3.22. jc_0632: Default case port in Multiport Switch block

ID: Title	**jc_0632: Default case port in Multiport Switch block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	If the order of the Multiport Switch block data ports is "index specified", the following settings should be performed: ● Set the default case data ports to "additional data port" ● Set the default case diagnosis to "none"

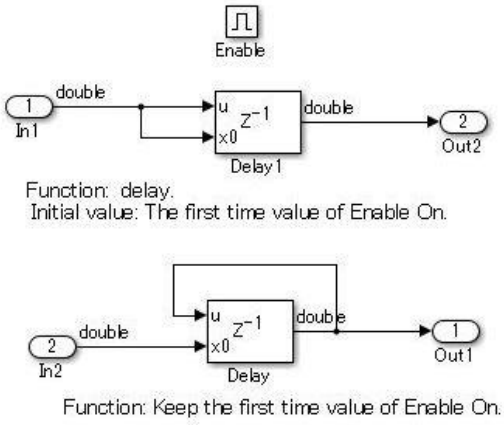
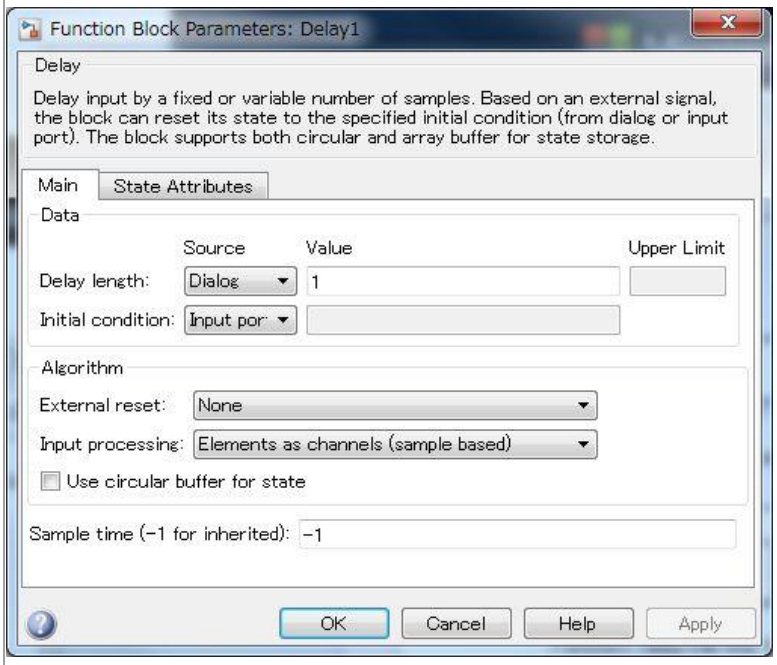


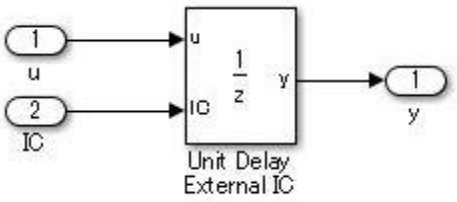
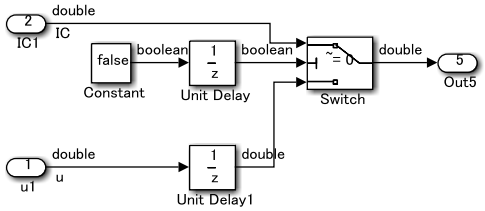


See Also	hisl_0022: Selection of index signal data type
Last Change	V4.0

4.4. Initialization

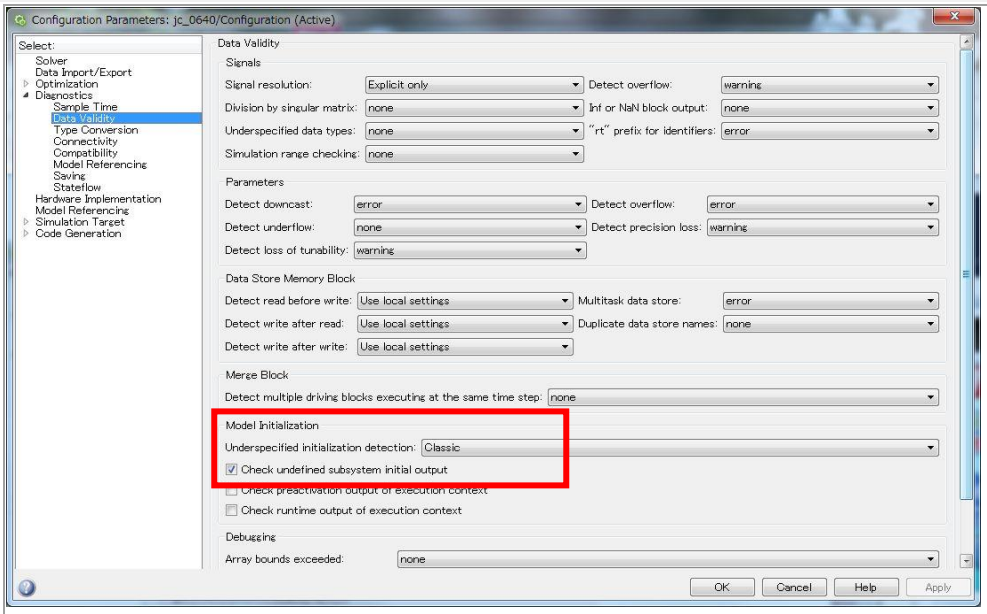
4.4.1. jc_0625: Unification of descriptions of external input values as initial values

ID: Title	**jc_0625: Unification of descriptions of external input values as initial values**
Priority	Recommended
Scope	JMAAB
MATLAB Version	R2011b and later
Prerequisites	
Description	For the Unit Delay, which sets external input values as the initial values, unify to any of the following: 1. Discrete/Delay (Recommended) 2. Additional Math& Discrete/Additional Discrete block group (Unit Delay External IC, etc.) 3. Own configured library Example 1. Delay Block usage example  

	2. Unit Delay External IC  Excluding the Unit Delay External IC mask, the modeling is the same as the 3rd case below. 3. Own configured library 
See Also	
Last Change	V4.0

4.4.2. jc_0640: Detection of undefined initial output

ID: Title	**jc_0640:** Detection of undefined initial output
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	To prevent omission of the initial value setting when system configuration enabling initialized parameters is performed, enable "Specification Shortage Initialization Detection". Select <diagnosis><data validity><specification shortage initialization detection><classic>. And <Check undefined subsystem initial output> flag is "On"

	
Notes	While normally the initial value is not valid, if the conditions are met, it is the Outport block and Merge block that change to blocks that have the initial value. When I meet the following conditions, there is not the need to use this rule. ・The output signal line of the Merge block has the setting of the Simulink object. Because an initial value is set to a signal, so initial value is explicit.
See Also	MISRA AC SLSF 007
Last Change	V4.0

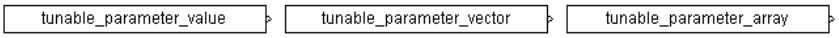
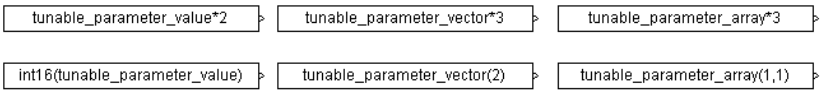
4.5. Block Parameters

4.5.1. db_0112: Indexing

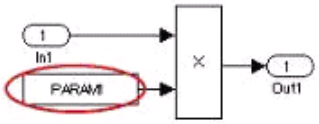
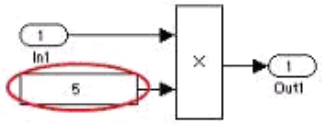
ID: Title	**db_0112: Indexing**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Use a consistent vector indexing method for all blocks. When possible, use zero-based indexing to improve code efficiency.
Notes	If mixing the one-based and zero-based indexing, establish the operations rules, and enable understanding of which index is being used.
See Also	cgsl_0101: Zero-based indexing hisl_0021: Consistent vector indexing
Last Change	V2.2

4.5.2. db_0110: Tunable parameters in basic blocks

ID: Title	**db_0110: Tunable parameters in basic blocks**
Priority	Strongly Recommended

Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	To ensure that a parameter is tunable, it must be entered in a block dialog field as follows: ● Without any expression. ● Without a data type conversion. ● Without selection of rows or columns. **Correct:**  **Incorrect:** 
Last Change	V2.2

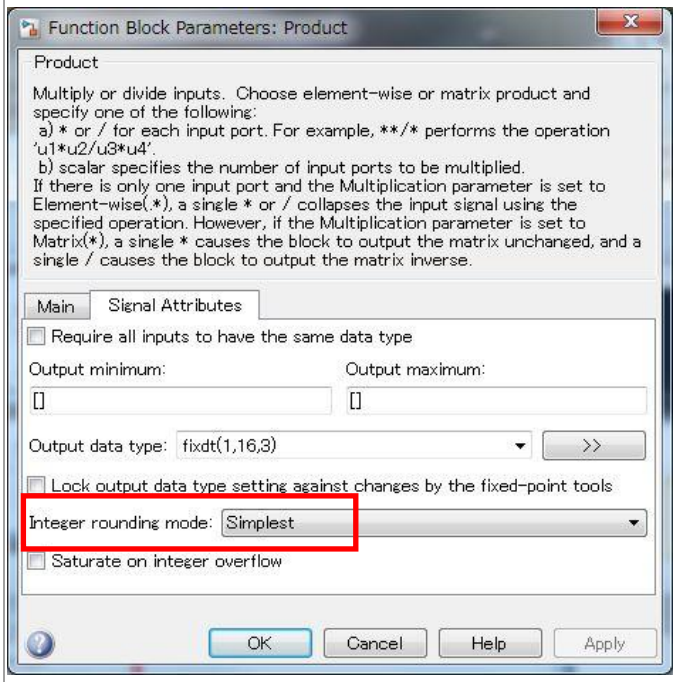
4.5.3. jc_0645: Named constant setting

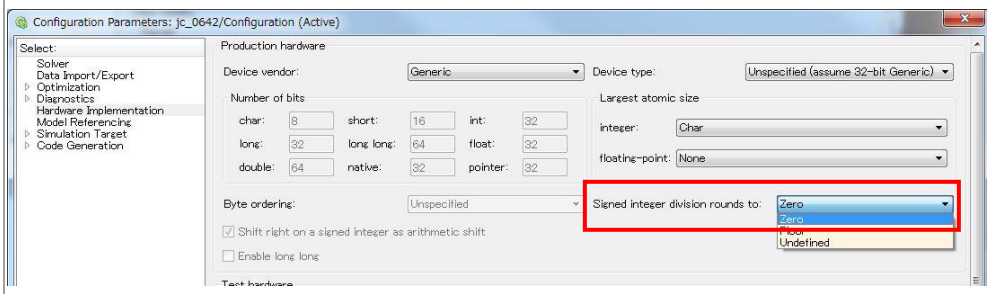
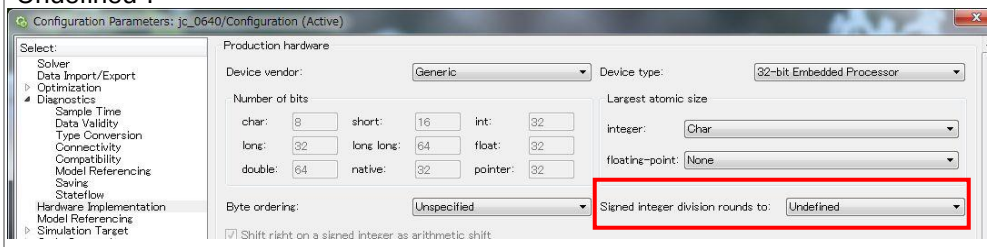
ID: Title	**jc_0645: Named constant setting**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Block parameters that are targets of calibration should be defined as named constants. Examples of parameters outside of calibration target: ● Iniitial value parameter 0 ● Increment, decrement 1 ● Gain block 1 Correct:  Incorrect: 
See Also	MISRA AC SLSF 006B
Last Change	V4.0

4.5.4. jc_0641: Sample time setting

ID: Title	**jc_0641: Sample time setting**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	All blocks with settings related to sample time in the parameters must be set so as to succeed to the input side settings. However, the blocks below are not targeted: ➢ Port block ➢ Atomic Subsystem ➢ Blocks with status variables such as Unit Delay blocks and Memory blocks ➢ Signal conversion blocks such as DataType Conversion and Rate Transition ➢ Blocks that do not have external inputs such as Constant blocks
See Also	MISRA AC SLSF 009D
Last Change	V4.0

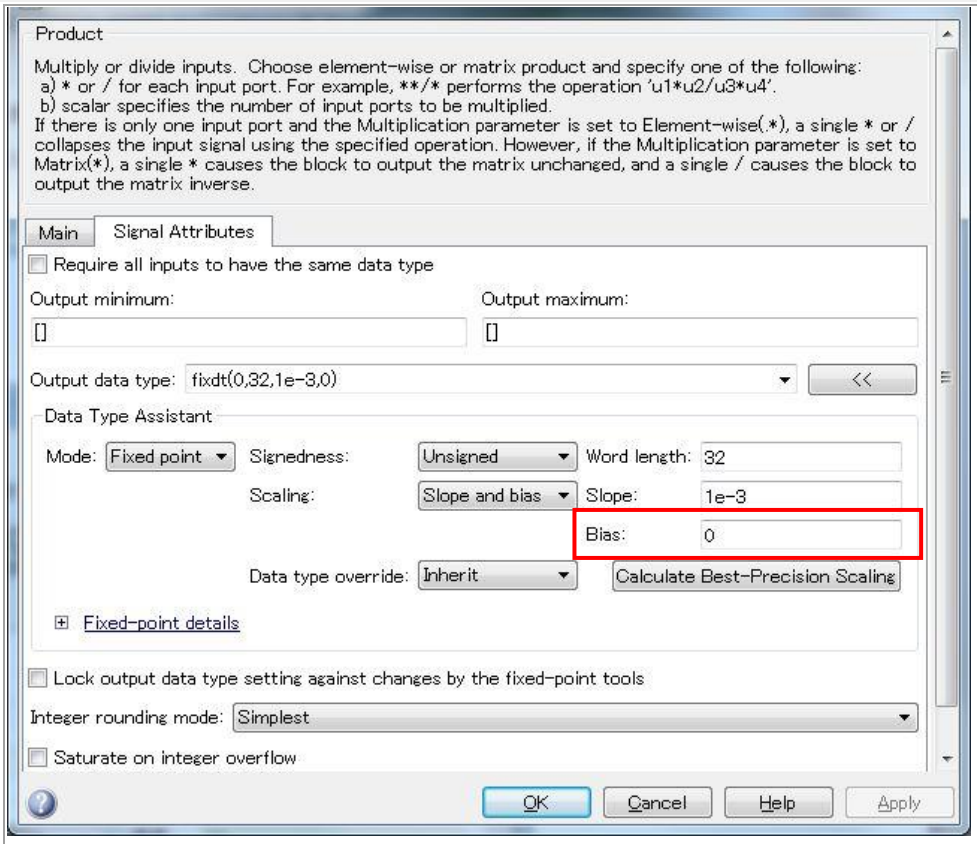
4.5.5. jc_0642: Integer rounding mode setting

ID: Title	**jc_0642: Integer rounding mode setting**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	If "simple" is selected in Integer Rounding Mode, since it is dependent on the configuration hardware setting, it should be set together with the configuration 

	Set <the division rounding of configuration><hardware execution><signed integer>.  **Incorrect** <the division rounding of configuration><hardware execution><signed integer> is set “Undefined”. 
Notes	If "Division Rounding of Signed Integer" option is set to "Simplest", automatically selects either "Rounding in Negative Infinite Direction" or "0", and generates the most efficient code. Effects of <the division rounding of configuration><hardware execution><signed integer> option. "No setting" or “Undefined” (Depends on versions) Select when the compiler action cannot be expressed in either "Zero" or "Rounding in Negative Infinite Direction", or when the action is unknown. "Zero" If the quotient is between two integers, the compiler selects an integer that is closer to 0 for the result. "Rounding in Negative Infinite Direction" If the quotient is between two integers, the compiler selects an integer that is closer to negative infinity for the result.
See Also	
Last Change	V4.0

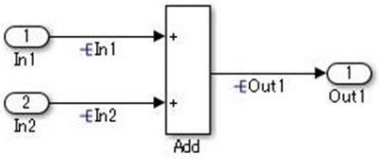
4.5.6. jc_0643: Fixed-point setting

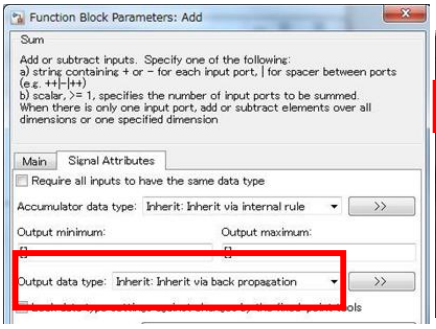
ID: Title	**jc_0643: Fixed-point setting**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	If the fixed-point setting is used for the data type, and "slope and bias" is selected for scaling, the bias must be set to 0

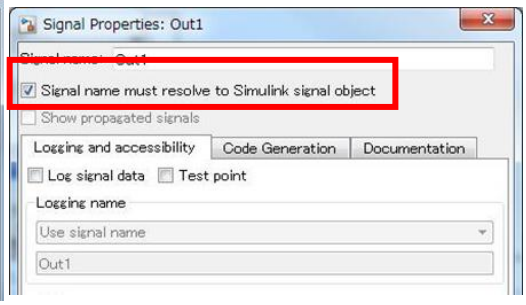
	
See Also	
Last Change	V4.0

4.5.7. jc_0644: Guideline for type setting

ID: Title	**jc_0644: Guideline for type setting**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	If the type is set by data object, the type is not set on the block side. However, this excludes the following: ・Reusable internal part of function ・Data Type Conversion block ・Type setting by fixdt ・Double type, boolean type designation

<table>
<tr><td></td><td> 




</td></tr>
<tr><td>Notes</td><td>Set the signal name in the signal line on the model block side, and associate it with the signal object.
・Inport block...Data type ”auto”
・Outport block...Data type ”auto”
・Sum block...Output data type ”Inherit via back propagation”
The type setting is performed in the data dictionary, while the storage class setting is optional.

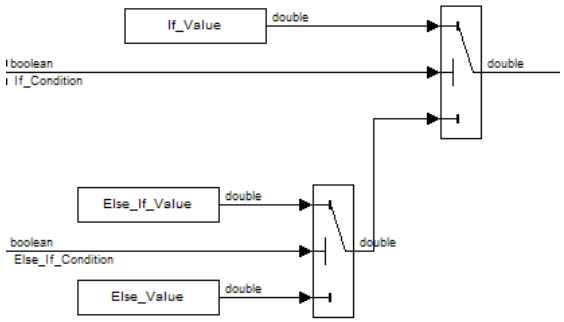
Exceptional items
• Inside of reusable function
Even if all block structures are identical, difference of input/output data type leads to different C source codes and it’s not reuseable. Regarding reusable functions, data types of input/output blocks should be fixed.
• Data Type Conversion block
Purpose of Data Type Conversion is to set data type. If needed, data type is explicitly set by using this block.
• Data types set by using fixdt
If fixed point is selected, detailed setting is necessary since each block can have different data points. Complete control of data type by using only data object is impossible.
• double type, boolean type
Some block type needs explicit setting to boolean.Double type is generally used in plant model and RCP. It is not subject to this rule.
In some cases, embedded software uses double type. However those cases are special case. Since number of double type use must be minimized, careful setting on necessary blocks is needed.</td></tr>
<tr><td>See Also</td><td></td></tr>
<tr><td>Last Change</td><td>V4.0</td></tr>
</table>

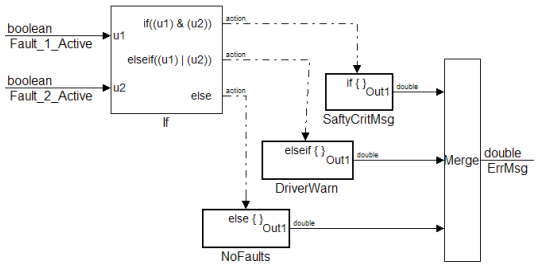
4.6. Simulink pattern

Below is an explanation of the classic patterns often used in the Simulink model.

4.6.1. db_0114: Simulink patterns for If-then-else-if constructs

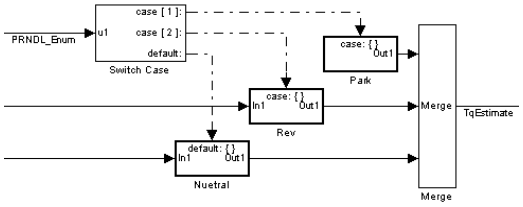
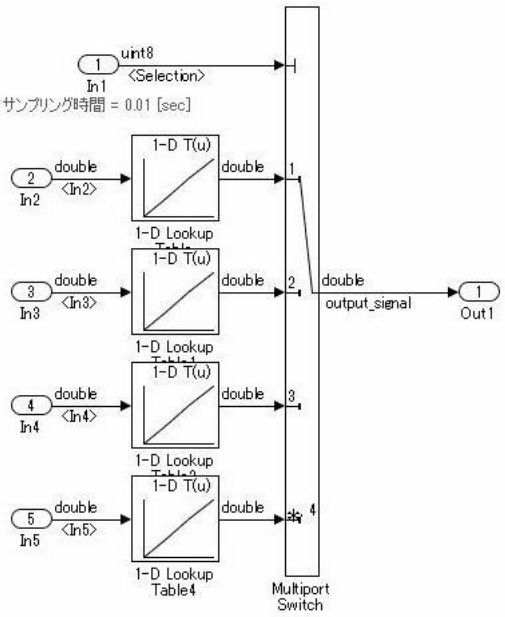
<table>
<tr><td>ID: Title</td><td>db_0114: Simulink patterns for If-then-else-if constructs</td></tr>
<tr><td>Priority</td><td>Strongly Recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>The following patterns should be used for If-then-else-if constructs:
<table>
<tr><th>Functionality</th><th>Simulink pattern</th></tr>
<tr><td>Switch Block is used
IF THEN ELSE IF construct

if (If_Condition)
{
  output_signal = If_Value;
}
else if (Else_If_Condition)
{
  output_signal = Else_If_Value;
}
else
{
  output_signal = Else_Value;
}</td><td> 
</td></tr>
<tr><td>IF THEN ELSE IF construct
using Action Subsystem

if (Fault_1_Active &
Fault_2_Active)
{
  ErrMsg = SaftyCrit;
}
else if (Fault_1_Active |
Fault_2_Active)
{
  ErrMsg = DriveWarn;
}
else
{
  ErrMsg = NoFaults;
}</td><td> 
</td></tr>
</table></td></tr>
<tr><td>Notes</td><td>While listed as an example explanation, If Action Subsystem is normally not used when switching the fixed value.</td></tr>
<tr><td>Update History</td><td>V2.0</td></tr>
</table>

4.6.2. db_0115: Simulink patterns for case constructs

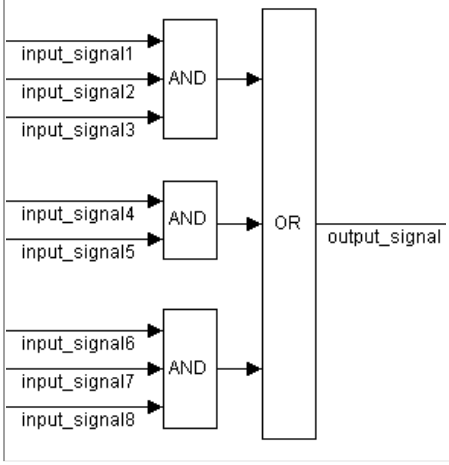
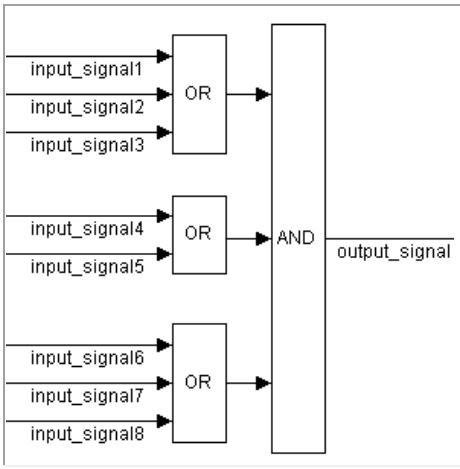
ID: Title	**db_0115: Simulink patterns for case constructs**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The following patterns are used for case constructs:

Function	Simulink pattern
Case constructs using Switch Case Action Subsystem switch (PRNDL_Enum) { case 1 TqEstimate = ParkV; break; case 2 TqEstimae = RevV; break; default TqEstimate = NeutralV; break; }	
Case construct using Multiport Switch block 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; }	

Update History	V4.0
Change classifications	Matched the Multiport Switch block example to the latest version code generation function and replace Deleted unneeded examples

4.6.3. db_0116: Simulink patterns for logical constructs with logical blocks

ID: Title	**db_0116: Simulink patterns for logical constructs with logical blocks**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Use the following patterns for logical constructs:

Function	Simulink pattern
Conjunctive normal form	
Disjunctive normal form	

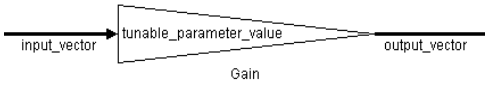
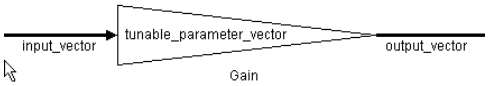
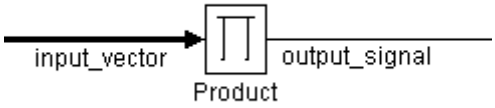
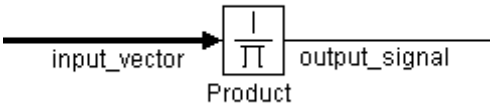
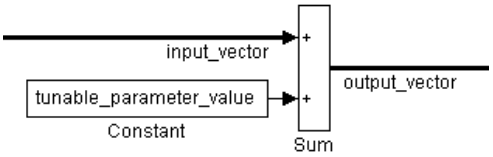
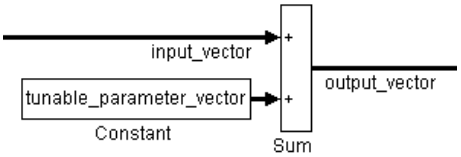
Update History	V1.0

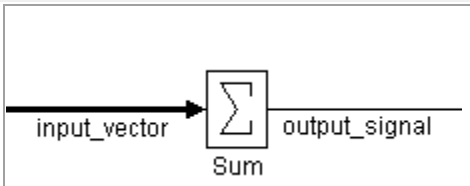
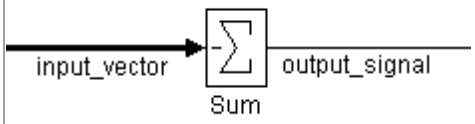
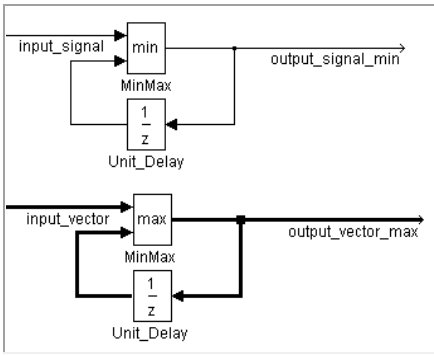
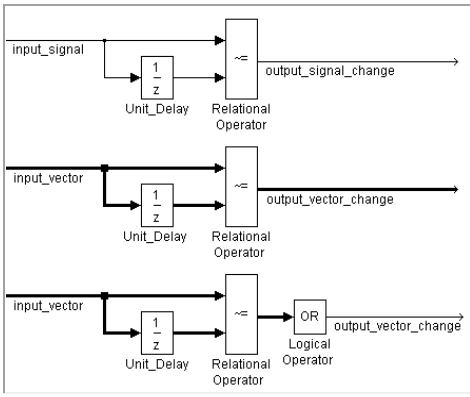
4.6.4. db_0117: Simulank patterns for vector signals

ID: Title	**db_0117: Simulank patterns for vector signals**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	

Description

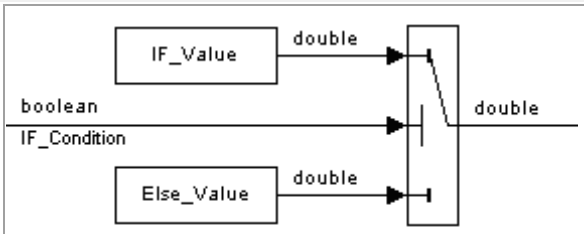
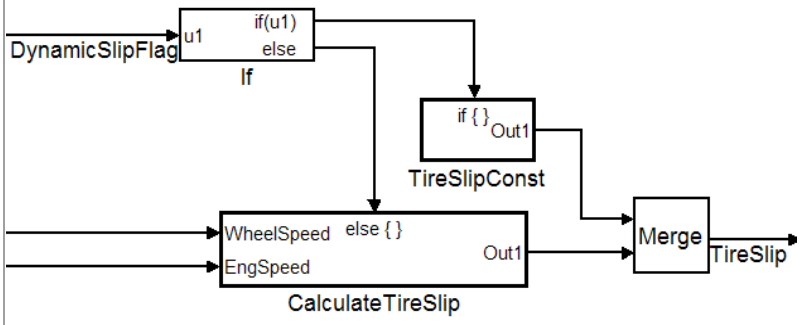
The following patterns are used for vector signals:

Functionality	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) for (i = 0; i < input_vectorDim; i++) { output_vector[i] = tunable_parameter_value * input_vector[i]; }	
Vector signal and parameter (vector) multiplication for (i=0; i>input_vector_size; i++) { output_vector[i] = input_vector[i] * tunable_parameter_vector[i]; }	
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; }	
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 addition output_signal = 0; for (i=0; i>input_vector_size; i++) { output_signal = output_signal + input_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	
	Edge detection	
Update History	V1.0	

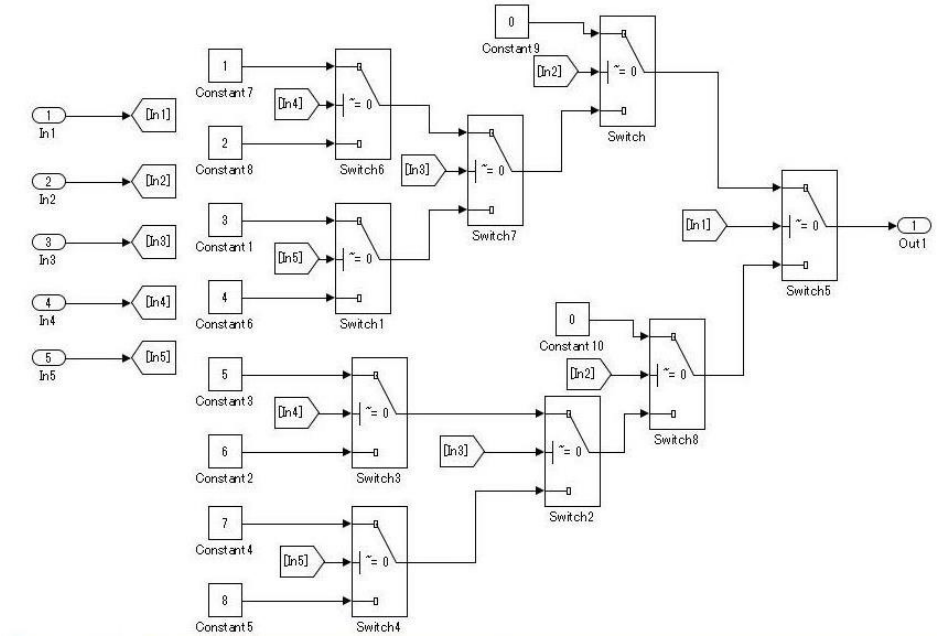
4.6.5. na_0012: Use of Switch vs. If-Then-Else Action Subsystem

ID: Title	**na_0012: Use of Switch vs. If-Then-Else Action Subsystem**
Priority	Strongly Recommended
Scope	NAMAAB
MATLAB Version	All
Prerequisites	
Description	The Switch block should be used for modeling simple if-then-else structures, if the associated then and else actions involve only the assignment of constant values.

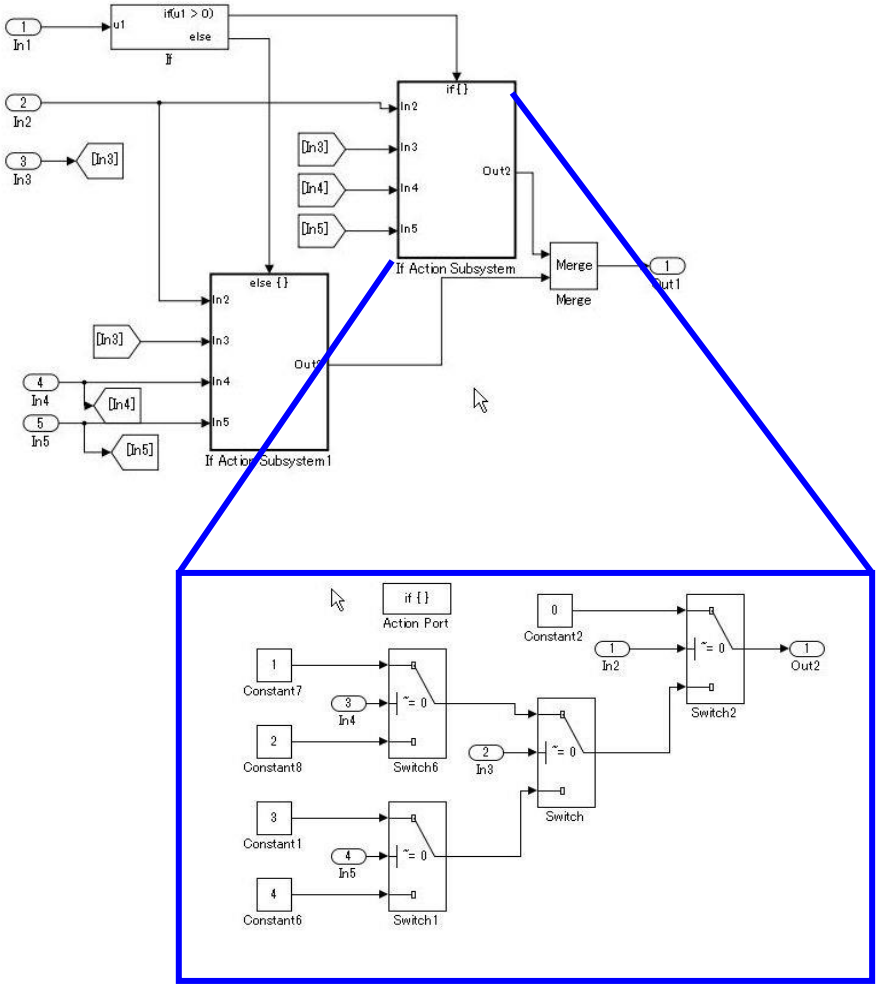
	 The if-then-else Action Subsystem should be used in the following cases: ● If the associated then action or else action requires complex calculations, use the if-then-else construct from within the conditional control flow for modeling. By doing so, not only the simulation efficiency, but also the generated code efficiency, will improve to the maximum limit (in basic blocks such as Table Lookup, pay attention to cases where quite complex calculations are required). 
Last Change	V4.0

4.6.6. na_0028: Use of If-Then-Else Action Subsystem to replace multiple switches

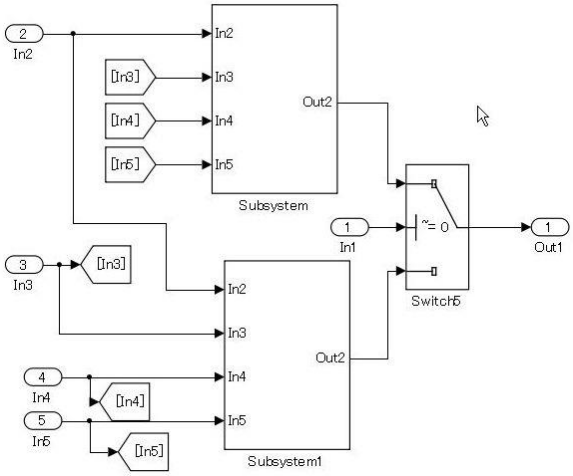
ID: Title	**na_0028: Use of If-Then-Else Action Subsystem to replace multiple switches**
Priority	Recommended
Scope	NAMAAB
MATLAB Version	All
Prerequisites	na_0012: Use of Switch vs. If-Then-Else Action Subsystem db_0114: Simulink patterns for If-then-else-if constructs
Description	Frequent use of the condition bifurcation by Switch block should be avoided. It should be operated based on a set upper limit target value. (For example, up to 3 levels) If the target value is exceeded, in its place a conditional control flow using the If-Then-else Action Subsystem can be listed. Incorrect: Nest is in 4 levels



Correct: With if-action Subsystem in 4th level, nest is limited to within a single level.

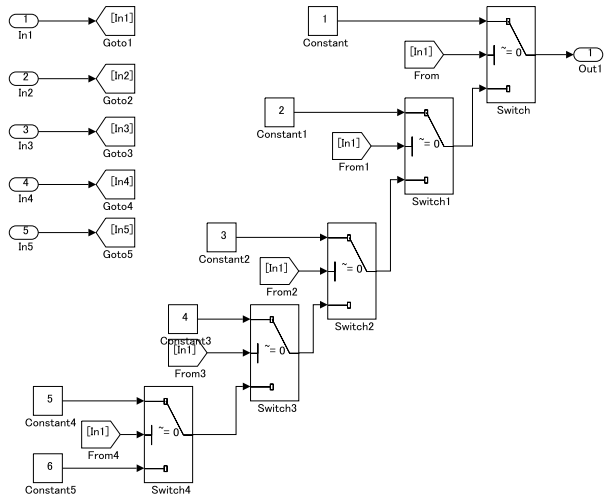


Incorrect: Not divided in if-action form.

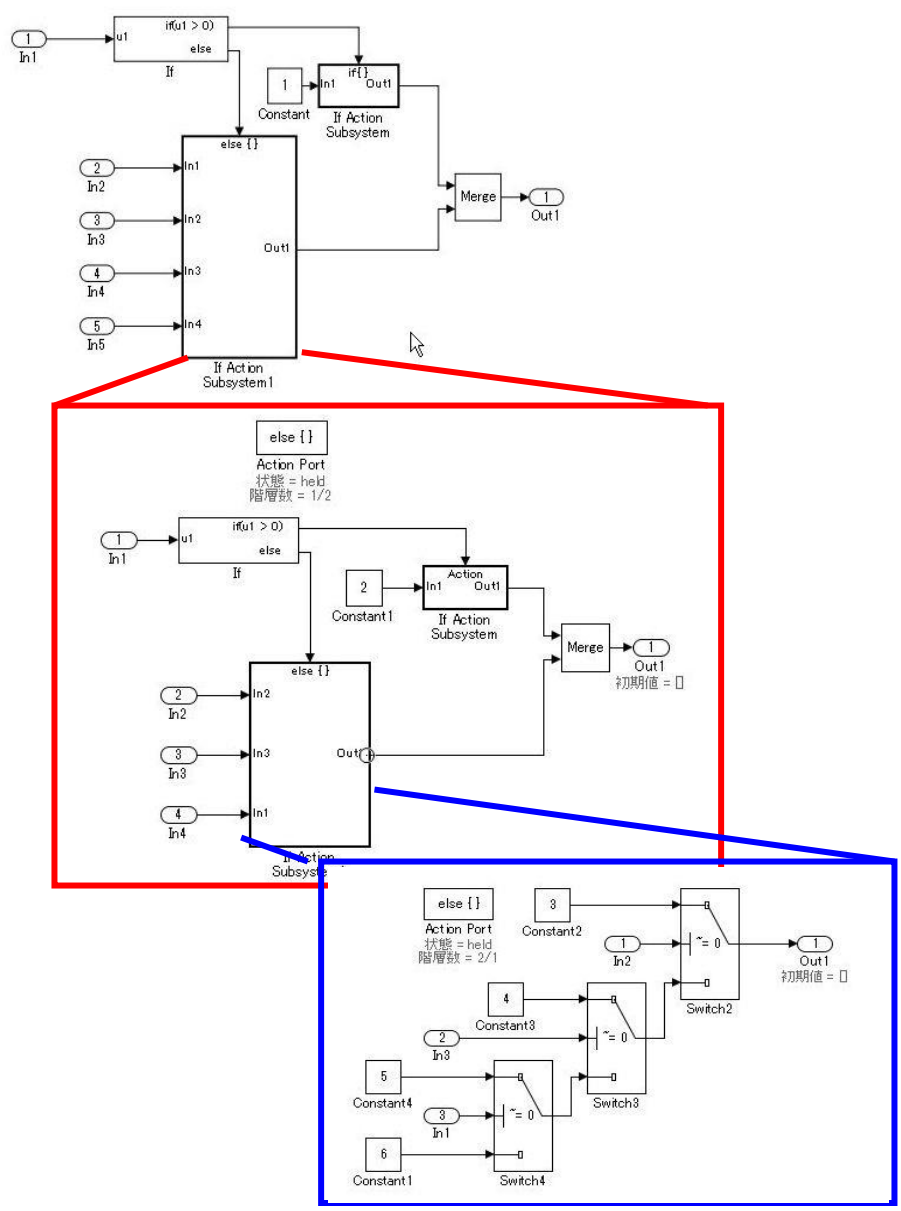


In the cases where the C code limit is reflected, it can be split into Atomic Subsystem + Function Setting. In this case, there is no need to use the if-then-else Action Subsystem, but the Switch block configuration can be split partway through, and merely encapsulated in the subsystem.

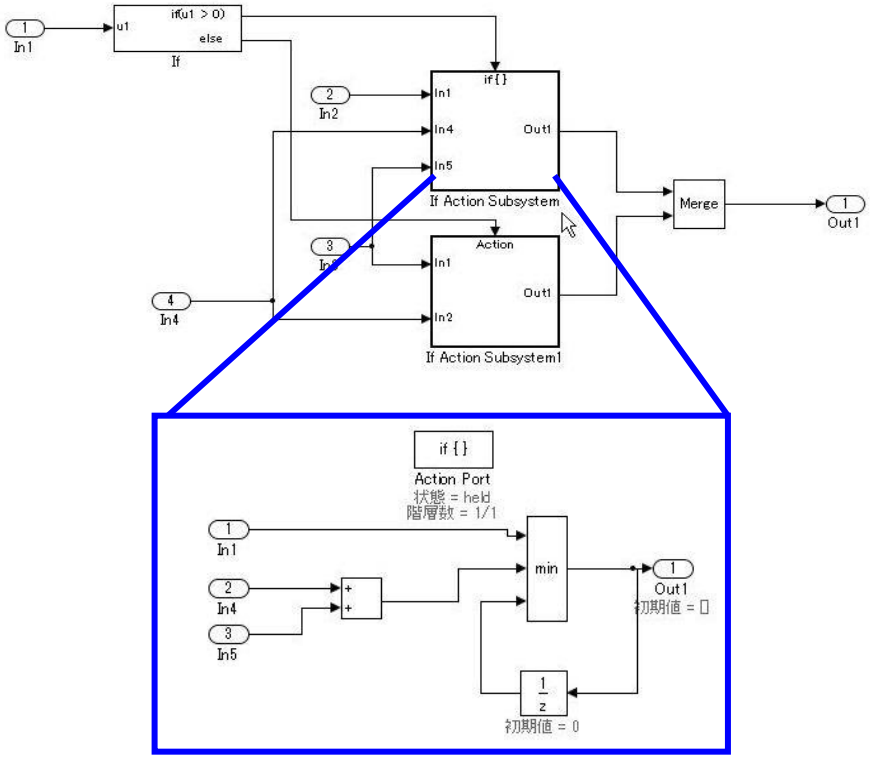
Example of model with 5-level nest
Incorrect:

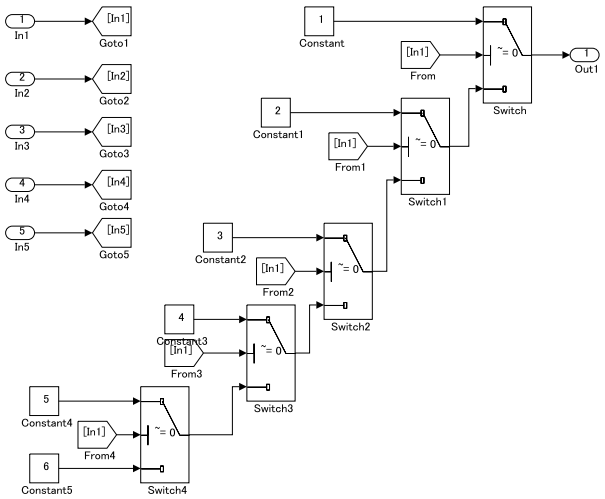


Correct: Description method that avoids layering of Switch nest

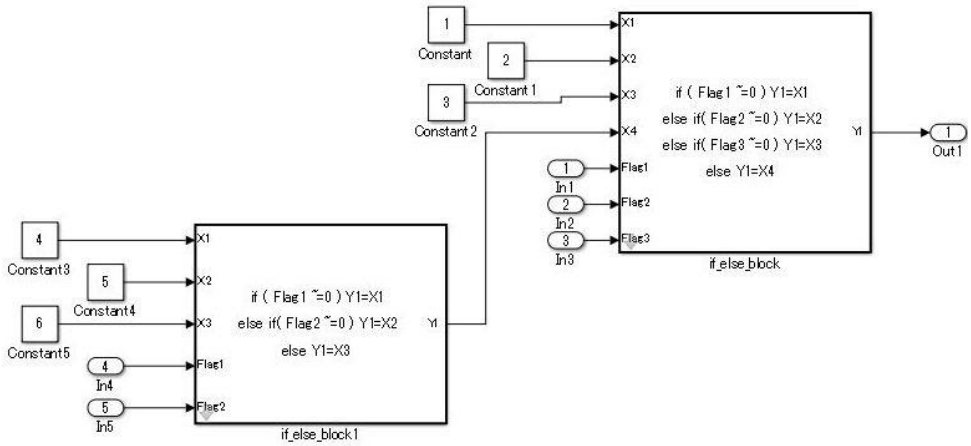
	
Notes	While listed as an example explanation, If Action Subsystem is normally not used in switching the fixed value In both the Correct and Incorrect above, if the user does not add a function conversion setting, the generated C code is the same. (Confirmed in R2010b to R2013a) This rule is not a constraint in the C code.
See Also	Orion_bn_0003: In place of multiple Switch, use the If-Then-Else Action Subsystem
Last Change	V3.0

4.6.7. jc_0658 : Usage rules for Action Subsystem using conditional control flow

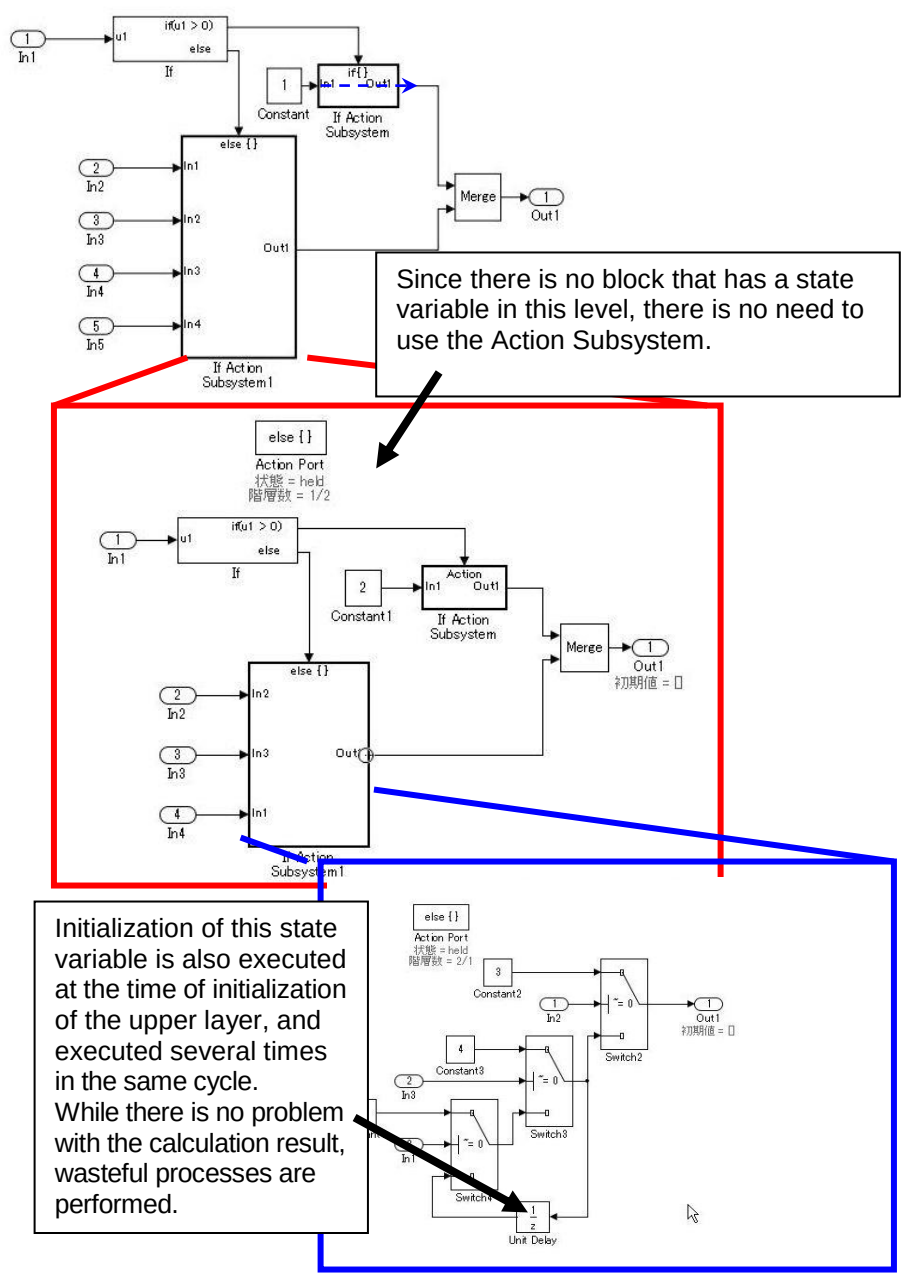
ID: Title	**jc_0658: Usage rules for Action Subsystem using conditional control flow**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	na_0012: **Use of Switch vs. If-Then-Else Action Subsystem**
Description	If the associated all actions do not have a status variable, the If-Then-Else Action Subsystem (Conditional Subsystem) should not be used. This rule adds strict limits to na_0012. No status variable: Use the Switch, Multiport Switch, and Index Vector. With status variable: The If-Then-Else Action Subsystem is usable as necessary. However, if the Action Subsystem exists in a layered lower layer, and if the status variable exists only in the lower Action Subsystem, the upper layer Action Subsystem is not used. For cases where a certain number of blocks or more are included in the related then Action and else Action rather than the Action Subsystem, use and list the normal subsystem and block that have a switching function (Switch, Multiport Switch, Index Vector). (Define and use the upper limit value for number of blocks.) Correct:  Example of model with 5-level nest Correct: Since there is no internal state, layering using the subsystem is not performed.



Correct:
The Atomic Subsystem is used to split either side of the Switch without using Action Subsystem.



Incorrect:
Layering using an unnecessary Action Subsystem is performed.



If a function can be achieved even without using the Action Subsystem, then layering using the Action Subsystem is not performed.

In the Incorrect example, when the lowest level UnitDelay existing on the third level is initialized, first, the conditional subsystem initialization is executed one time on the upper first level, and then the conditional subsystem is initialized on the second level for a total of two times of initial value settings. In order not to generate unnecessary code, in levels where the state variable does not exist, it is recommended that no listing be made in conditional subsystems.

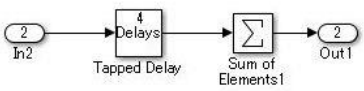
In addition, this rule does not coexist with na_0028, and becomes a selective expression rule.
Select and use either one within the model.
na_0028 is based on the concept that the model (not the code) complexity is reduced by dropping to a level. This rule is a rule for the purpose of avoiding execution of unnecessary initializations.

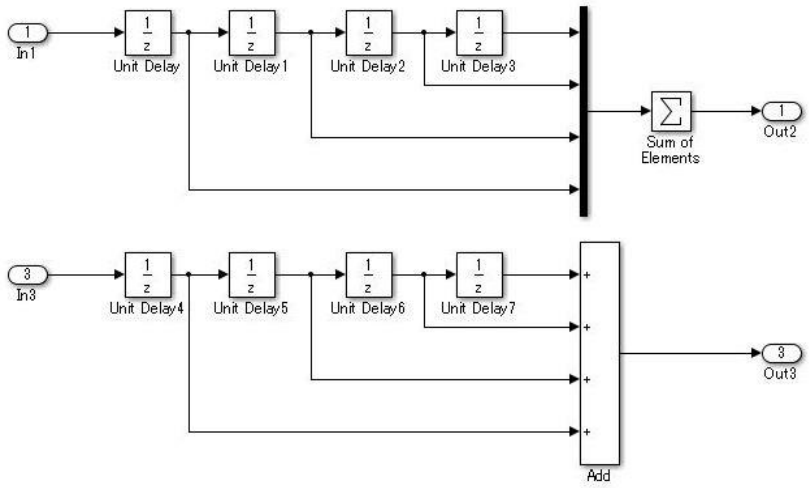
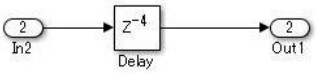
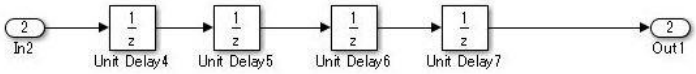
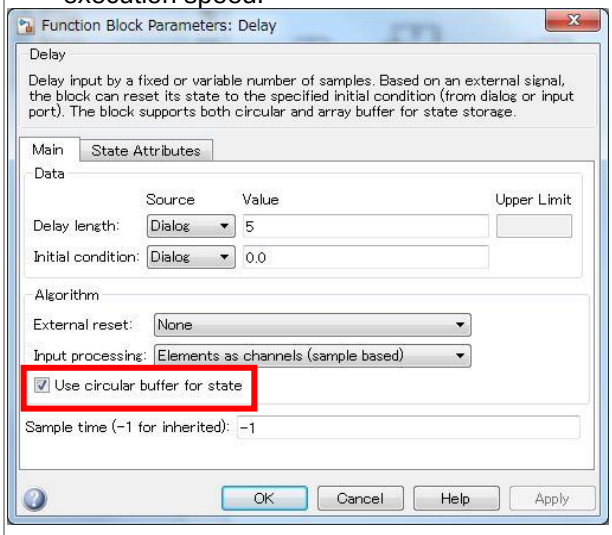
Notes	While unrelated to the regulations in this rule, the bifurcation of systems where the bifurcation condition nest has a deep structure is split by function conversions so as to lower the code bifurcation nest. For this purpose, functions before and after the Switch block are divided into respective subsystems, and function settings are performed for the Atomic Subsystem＋function. However, since there is a possibility that unintentional implementation could result in addition of unnecessary RAM, a check of trade-offs is required. Since both have their strengths and drawbacks, select a description method that matches the model.
See Also	
Last Change	V4.0

4.6.8. jc_0623: Use of Memory block vs. Unit Delay block

ID: Title	**jc_0623: Use of Memory block vs. Unit Delay block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	・The Memory block is not used within discrete type models or subsystems. (Use the Unit_Delay Block.) ・The Unit_Delay Block is not used within continuous type models or subsystems. (Use the Memory Block.)
See Also	
Last Change	V4.0

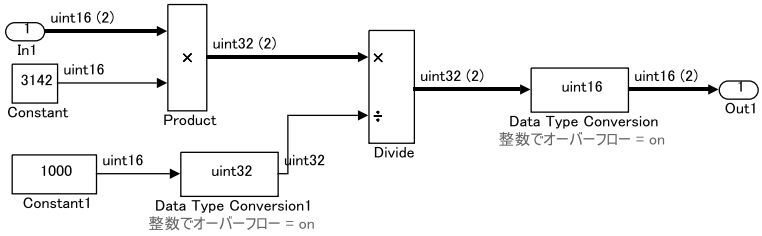
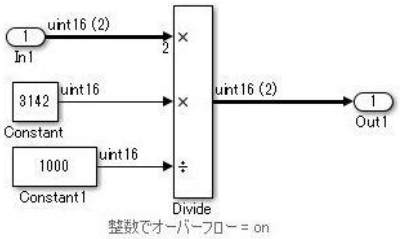
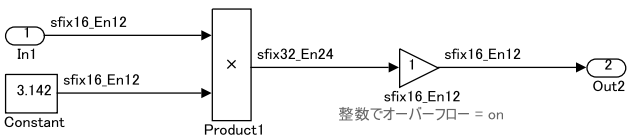
4.6.9. jc_0624: Guideline for using the Delay block

ID: Title	**jc_0624: Guideline for using the Delay block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	R2011b and later
Prerequisites	
Description	● If wanting to obtain a vector signal that includes past values, rather than lining up multiple Unit Delays, the Tapped Delay block should be used. ● If wanting to obtain the oldest value only, the Delay block should be used. Tapped Delay block example Correct:  Incorrect:

	 Delay block example Correct:  Incorrect: 
Notes	Supplement ● The Tapped Delay and Delay blocks are set with arrays holding past values, and have improved code visibility to assist code efficiency. ● If the number of delays is frequent (for example, five or more), using the Delay block means performing settings for use of code using a cycling buffer, which can assist execution speed. 
See Also	
Last Change	V4.0

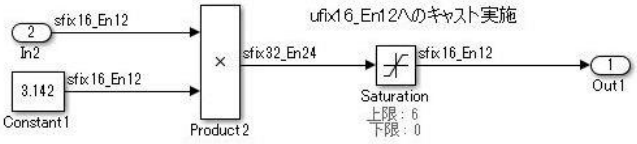
4.6.10. jc_0651: Guideline for use when implementing cast

ID: Title	**jc_0651: Guideline for use when implementing cast**

Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	**jc_0628: Guideline for using the Saturation block**
Description	If implementing down cast, it should be split with operations (addition, subtraction, multiplication, division) for other purposes. This is virtually the same purpose as clearly listing parentheses, and clarifying the execution order. Dividing the operations and cast can help to clarify the order of execution and up to which operation should use which data type in the block structure. Blocks implementing down cast consist of the following three types of blocks: 1. Data Type Conversion 2. Gain: However, value is 1 3. Saturation If there is not otherwise a particular reason, use Data Type Conversion. Gain block is an alternative block that is often used when Data Type Conversion cannot be used due to tool constraints. Saturation is used when implementing a saturation process and down cast in a single block. However, use of Saturation is not desirable when the saturation process is used for purposes of overflow prevention. If using something other than Data Type Conversion, use block names or annotations to add comments for clarifying that it is a cast. Correct: Example of using Data Type Conversion  Perform cast, unify the internal data type, and clearly show the calculation order. Incorrect:  All cast processes is consigned to auto code. Correct: Example 1: using other than Data Type Conversion 

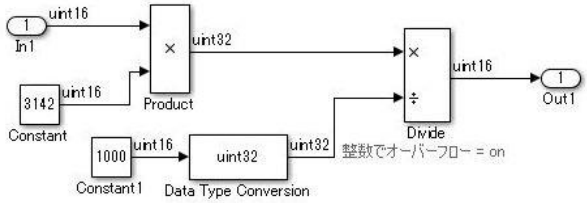
Shows that it is a cast in the Gain block name.

Correct:
Example 2: using other than Data Type Conversion



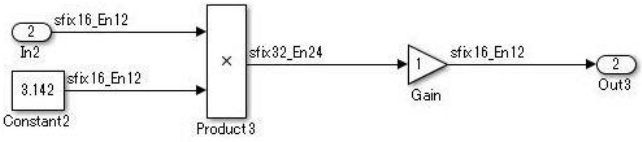
Change to a value smaller than the type constraints, and implement cast.
Use comments near Saturation to clarify the cast implementation.

Incorrect:



Since operations and cast are processed in the same block, the precision of calculations in progress cannot be confirmed. In this case, it can not see the accuracy of the calculation during during division. (Simulink automatically changes to 32bit operation. And after operation, it is restored to 16bit. Although it relies on Simulink function, It is better to set data type explicitly.)

Incorrect:



While there is an exclusive Gain block for cast, it is not clearly understood that its purpose is cast.

Notes

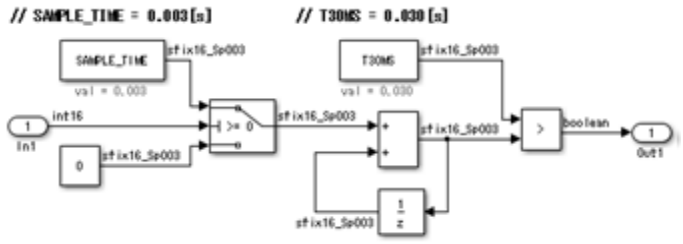
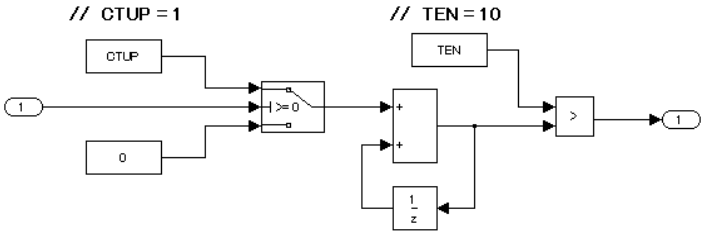
Although Correct and Incorrect both depend on the configuration setting, since virtually the same code can be obtained,
the purpose is not for the generation code, but for becoming able to confirm the process in the model.

Block usage pattern

<table>
<tr><th>Usage pattern</th><th>Saturation process
Implementation block</th><th>Down cast
Implementation block</th></tr>
<tr><td rowspan="2">Type conversion only
(No saturation process)</td><td>—</td><td>DataTypeConversion
(No overflow saturation)</td></tr>
<tr><td>—</td><td>Gain: 1
(No overflow saturation)</td></tr>
<tr><td rowspan="2">Type conversion after saturation process
for purposes of overflow prevention</td><td colspan="2">DataTypeConversion
(With overflow saturation)</td></tr>
<tr><td colspan="2">Gain: 1
(With overflow saturation)</td></tr>
<tr><td rowspan="2">Not for purposes of overflow prevention
(with significance)
Type conversion after saturation process</td><td colspan="2">Saturation/Dynamic Saturation
(Type conversion in output type setting)</td></tr>
<tr><td>Saturation/
Dynamic Saturation</td><td>DataTypeConversion
(No overflow saturation)</td></tr>
</table>

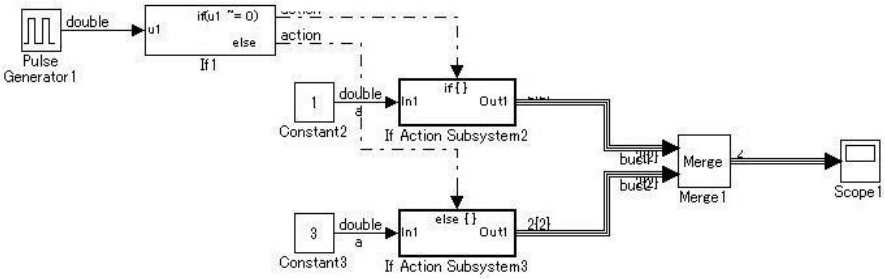
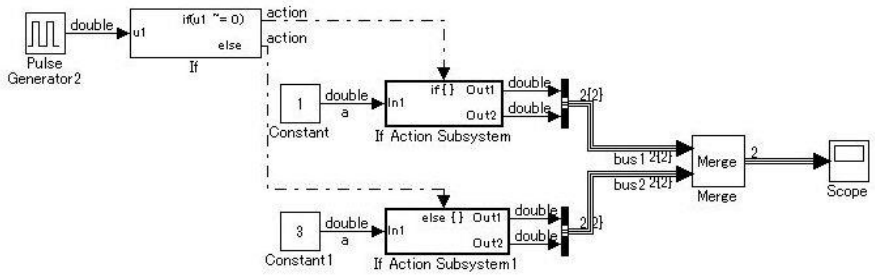
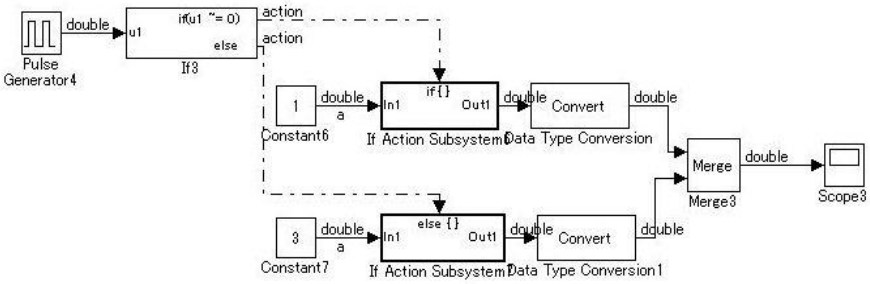
			Gain: 1 (No overflow saturation)
See Also	MISRA SLSF0002A		
Last Change	V4.0		

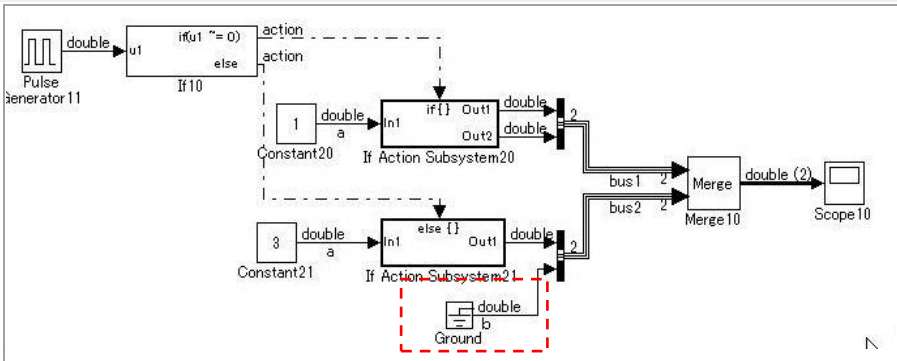
4.6.11. jc_0652: Constant related to timer counter

ID: Title	**jc_0652: Constant related to timer counter**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	The constant related to the Timer Counter is not expressed by the number of occurreces but by that specific time. If the number does not match the time, use a comment to list the time unit. Can also list in ＜Block Property＞＜Explanation＞ or mpt.Parameter description, and display by using block annotation. Use a consistent method, and insert descriptions of the time unit also for the listing content. [List the time units that have been determined, such as second(s), milliseconds (msec), etc.] Correct: Constant is expressed in time  **Incorrect:** Constant is not expressed in time 
Notes	If real number is used, total value may not be equal to real counting time. In that case, set in consideration of the tolerance.
See Also	
Last Change	V4.0

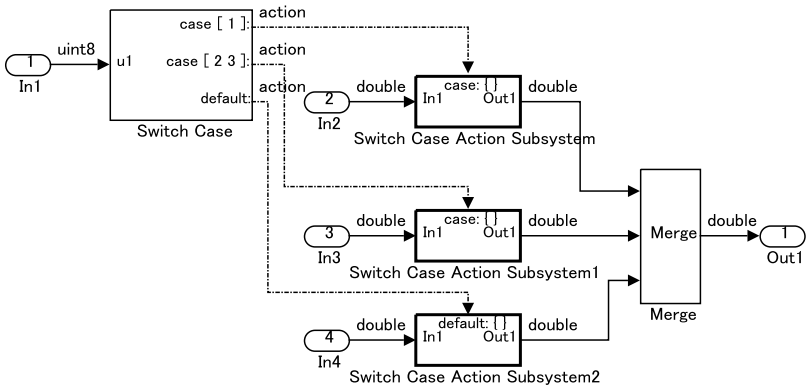
4.6.12. jc_0659: Usage restrictions of signal lines inputted to Merge block

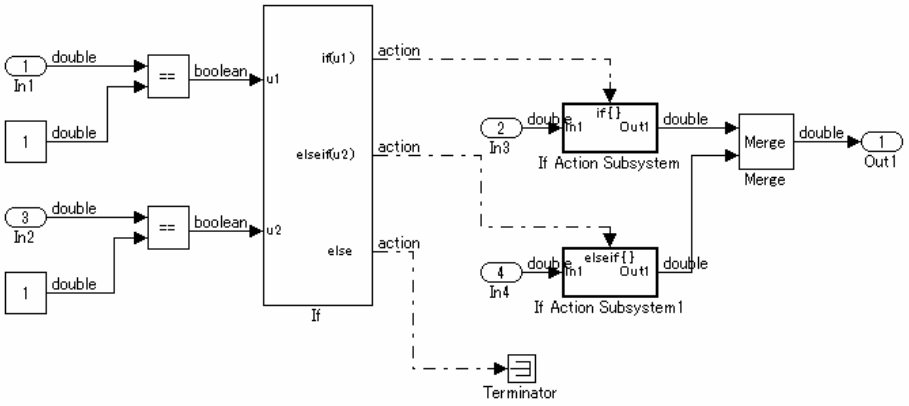
ID: Title	**jc_0659: Usage restrictions of signal lines inputted to Merge block**
Priority	Strongly Recommended
Scope	JMAAB

MATLAB Version	All
Prerequisites	
Description	No blocks should be positioned between the Conditional Subsystem and Merge block. Correct:  Incorrect: 
Notes	Analysis: A virtual block can be inserted between the Conditional Subsystem and Merge block. Reference: Virtual block http: //www.mathworks.co.jp/jp/help/simulink/ug/about-blocks.html The above subsystem can also output normal results. However, if the above is allowed, there is a possibility of inducing two mistakes. This is because it is difficult to understand all of the virtual block types shown in the above Help, and to use the correct combinations. For example, the Example 1 DataTypeConversion block is not a virtual block. This will not operate correctly. Example 1  This uses the checker 'mathworks.design.MergeBlkUsage', to detect this case as a violation. In the next Example 2, the configuration is virtually the same as the Incorrect example, and the input signal to the Bus Creator block is connected from the Ground. Example 2

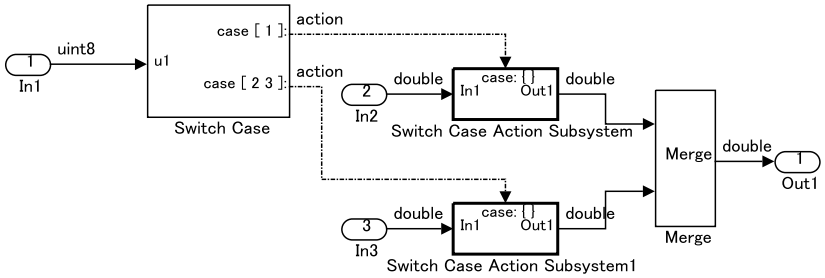
	 In this description, since the Ground block is always active, the bus name b signal always has an output value of 0, and correct results cannot be obtained. The checker 'mathworks.design.MergeBlkUsage' recognizes that this case is correct. If a model in the Incorrect example is created, and then converted to the Example 2 type in accordance with later changes in specifications, it will result in unintentional actions. The intention of this rule is prevention of these mistakes beforehand.
See Also	
Last Change	V4.0

4.6.13. jc_0656: Guideline for using the Conditional Control block

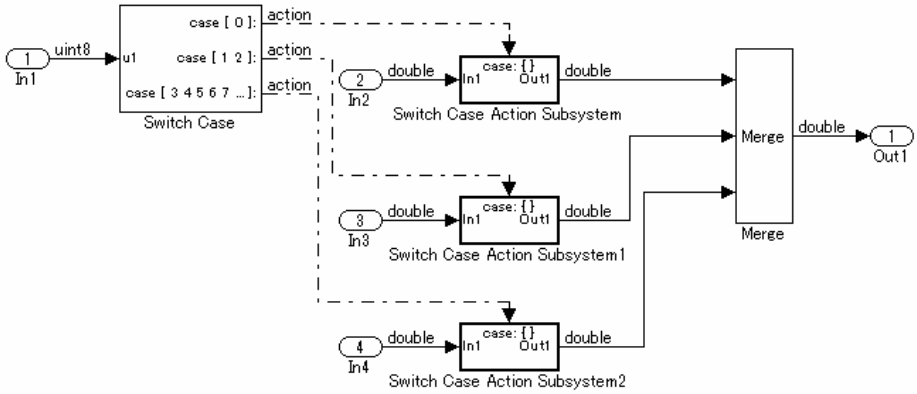
ID: Title	**jc_0656: Guideline for using the Conditional Control block**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	In the Conditional Control Flow block (if block, Switch Case block), use the settings below to make all actions in the conditions explicit. ● For the if block, set "else condition display" to On for use. ● For the Switch Case block, set "display default case" to On for use. Correct: Modeling when a function showing the default action exists  Correct: Modeling when a function showing the default action does not exist



Incorrect:
Default port does not exist.



Incorrect:
Default port is not used and all values of used data type are defined.

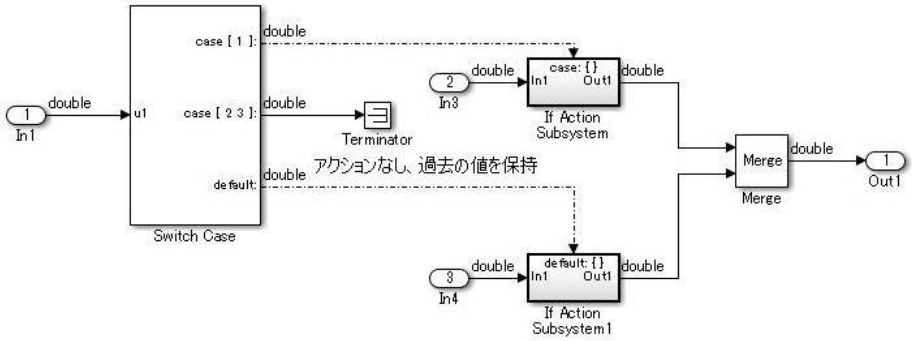
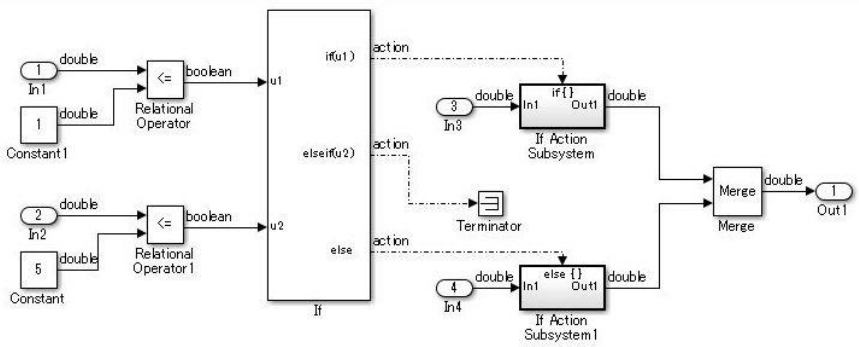
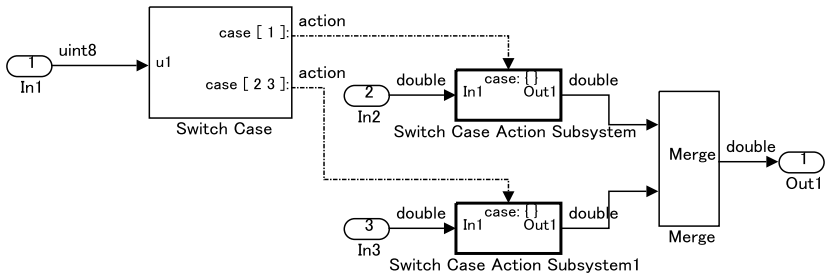


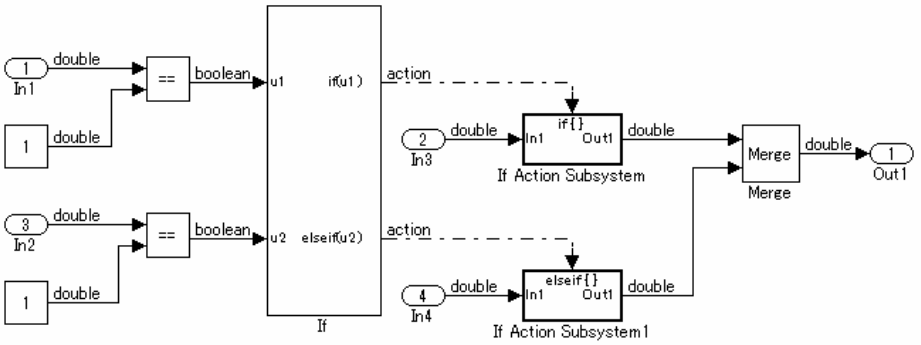
As seen in this model, even if conditions are set on the full range of input signal types, if data type of input signal is changed, undefined range can exist. This description does not mean clarification has been made for all conditions.

See Also	hisl_0010: Guideline for using the if block and Action Subsystem block hisl_0011: Guideline for using the if block and Action Subsystem block MISRA AC SLSF 011B
Last Change	V4.0

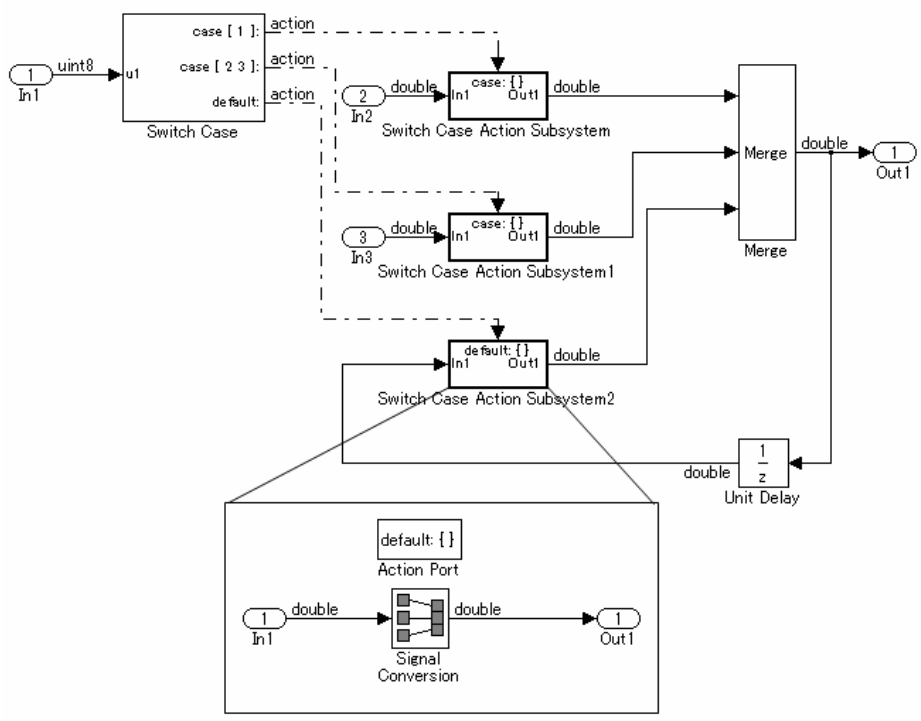
4.6.14. jc_0657: Retention of output value based on Conditional Control Flow block and Merge block

ID: Title	**jc_0657: Retention of output value based on Conditional Control Flow block and Merge block**

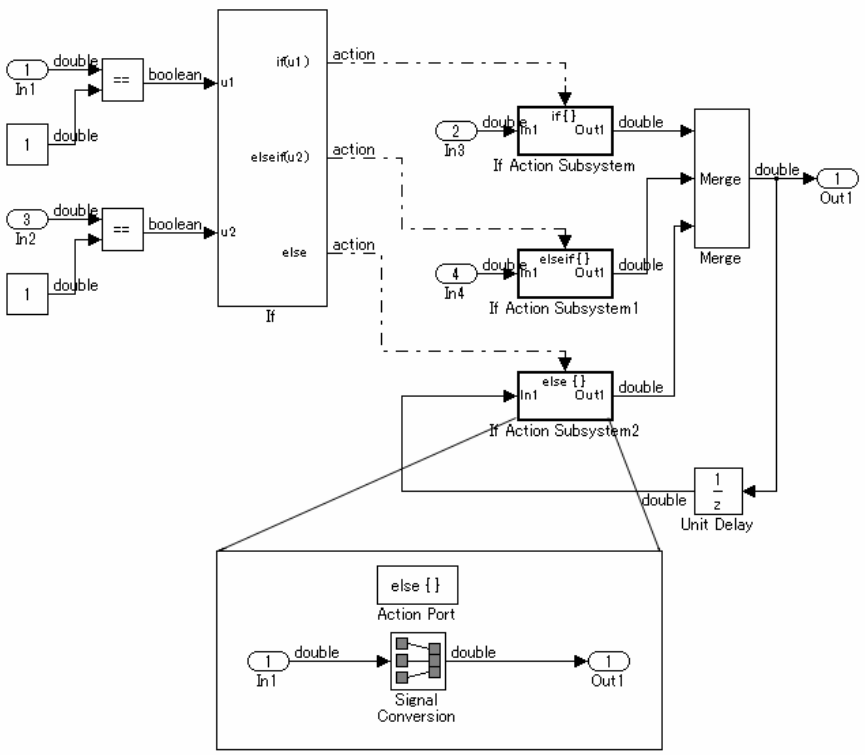
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	If using the Conditional Control block (if block, Switch Case block) to switch the executed function, and using the Merge block to select the results, and if, depending on the conditions, retaining past values only, connection of the condition ports to the Terminator block will clarify retention of past values. (This is different to setting default port) Correct: Switch-case example  Correct: if-else example  If performing automatic code generation, a highly efficient code is outputted without taking up excess RAM. This means that, if past values are retained even in other than default (else), connections to the Terminator block can be used. Incorrect: 



While the automatic code generation results are the same as above, it is not clear whether actions outside of conditions are OK with retention of past values.
Incorrect:



Incorrect:

<table>
<tr><td></td><td> 

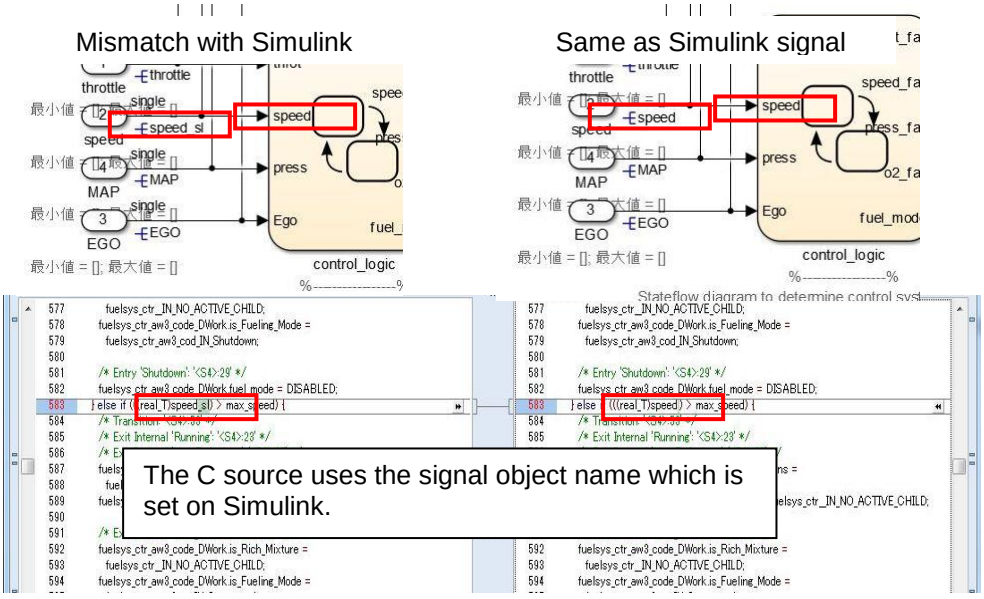
While the actions are clear, design of excessive subsystems is necessary for retaining past values, and in some cases the code efficiency deteriorates because of verbose RAM allocation.</td></tr>
<tr><td>Notes</td><td>It is better to describe comments around Terminator blocks in order to clearly show that it is the block structure to retain past values.</td></tr>
<tr><td>See Also</td><td>hisl_0010: Guideline for using the if block and Action Subsystem block
hisl_0011: Guideline for using the Switch Case block and Action Subsystem block
hisl_0015: Guideline for using the Merge block
MISRA AC SLSF 011B</td></tr>
<tr><td>Last Change</td><td>V4.0</td></tr>
</table>

5. Stateflow

Explanation of the Stateflow® chart appearance, data and operation, event, state chart pattern, and flowchart pattern guidelines

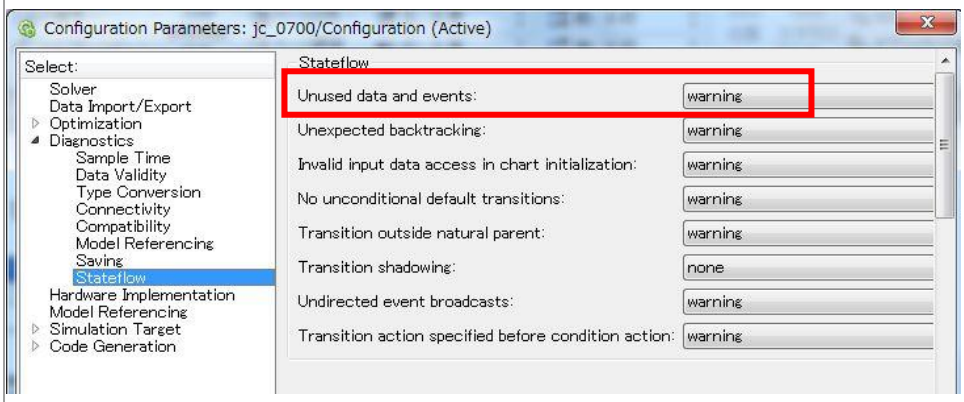
5.1. Stateflow variable settings

5.1.1. db_0123: Stateflow port names

ID: Title	**db_0123: Stateflow port names**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The name of a Stateflow block input/output should be the same as the corresponding signal. Exception: Stateflow blocks performing reusable function settings may have different port names. Stateflow blocks include the Chart block, MATLAB block, and Truth Table, etc. If adopting jc_0602, this rule is included within it. If jc_0602 is not adopted, use this rule for the Stateflow description only.
Notes	This rule is not a rule for C code generation.(if data objects are used.) This rule is for improving model readability. 
See Also	MISRA AC SLSF 036-C
Last Change	V1.0

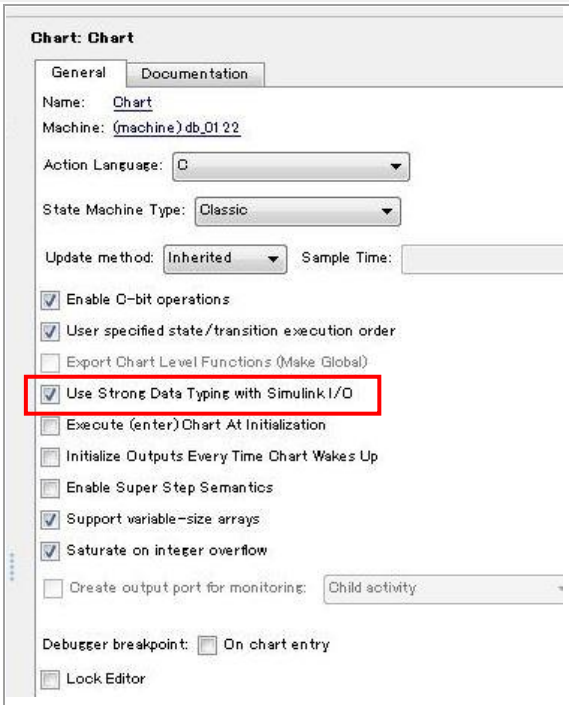
5.1.2. jc_0700: Unused data in Stateflow block

ID: Title	**jc_0700: Unused data in Stateflow block**
Priority	Strongly Recommended

Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Unused data and events should not exist in the Stateflow block. In R2010b and later, set the configuration parameter diagnosis ＞Stateflow＞"Unused data and events" to other than "None". 
See Also	MISRA AC SLSF 037G
Last Change	V4.0

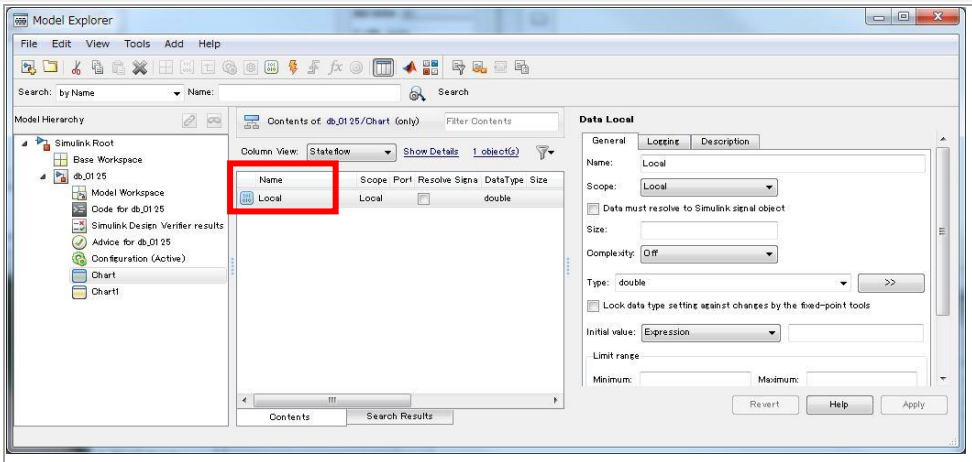
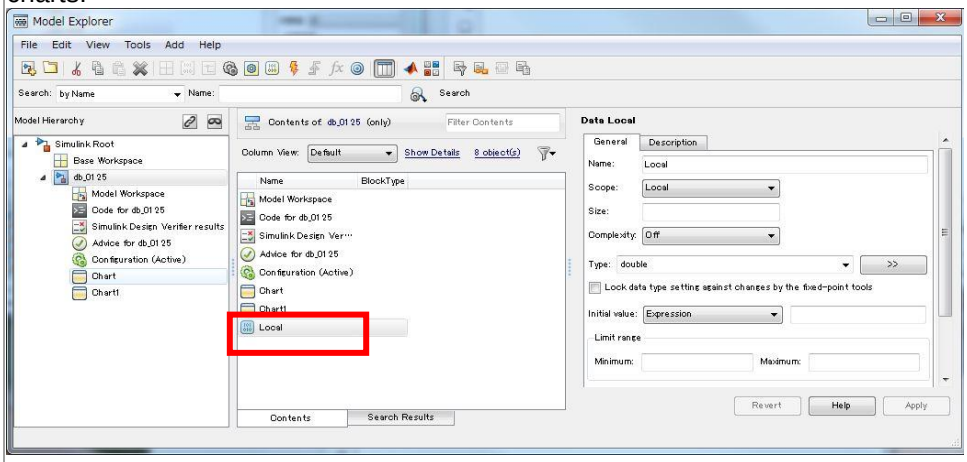
5.1.3. db_0122: Stateflow and Simulink interface signals and parameters

ID: Title	**db_0122: Stateflow and Simulink interface signals and parameters**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Use the Simulink and Stateflow types as equivalent. Select File > Chart Property > “Strict type specification in Simulink I/O".

	
Notes	Attention: This option is going to be deleted on future version. Property name difference of versions. Up to R2008b, "Retain data type in Simulink and I/O" From R2009a, "Strict type specification in Simulink I/O" If “Use Strong Data Typing with Simulink I/O” is deactivated and Simulink data type is not double, Stateflow cannot be executed.
Last Change	V2.0

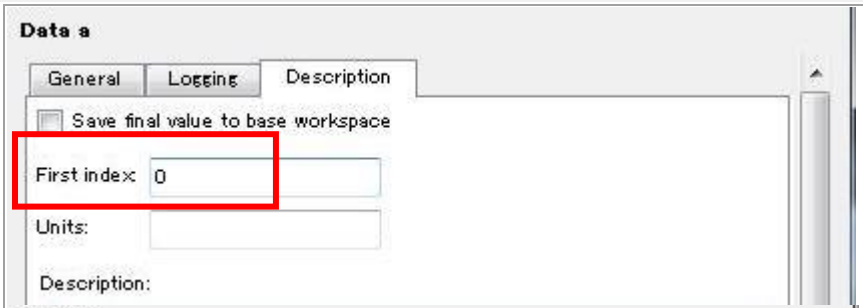
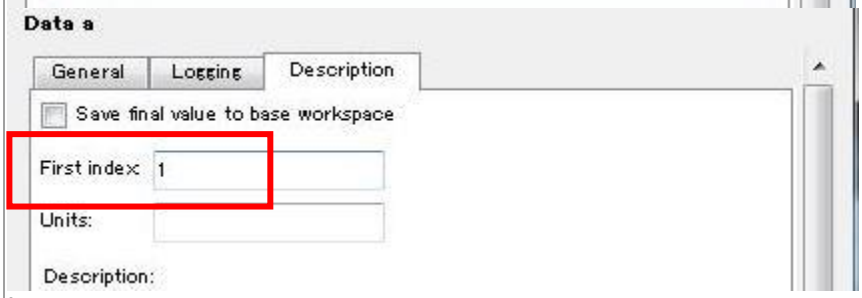
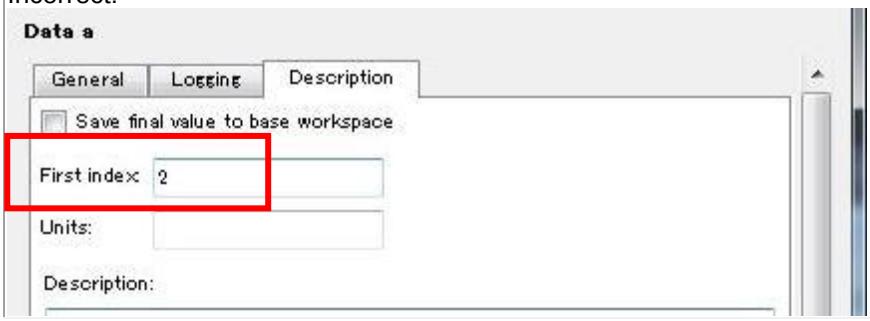
5.1.4. db_0125: Scope of internal signals and local auxiliary variables

ID: Title	**db_0125:** Scope of internal signals and local auxiliary variables
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Variables only used inside the Stateflow Chart must satisfy the following conditions: ● All local data of Stateflow block must be defined on the Chart level or below the Object Hierarchy. ● No local variables exist on the machine level. (That is, there is no interaction between local data in different charts). ● Parameters and constants are allowed at the machine level. ● Local data having the same name should not be included within the charts/states with parent-child relationships. **Correct:** Local variable is defined under Chart

	 **Incorrect:** Local variable is defined on machine level on which signals can be shared among several charts. 
See Also	MISRA AC SLSF 037 B
Last Change	V4.0

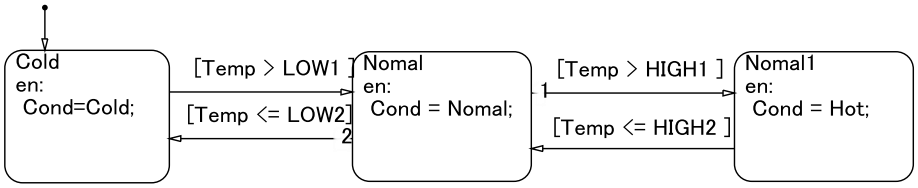
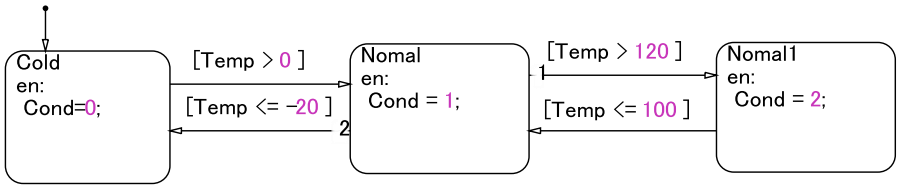
5.1.5. jc_0701: Usable numbers in first index

ID: Title	**jc_0701: Settable numbers in first index**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Set the first index of arrays used in Stateflow to "0" or "1". Caution: Since the first index if not specified is handled as "0", there is no need for designation unless specifically required. Correct:

	  Incorrect: 
See Also	
Last Change	V4.0

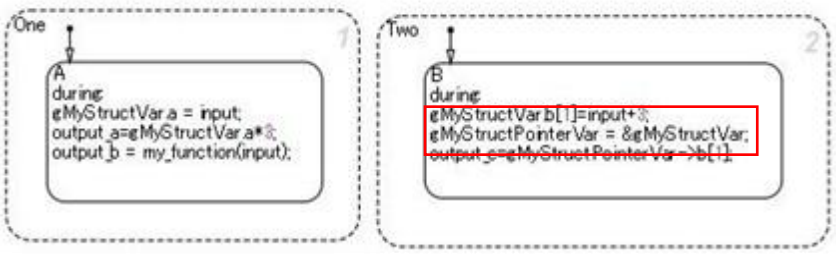
5.1.6. jc_0702: Stateflow parameters and constants

ID: Title	**jc_0702: Stateflow parameters and constants**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	・Parameters and constants within Stateflow should not directly use numbers. ・Labels should be used for the parameters and constants within Stateflow. Exceptions: ・"0" can be used as an initial value for variables. ・"1: can be used for variable increments and decrements. Usage examples for ordinary parameters Correct:

	 Incorrect: 
See Also	MISRA AC SLSF 048G、H
Last Change	V4.0

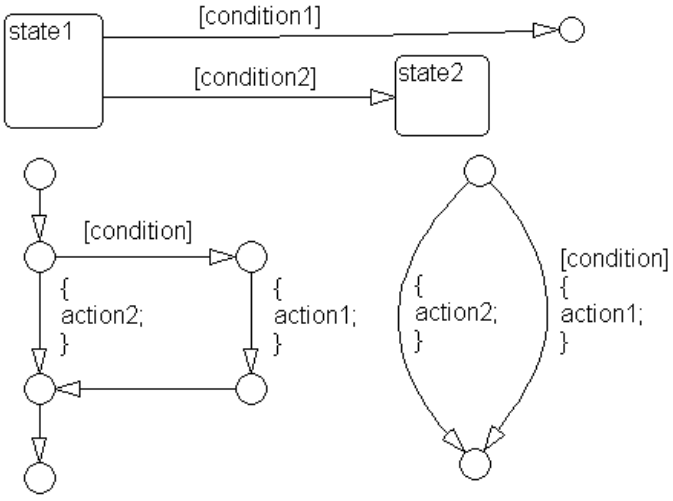
5.1.7. jm_0011: Pointers in Stateflow

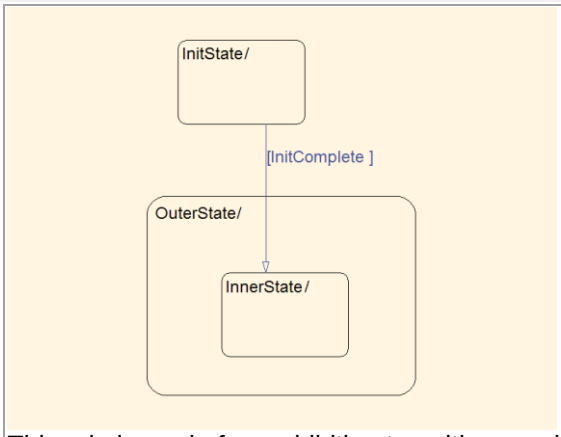
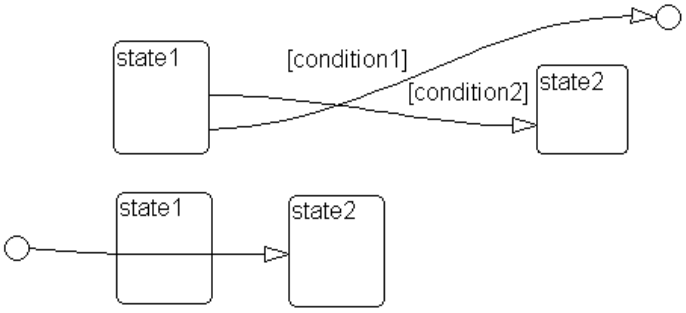
ID: Title	**jm_0011: Pointers in Stateflow**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	In the Stateflow diagram, pointers to custom code variables should not be used. Direct reference to pointer variable while accessing to device driver is inhibited. Incorrect: void *pointerToVar = (void *) 0x32344a3; Correct uint32 Var = Signal;
Notes	This rule is not a rule prohibiting use of pointers within the custom code. Within the custom code, pointers to variables within the custom code may be used for access. Direct reference from Stateflow to variables declared in C code is possible. • Variable declarations in custom C source code. MyStruct gMyStructVar; MyStruct *gMyStructPointerVar=NULL; • Description in Stateflow chart

	 Stateflow can directly refer to the signals of gMyStructVar and gMyStructPointerVar which are defined in in C source code. However, use of signals which is not defined in any model makes model difficult to understand. Although this rule doesn't limit, it is better to not use it.
Last Change	V1.0

5.2. Basic appearance of state transition

5.2.1. db_0129: Stateflow transition appearance

ID: Title	**db_0129: Stateflow transition appearance**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	・In Stateflow transitions, the following regulations are applied: ● Do not cross each other as much as possible. ● Do not draw upon the other. ● Do not cross any states, junctions or text fields. However, crossing with forced transition from external states to internal states is possible. ・For transition labels, set to show visual relationships with the corresponding transition. Correct:  Correct: Transition crosses state boundary to connect to substrate

	 This rule is a rule for prohibiting transition overlap, and does not prohibit state transitions from outside to center, or from center to outside. Incorrect: Transitions crosses each other and transition crosses through state. 
Last Change	V2.2

5.2.2. db_0137: States in state machines

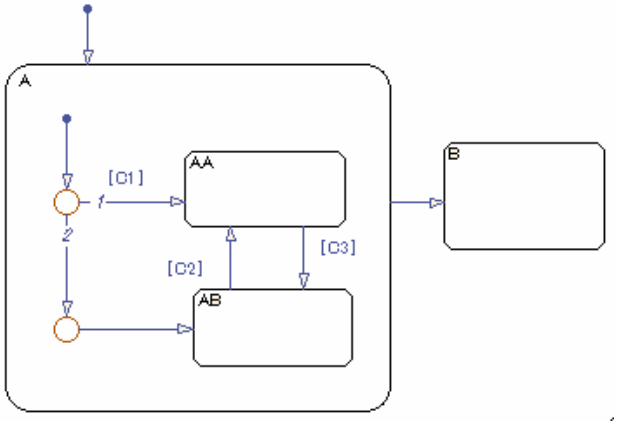
ID: Title	**db_0137: States in state machines**
Priority	Mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	jc_0743: Guideline for writing condition actions jc_0531: Placement of the default transitions
Description	In all levels in a state machine, including the root level, for states with exclusive decomposition, the following rules apply: ● In the same level, at least two exclusive states must exist. If parallel is selected, only one state can be established.
Notes	In the old description, jc_0531 was here: Only part of the default transition was listed, and it was deleted since there were duplicated roots.
Last Change	V4.0

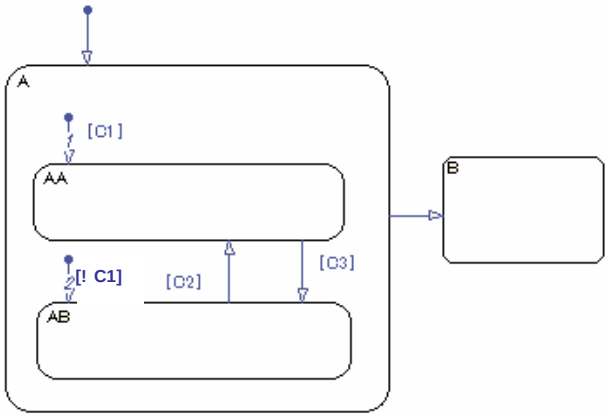
5.2.3. jc_0711: Division in Stateflow

ID: Title	**jc_0711: Division in Stateflow**
Priority	Recommended
Scope	JMAAB

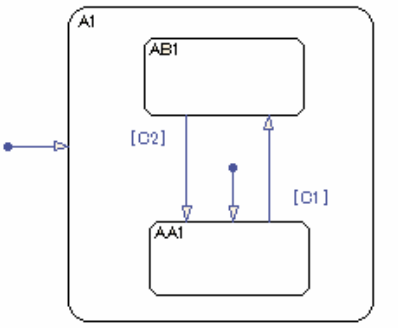
MATLAB Version	All
Prerequisites	
Description	If using division, the user must perform modeling of process for avoiding division by zero.
Notes	
See Also	MISRA AC SLSF 038B
Last Change	V4.0

5.2.4. jc_0531: Placement of the default transition

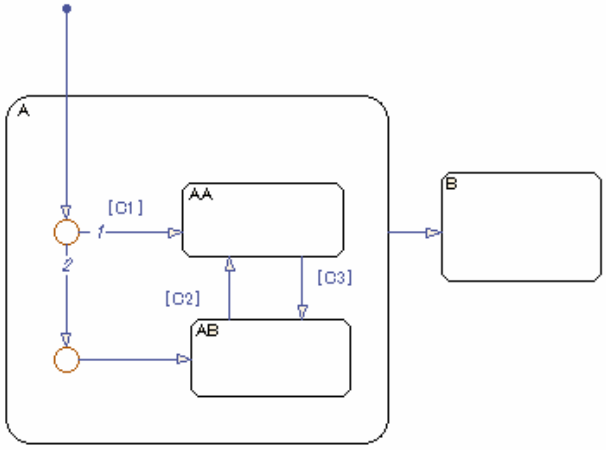
ID: Title	**jc_0531: Placement of the default transition**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	**db_0137: States in state machines**
Description	Default transitions should be drawn so that the following conditions are satisfied: ● If an exclusive (OR) and substate exist, the default transition is internally established. ● Multiple default transitions cannot be included in the same level. ● Default transitions are directly connected to the upper part of the state or junction. ● The transition destination state or transition destination junction for the default transition is positioned in the far upper left within the same level. ● The default transition must not exceed the state boundary. ● The default transition within a state chart must have a non-guard path to the state. Correct:  Incorrect: Multiple default transitions are included in the same level.



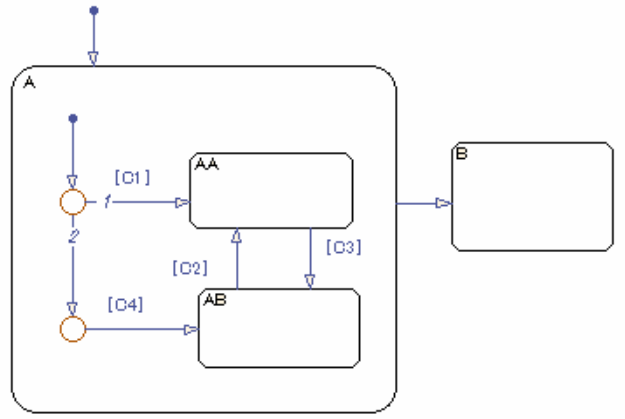
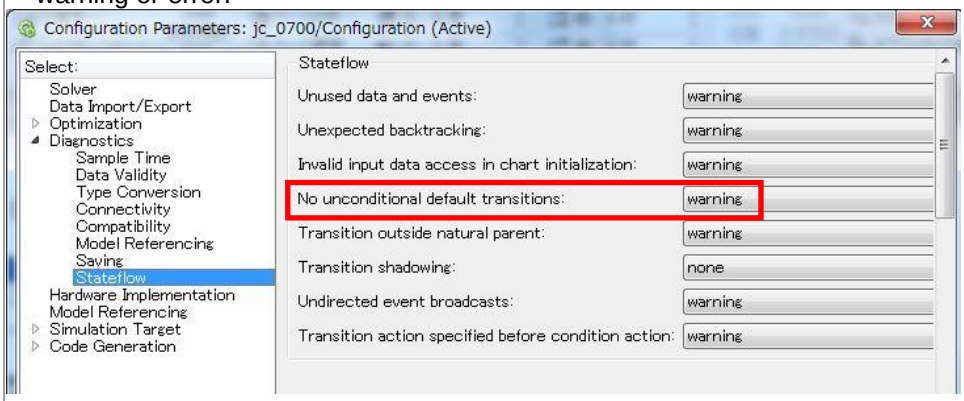
Incorrect:
The default transition is positioned in the side area.
The transition destination state of the default transition is not the highest within the same level.



Incorrect: The default transition exceeds the boundary.

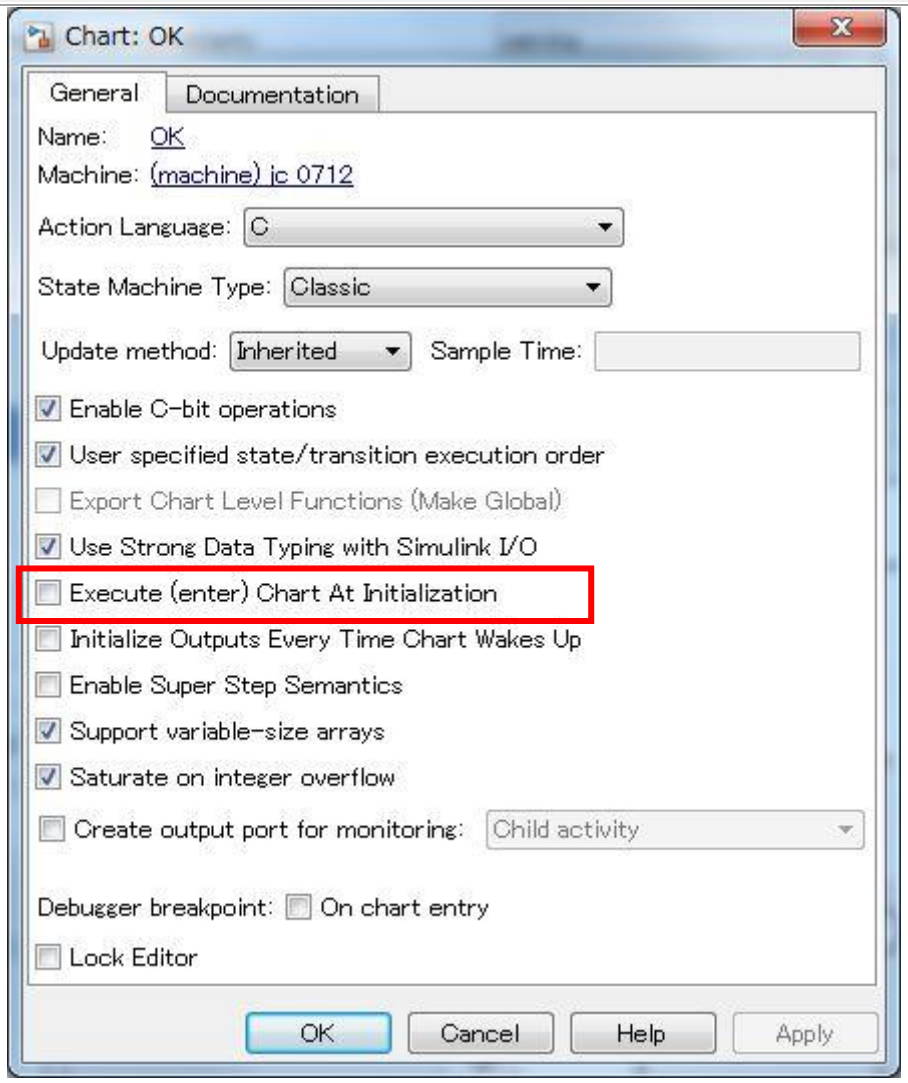


Incorrect:
There is no non-guard transition.

	
Notes	● If the state with default transition is placed on most left-upper position, transition goes from top to bottom or from left to right. ● Violation of“the default transition within a state chart must have a non-guard path to the state.” can be avoided by setting <Configuration><Diagnostics><Stateflow><No unconditional default transitions> to warning or error. 
See Also Guidelines	MISRA AC SLSF 042ABCDE MISRA AC SLSF 051A (051A is a rule about layouts)
Last Change	V4.0

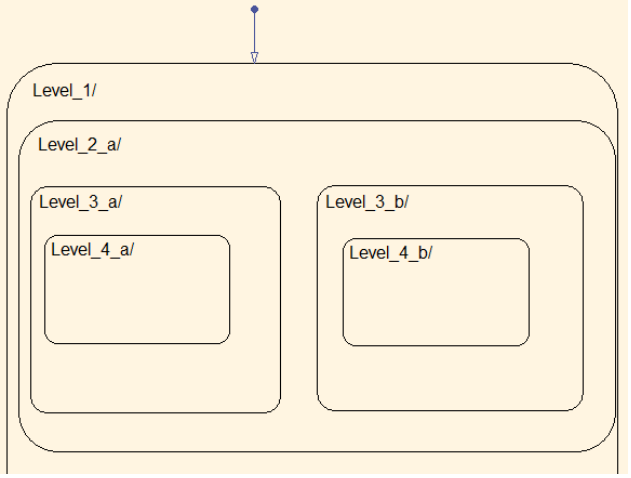
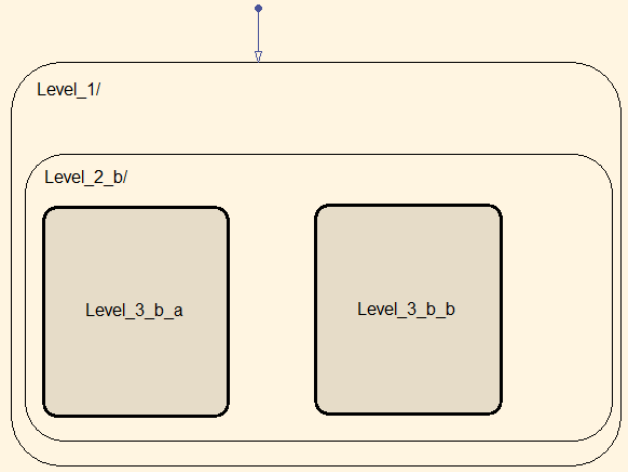
5.2.5. jc_0712: Execution timing for default transition path

ID: Title	**jc_0712: Execution timing for default transition path**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	In all Stateflow Charts, "Execute the specified Chart at time of initialization" must be deactivated. Release the selection of File > Chart Property > “Execute the specified Chart at time of initialization".

	
See Also	MISRA AC SLSF 034D
Last Change	V4.0

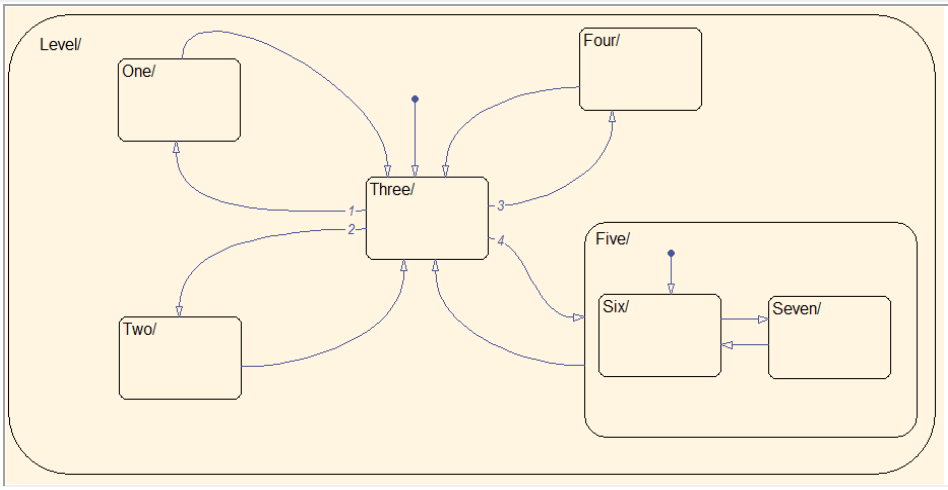
5.2.6. na_0038: Levels in Stateflow charts

ID: Title	**na_0038: Levels in Stateflow charts**
Priority	Recommended
Scope	MAAB
MATLAB Version	All
See Also	
Description	Within a single Viewer (Subviewer), multiple layering should be limited. For example, within a single Viewer (Subviewer), limiting goals for up to 3 levels should be established. If the constraint goals are exceeded, use subcharting to switch the screen. Incorrect: Level_4_a and Level_4_b have more than 3 levels, and are nested.

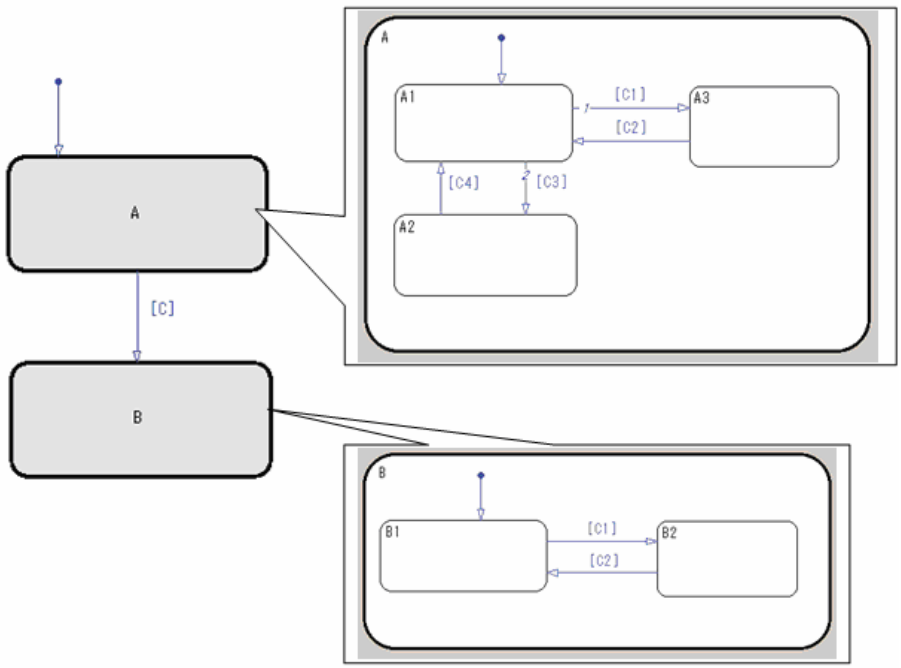
	 **Correct:** The 4th level is encapsulated in a subchart. 
See Also	
Last Change	V3.0

5.2.7. na_0040: Number of states per container

ID: Title	**na_0040: Number of states per container**
Priority	Recommended
Scope	MAAB
MATLAB Version	All
See Also	
Description	The number of viewable states per Stateflow Viewer (Subviewer) should be limited. (Typically to 6 to 9 states per Viewer) This number is based on the visible states in the diagram. Correct:

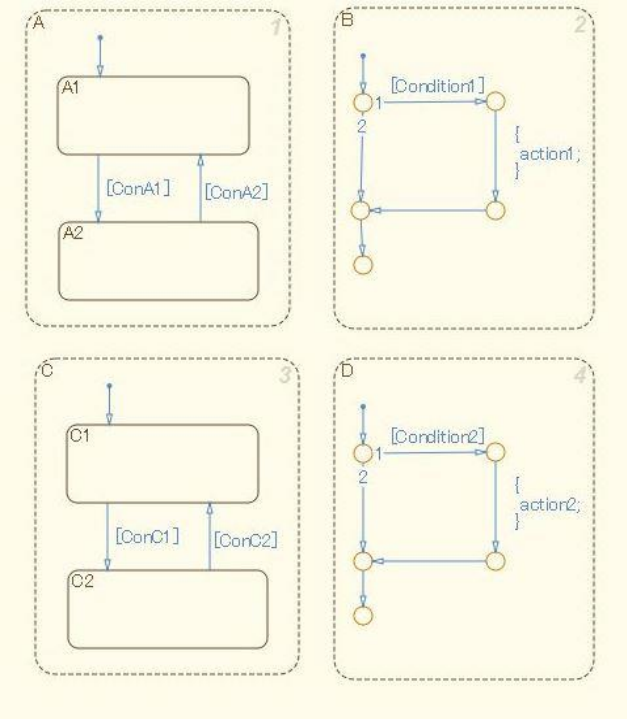
	
See Also	
Last Change	V3.0

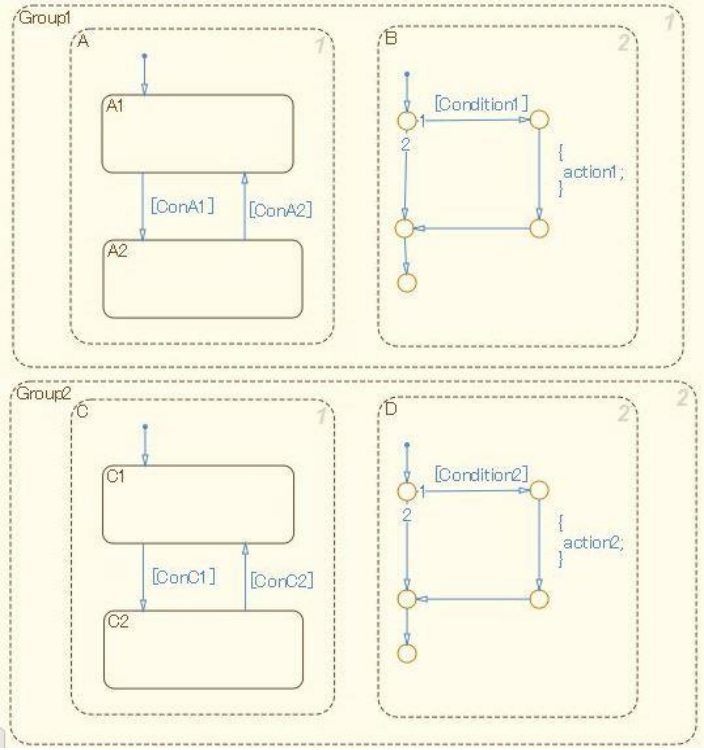
5.2.8. jc_0720: Guideline for using subcharting

ID: Title	**jc_0720: Guideline for using subcharting**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Subcharting is used in the following cases. ● State machine is hard to view on the screen. ● Hard to view in printed state. This rule is not applicable to Atomic Subchart. Subchart Example: 
See Also	MISRA AC SLSF 039B

Last Change	V4.0

5.2.9. jc_0721: Guidelines for using parallel states

ID: Title	**jc_0721: Guidelines for using parallel states**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Parallel states should not be used for the purpose of grouping. That is, the substates of parallel states should not be parallel states. Correct:  Incorrect: Substates of parallel states are parallel states.

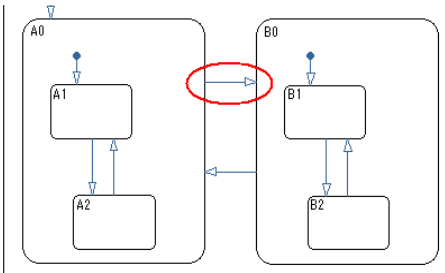
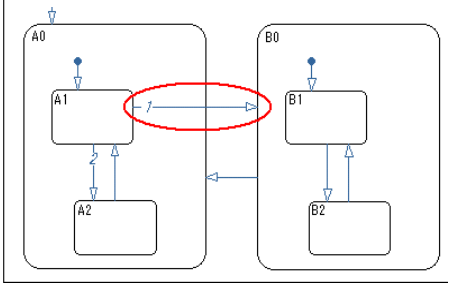
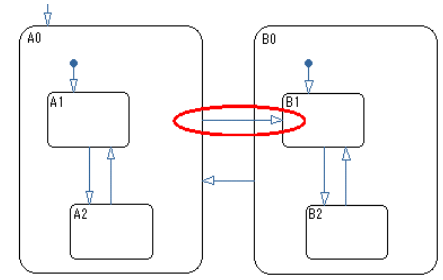
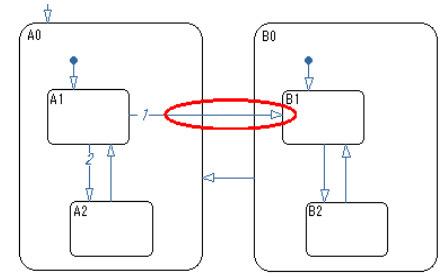
	 The four states (A, B, C, D) are in the same execution order, even if there is no parent (Group1, Group2).
See also	MISRA AC SLSF 040B
Last Change	V4.0

5.2.10. jc_0722: Guidelines for setting local variables in parallel states

ID: Title	**jc_0722: Guidelines for setting local variables in parallel states**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Unless the same data is required by two or more parallel states, the scope of local variables should be set to be restricted to one parallel state.
See also	MISRA AC SLSF 037D
Last Change	V4.0

5.2.11. jc_0723: Prohibited direct transition from external state to child state

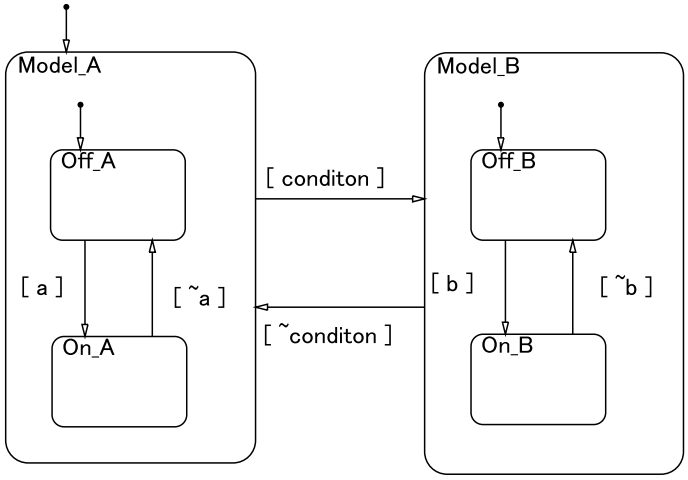
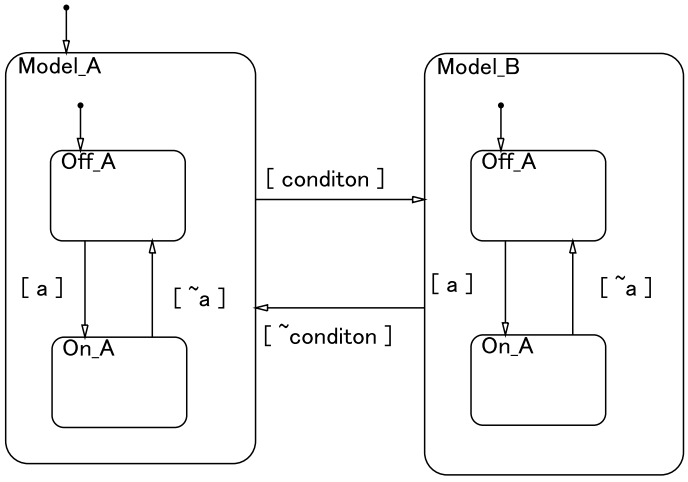
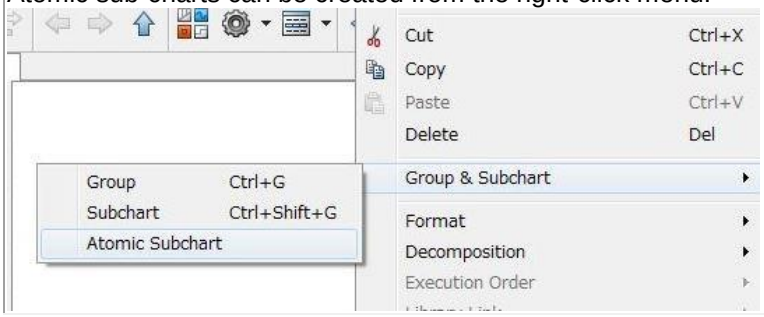
ID: Title	**jc_0723: Prohibited direct transition from external state to child state**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	The transition from external state to child state is prohibited. However, it is possible to transfer from child state to an external parent state.

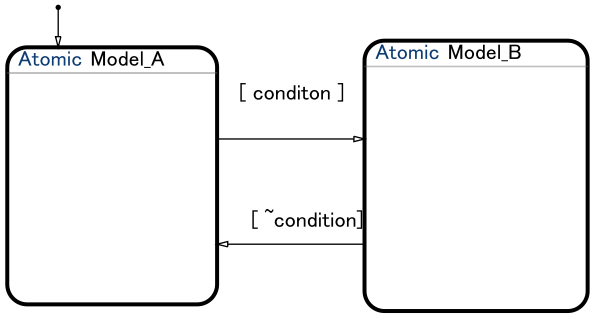
	● When viewed from the child, other parents that exist in parallel to one's own parent, or the child of other parents, exist outside of objects that have been encapsulated. Based on this premise, transitions that are direct transitions into other objects from outside of an object should be prohibited. If this transition is set, there is a high possibility that the concept of encapsulating the state is incorrect. Unless it is configured with a correct understanding of the concept of the state, the system will become complicated and the content would not be understood. By using such transitions, the state will become complicated, the specification will not be clear, and it will be a factor in causing mistakes in the specification itself. Correct: The transition from the super State A0 to another super State B0.  The transition from the child State A1 to another super State B0.  Incorrect: The transition from the super State A0 to another child State B1.  The transition from the child State A1 to another child State B1.  However, if the super state A0 is a virtual state that does not exist in reality and was created in order to use the internal transition or unify transition lines, it is classified in the state referred to as virtual state or "pseudo state" expressed in UML. For this kind of state, the above rules do not apply. Virtual states, or states referred to as pseudo states, and normal states should be written differently by distinction, and the scope of application of rules should be made clear.
See also	
Last Change	V4.0

5.3. Description of state label

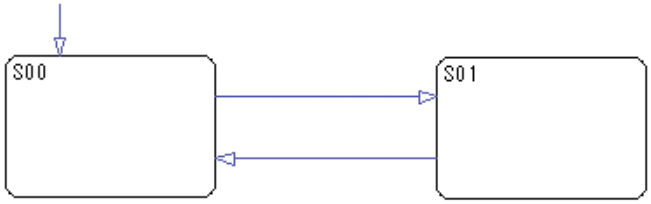
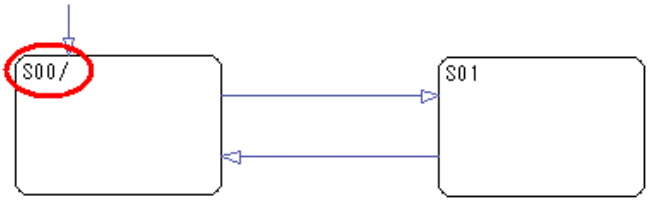
5.3.1. jc_0730: Independence of state name in charts

ID: Title	**jc_0730: Independence of state name in charts**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All

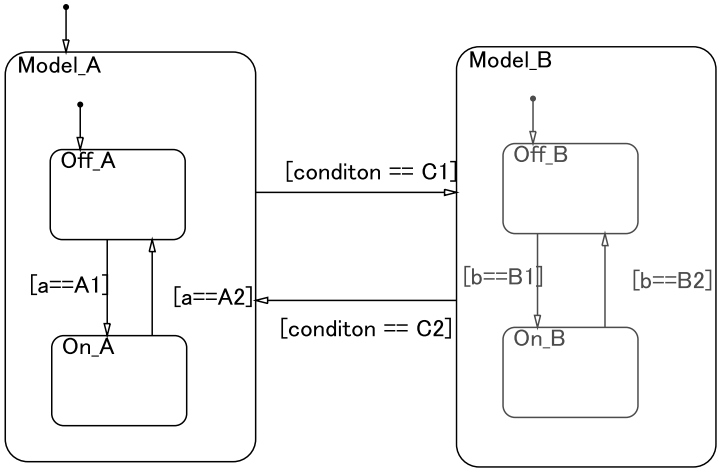
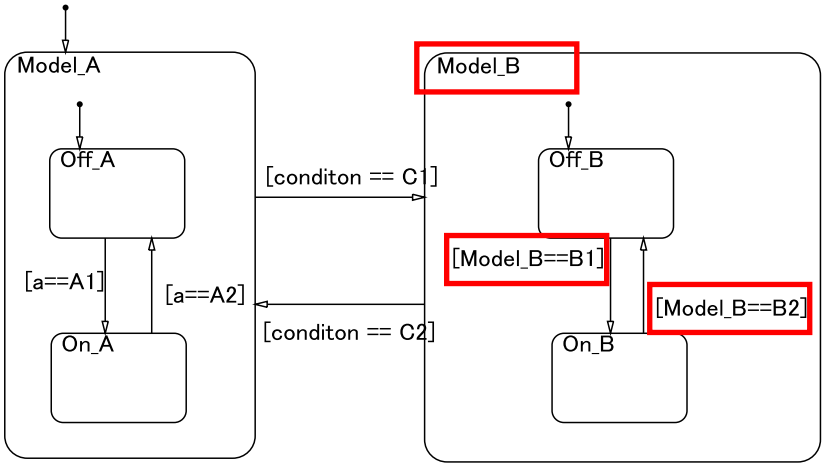
Prerequisites	
Description	State names must be unique in charts. Atomic sub-charts within charts should be treated as separate charts. In other words, state names must be unique in the atomic sub-charts, but there would not be a problem even if the same state name existed in a different atomic sub-chart. (Atomic sub-charts can be used from R2010b) Correct:  Incorrect:  Reference: Guidelines for creating an atomic sub-chart Atomic sub-charts can be created from the right-click menu. 

	
See also	MISRA AC SLSF 052A
Last Change	V4.0

5.3.2. jc_0731: Slash (/) in the state name

ID: Title	**jc_0731: Slash (/) in the state name**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Slashes (/) should not be included in state names. Start a new line after the state name without describing executable statements. Correct:  Incorrect:  In case of describing executable statements in continuation after state names, a slash (/) is required.
See also	
Last Change	V4.0

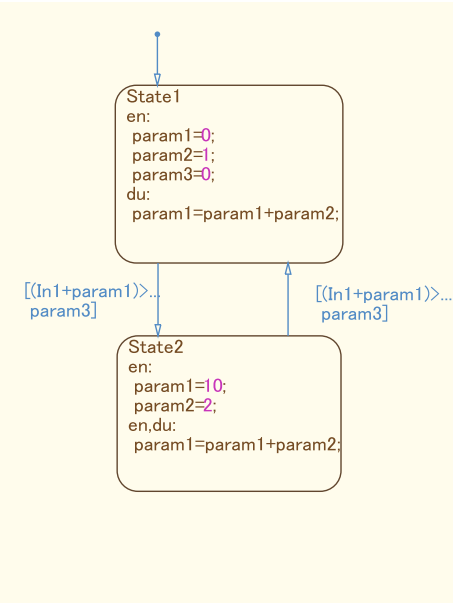
5.3.3. jc_0732 ：Distinction between state name and data item name

ID: Title	**jc_0732:** Distinction between state name and data item name
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	In a single chart, the same name as the data item should not be given to the state name. Correct:  Incorrect: 
See also	MISRA AC SLSF 052B
Last Change	V4.0

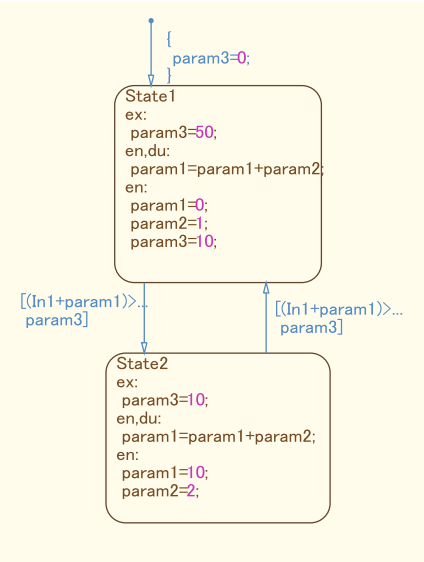
5.3.4. jc_0733: Order of state action types

ID: Title	**jc_0733: Order of state action types**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Action types should be stated in the order of entry (en), during (du) and exit (ex). ・In the case of describing combination action types (en,du: , du,ex: , en,ex: , en,du,ex: , the combination action types should only be described in the line at the top or the end.

Correct



Incorrect

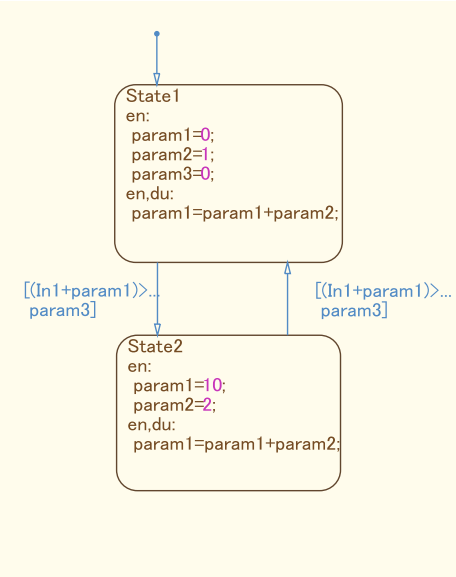
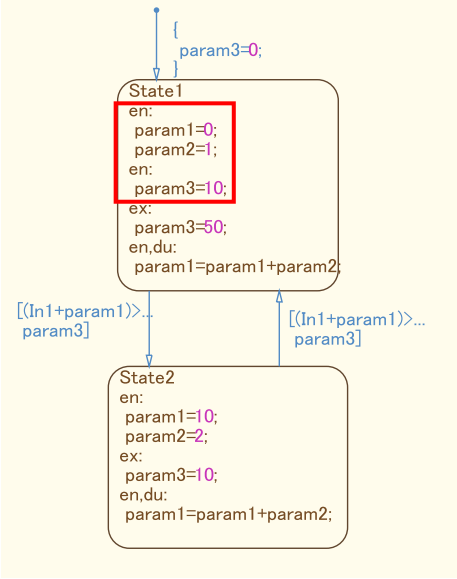


The combination statement is at the center of the whole.
The entry processing is described after the exit processing.

See also	MISRA AC SLSF 055A
Last Change	V4.0

5.3.5. jc_0734: Number of state action types

ID: Title	**jc_0734: Number of state action types**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	**jc_0733: Order of state action types**
Description	The same action types (entry (en), during (du), exit (ex), en, du: , du, ex: , en, ex: , en, du, ex:) should not be described two or more times. In particular, when using both the single actions of en and du and the combination action of "en, du: ", the execution order should differ depending on the order in which they are described.

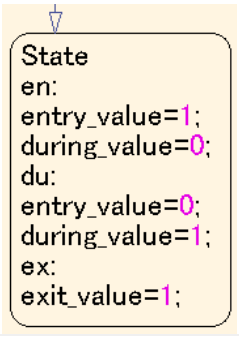
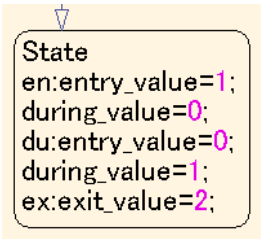
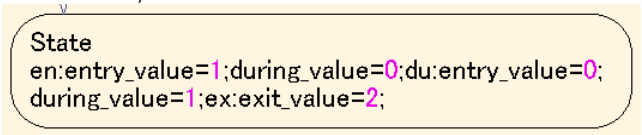
	If the action type is described more than once, the actual execution order will be hard to understand.
	Correct  Incorrect  The entry is separated in two and described twice.
See also	MISRA AC SLSF 055D
Last Change	V4.0

5.3.6. jc_0740: Usage restrictions of action type exit

ID: Title	**jc_0740: Usage restrictions of action type exit**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Exit should not be used when the design intent can be expressed by the transition destination entry, condition action and transition action. Because exit is executed when the state transitions to another state, the execution timing will become ambiguous.
See also	
Last Change	V4.0

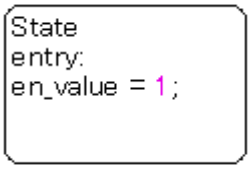
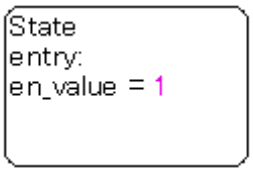
5.3.7. jc_0501: Format of entries in a State block

ID: Title	**jc_0501: Format of entries in a State block**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All

Prerequisites	
Description	A new line should: ● Start after state names. ● Start after the entry (en), during (du) and exit (ex) statements " : ". ● Start after the completion of an assignment statement ";". Correct:  Incorrect: Failed to start a new line after en, du and ex.  Incorrect: Failed to start a new line after the completion of an assignment statement ";". 
Notes	This rule has intention of not indicating actions, such as en and du, behind the State name. It does not become violation even if it attaches / behind the State name. jc_0731 is prohibition of /.
Last Change	V4.0

5.3.8. jc_0735: Semicolons in state label

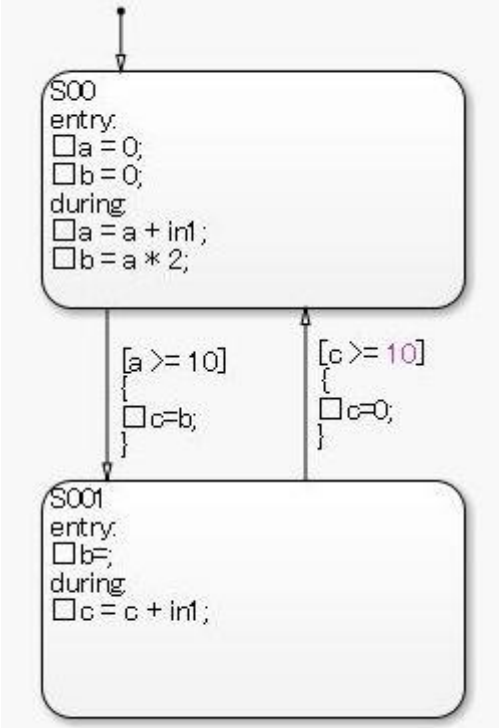
ID: Title	**jc_0735: Semicolons in state label**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	The end of each action in state label must be a semicolon ";" . Note: Action types (entry(en), during(du) and exit(ex)) are not subject to this rule. Correct:

	 Incorrect:  If the semicolon ";" is taken out, the value is outputted to the command window after running the simulation. It is convenient if it is used when checking operations, but the simulation speed will be slower.
See also	MISRA AC SLSF 043D
Last Change	V4.0

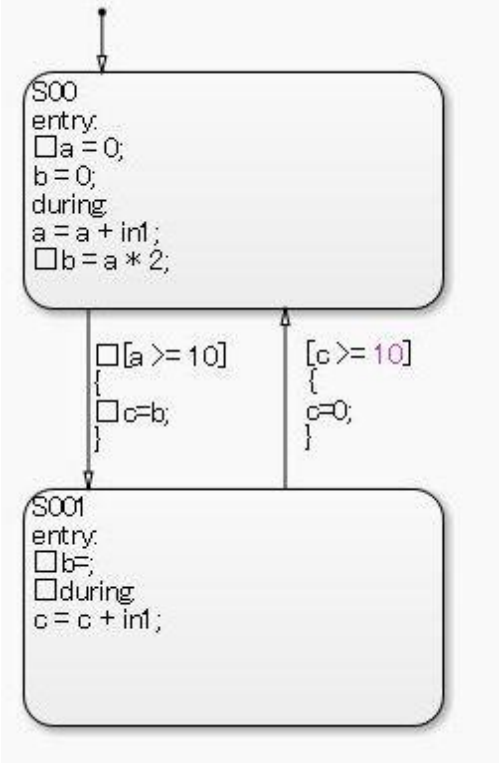
5.3.9. jc_0736: Uniform indentations in Stateflow blocks

ID: Title	**jc_0736: Uniform indentations in Stateflow blocks**
Priority	Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	jc_0752: Parentheses of condition actions
Description	Indentations in Stateflow blocks should be described uniformly. 1. Example of state label rules ● No spaces in front of action types (entry (en), during (du) and exit (ex)) ● Insert one space for other statements. 2. Example of transition-condition and action rules ● Do not insert spaces before [ ]. 3. Example of transition-action rules ● Always insert one space.

Correct: Indentations are described uniformly.



Incorrect: Indentations are not uniform.



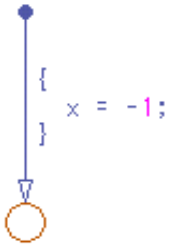
Note:
In versions after R2012b, it is possible to use MATLAB language-based charts called Chart MATLAB, instead of the conventional C-based ones.
In these MATLAB language-based charts, an indentation is automatically added at the time of describing the state labels. The rules of indentation are unified by the following:

- No spaces in front of entry (en), during (du) and exit (ex) statements

	● Insert one space for other statements. Transition lines do not have an automatic indentation function.
See also	
Last Change	V4.0

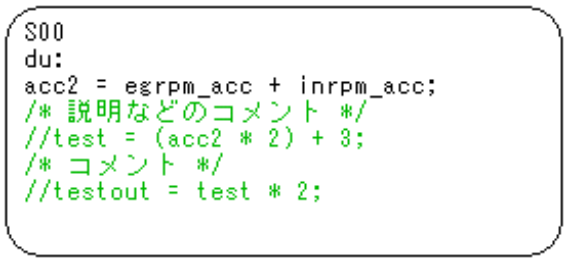
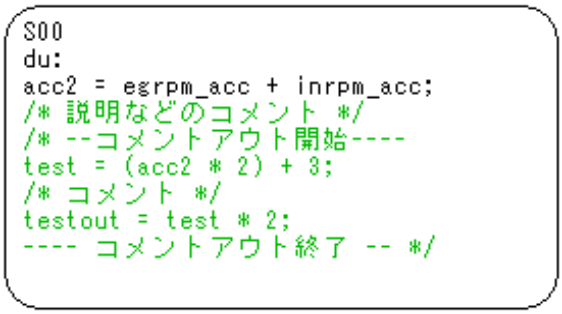
5.3.10. jc_0737: Uniform spaces before and after operators

ID: Title	**jc_0737: Uniform spaces before and after operators**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Spaces before and after operators should be described uniformly. 1. Example of arithmetic operator rules ● For unary operators, do not insert a space between the operators and operands. ● For binary operators, insert one or more space between the operators and operands. Correct: A = B□+□(-1); Incorrect: Incorrect: A = B□+□(-□1); A = B+(-1); 2. Example of increment/decrement operator rules ● Do not insert a space between the operators and operands. Correct: a++; Incorrect: a□++ 3. Example of relational operator (comparative operator) rules ● Insert one or more space between the operators and operands. Correct: Correct [A □>□B] [A □>=□B] Incorrect: Incorrect: [A >B] [A >=B] 4. Example of logical operator and C-bit operator rules ● Insert one or more space between the operators and operands. However, in the case of using negation operators [! , ~] and the complement of C-bit operators [~], do not insert a space between the operators and operands. Correct: Correct: [A = B□&□C] [A = !B□&□C] Incorrect: Incorrect: [A = B&C] [A = !□B□&□C] 5. Example of assignment operator [=] and compound assignment operator [+=, -=, *=, /=, %=, <<=, >>=, ^=, |=] rules ● Insert one or more space between the operators and operands. Correct: A□=□B□+□(-1); A□+=□1; Incorrect: Incorrect: A=B+(-1); A+=1; 6. Example of pointer operator [*, &] rules ● Do not insert a space between the operators and operands. Correct:

	A = fcn(x,y,&map); Incorrect: A = fcn(x,y,&□map); 7. Example of array index [[]] rules ● Do not insert a space between the operators and operands. Correct: A = B[1] + C[k + 1]; Incorrect: A = B□[1] + C[k + 1]; 8. Handling of new lines ● New lines may be started in the middle of an expression if there are many characters in a line and it is unavoidable. ● For operators that require spaces, it is okay for new lines to be started immediately before or after an operator, and the number of spaces before or after operators is optional. ● However, for operators in which spaces must not be inserted, new lines should not be started. Correct:  Incorrect: The expression is hard to read because there are no spaces. 
Notes	Operand: It means operation target value and variables in computer programming. For example, in the expression “A+10”, “A” and “10” are operands. “+” is operator. Unary operator is used to express minus value like “-1” Binary operator is used to express one operation like “+” in “K+3”
See also	
Last Change	V4.0

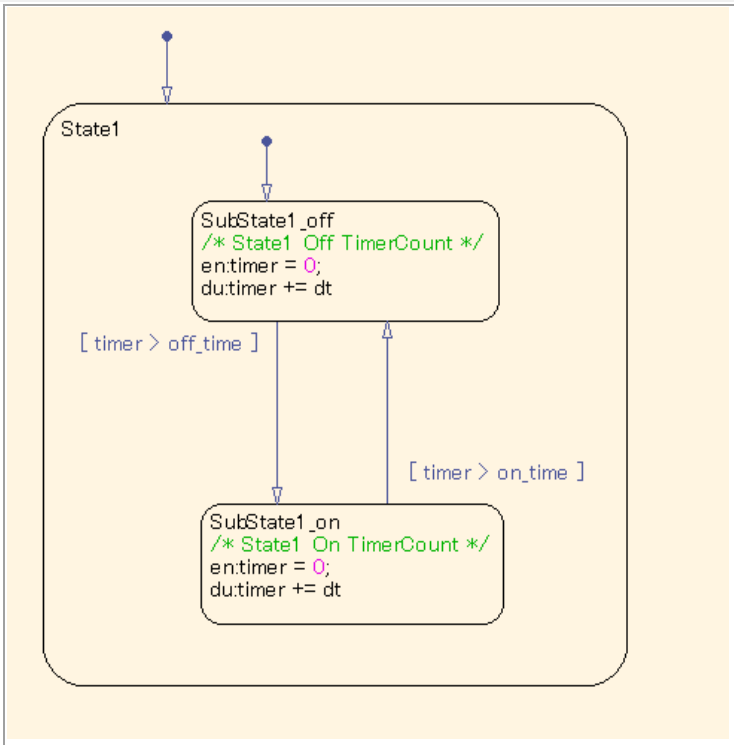
5.3.11. jc_0738: Guidelines for writing comments in state actions

ID: Title	**jc_0738: Guidelines for writing comments in state actions**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All

Prerequisites	
Description	When using /* */ in the comment, new lines must not be started in the middle. This is in order to prevent duplicated comment symbols /* */ Correct:  Incorrect: 
Notes	Note: In versions after R2012b, it is possible to use MATLAB language-based charts called Chart MATLAB, instead of the conventional C-based ones. In these MATLAB language-based charts, only the % comments used in MATLAB language can be used for comments.
See also	
Last Change	V4.0

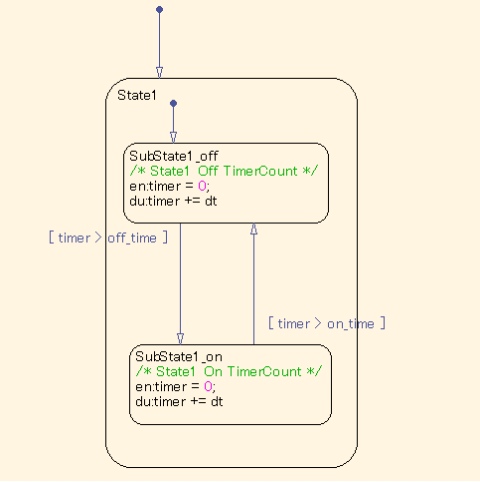
5.3.12. jc_0739: Guidelines for describing texts inside states

ID: Title	**jc_0739: Guidelines for describing texts inside states**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Texts inside states should not be described beyond the boundaries of that state. Correct:

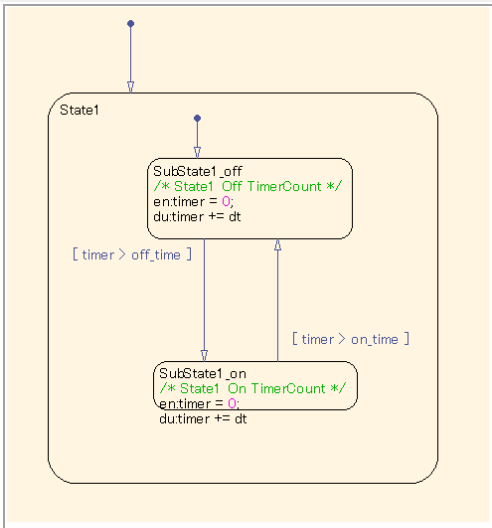
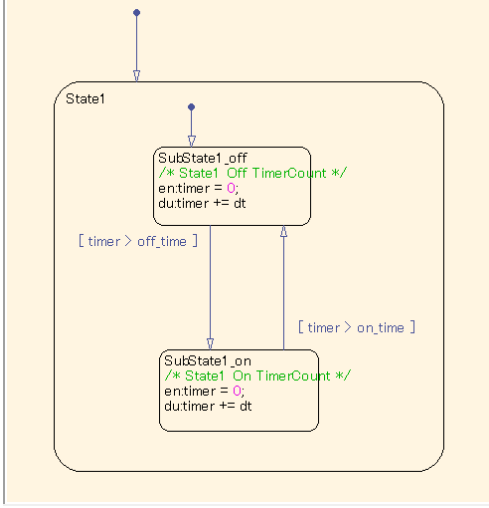


Incorrect:

The transition condition goes beyond the boundaries of the state

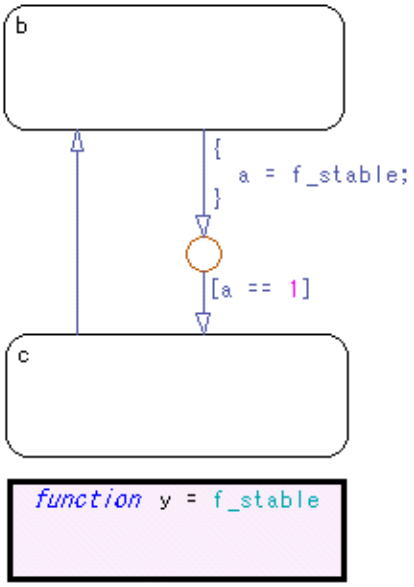
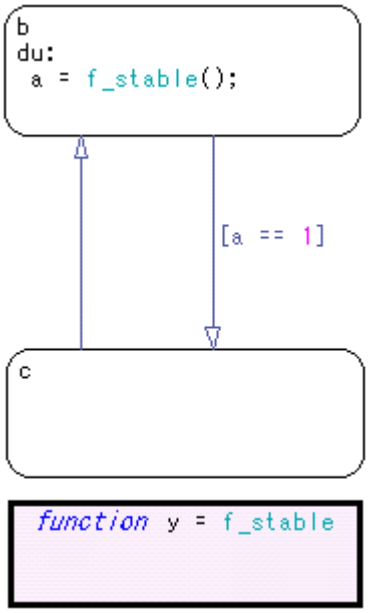


The state action goes beyond the boundaries of the state

	 The comment goes beyond the boundaries of the state 
See also	MISRA AC SLSF 050F
Last Change	V4.0

5.3.13. jc_0741: Timing to update the variables used in the state's transition conditions

ID: Title	**jc_0741: Timing to update the variables used in the state's transition conditions**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Variables that will be used in the state's transition conditions should not perform an update by "during". Note The processing of "during" will be executed if a transition does not occur after the state's transition conditions are executed. If a statement such as the "Incorrect" below is made, there is a possibility that the transition timing will be delayed by one sampling because the transition will use the results executed one time prior.

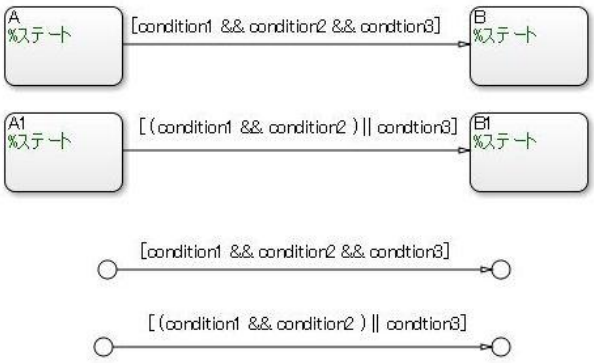
	Correct:  Incorrect: 
See also	
Last Change	V4.0

5.4. Conditions and conditional actions

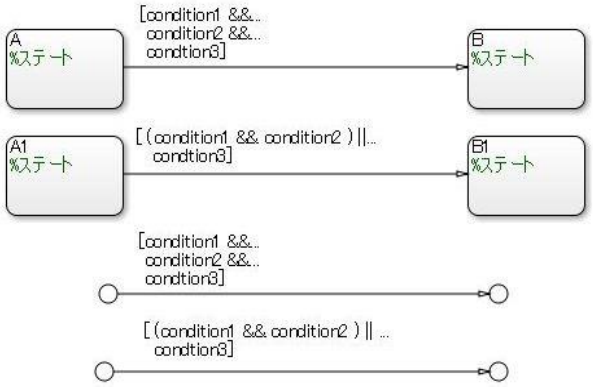
Describes the method of condition description that will be common in the description of both state transition and Flow Chart.

5.4.1. jc_0742: Guidelines for writing Boolean operations in condition labels

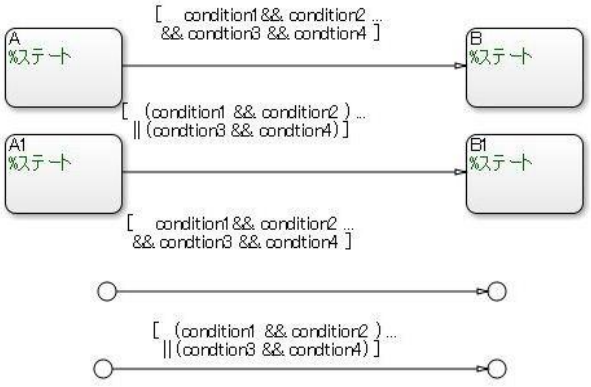
ID: Title	**jc_0742: Guidelines for writing Boolean operations in condition labels**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	jc_0751: Backtracking prevention in state transition jc_0773: Guidelines for describing exception processing
Description	● If there are up to three conditions, they can be described on one line. ● If there are two or more types of Boolean operations, priorities should be described using parentheses. ● If there are four or more conditions, they can be described in more than one line. ● If two or more types of Boolean operations are described in more than one line, position of operations (before conditions or after conditions) should be unified in the chart. Correct: Multiple conditions are described on one line



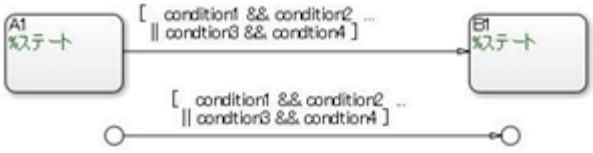
Correct:
Multiple conditions are described on more than one line
(positions of operations are unified to after conditions)



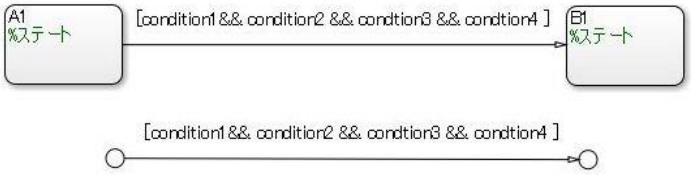
Correct:
Multiple conditions are described on more than one line
(positions of operations are unified to before conditions)



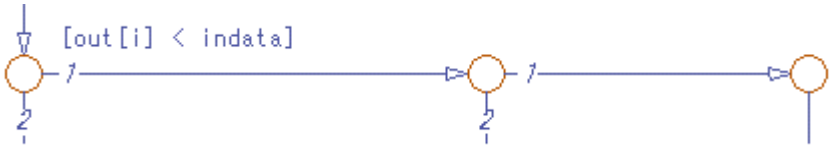
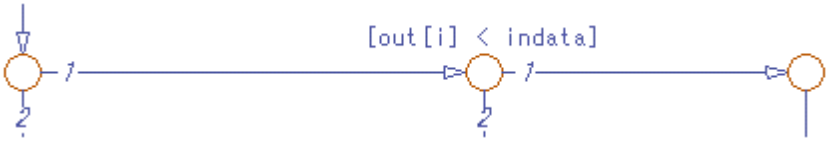
Incorrect
Although different types of logical operator exist, priority by using parenthesis is not shown.



Incorrect:

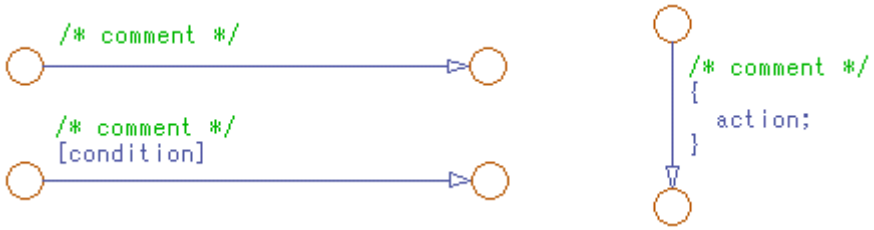
	Four conditions are described in one line. 
Notes	In this rule, the writing method of Boolean operations within conditions as well as its limitations are described. Here, there is no description of how to separate a single condition or its limitations. In the case of separating a condition, a description that adheres to jc_0751 for state transitions and to jc_0773 for Flow Charts is necessary so that backtracking does not occur.
See also	
Last Change	V4.0

5.4.2. jc_0770: Placement of conditional statements and action statements

ID: Title	**jc_0770: Placement of conditional statements and action statements**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	db_0129: Stateflow transition appearance
Description	For the placement of the conditional statement of the Flow Chart and the action statement, select either of the following and unify it within the model (1 is recommended) 1. Describe from near the transition origin of the transition (transition line). 2. Describe near the center of the transition (transition line). It is important to know which transition's condition the conditional statement and action statement belong to. Also ensure that the conditional and action statements do not overlap with other characters and lines. (db_0129: Stateflow transition appearance) Correct:  Incorrect: It is difficult to know which transition (transition line) the condition belongs to. 

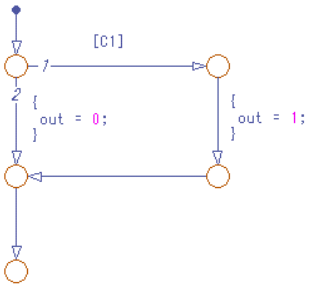
Notes	
See also	
Last Change	V4.0

5.4.3. jc_0771: Placement of comments in transition lines

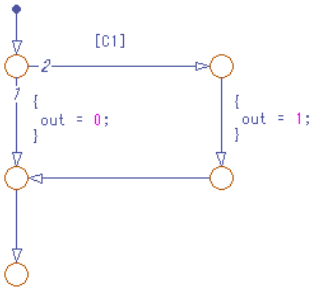
ID: Title	**jc_0771: Placement of comments in transition lines**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Placement of comment descriptions in transition lines ・ Unify to either the top or bottom of the conditional statement. ・ Unify to either the top or bottom of the action statement. "Unifying the descriptions to the top side" is recommended. It is important that which conditional statement the comment corresponds to is explicitly stated. Example 
See also	
Last Change	V4.0

5.4.4. jc_0772: Execution order and transition conditions of transition lines

ID: Title	**jc_0772: Execution order and transition conditions of transition lines**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Transitions other than the last one in the execution order must always set conditions. Correct:

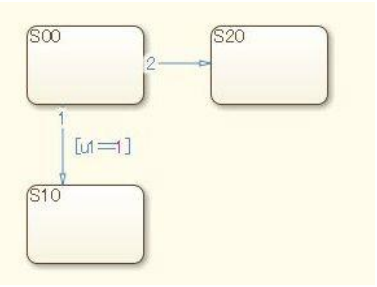


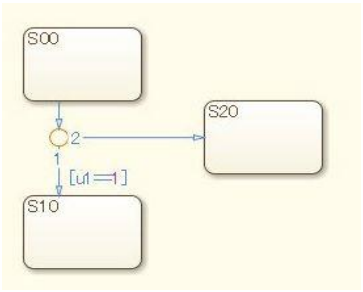
Incorrect:



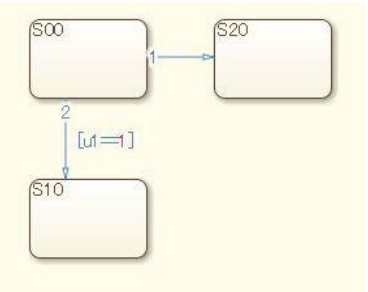
Execution order 1 is an unconditional transition and conditional expression [C1] is described in execution condition 2.

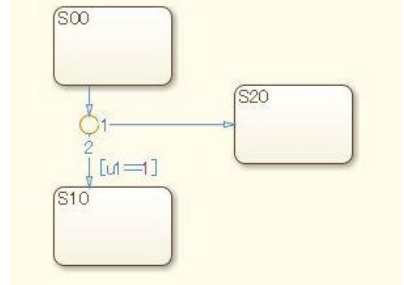
Examples includes state transition
Correct:





Incorrect
Priority of unconditional transition is higher than conditional transition.



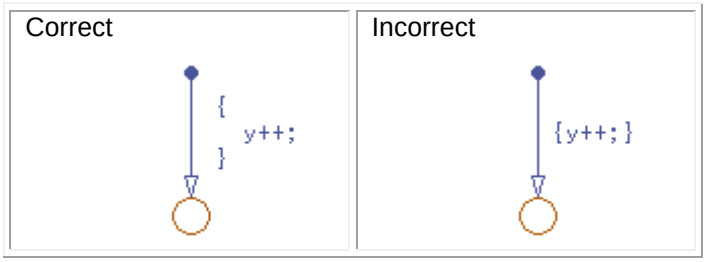


In state transition, uncontitional transition is not invariably necessary.

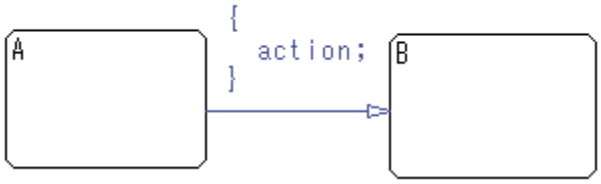
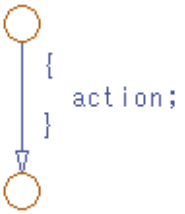
See also	
Last Change	V4.0

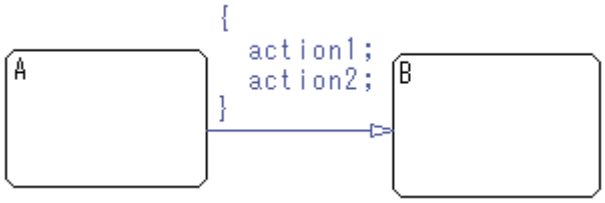
5.4.5. jc_0752: Parentheses of condition actions

ID: Title	**jc_0752: Parentheses of condition actions**
Priority	Recommended

Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	The parentheses of condition actions should make one line just by the parentheses. (Start a new line before and after parentheses.)  The example was described in the Flow Chart but the same applies to state transitions.
See also	MISRA AC SLSF 054E
Last Change	V4.0

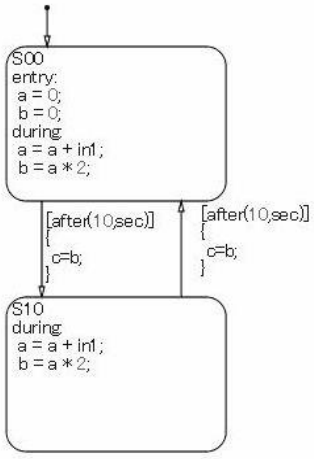
5.4.6. jc_0743: Guidelines for writing condition actions

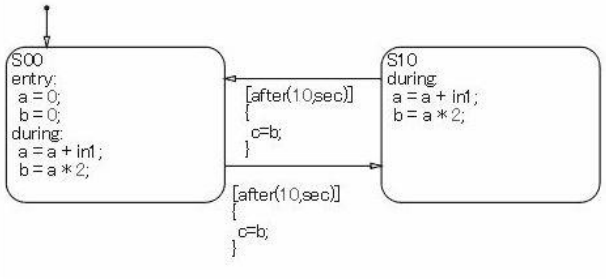
ID: Title	**jc_0743: Guidelines for writing condition actions**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	The writing method of state condition actions and Flow Chart actions are shown below.. Describe a semicolon (;) at the end of an action. If there is one action Example of a state condition action:  Example of a Flow Chart condition action:  If there are two or more actions, describe them in more than one line. (Multiple actions should not be described in 1 line.)

	Example of a state condition action:  Example of a Flow Chart condition action: 
See also	
Last Change	V4.0

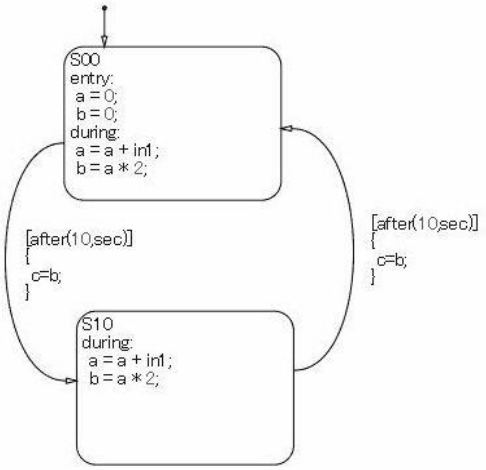
5.5. State transition

5.5.1. jc_0750: Guidelines for drawing transition lines in Stateflow

ID: Title	**jc_0750: Guidelines for drawing transition lines in Stateflow**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	The transition lines inside the state chart are drawn by horizontal and vertical straight lines. Correct Vertical:  Correct Horizontal:

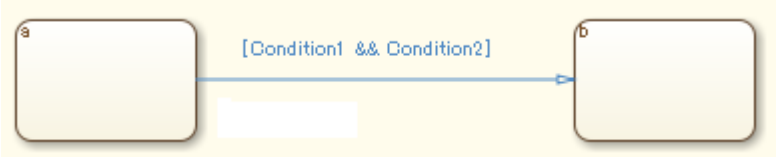
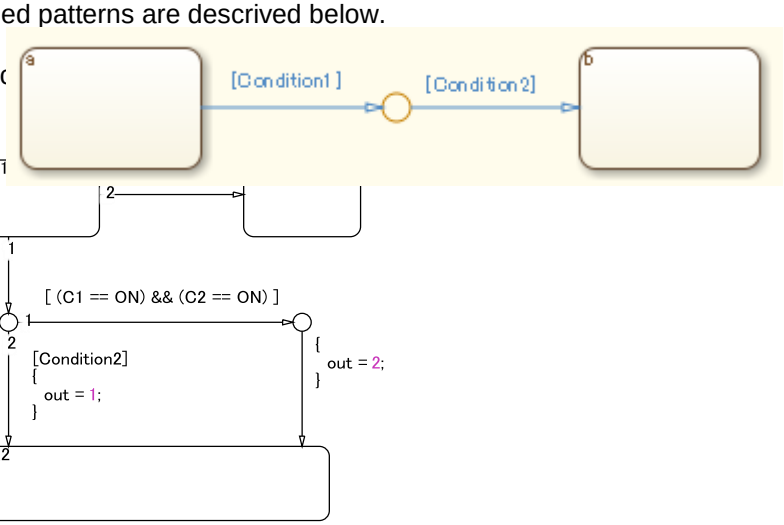
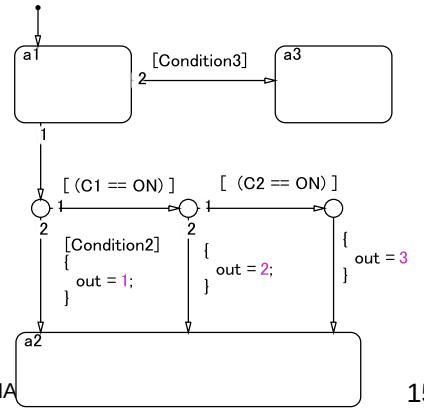


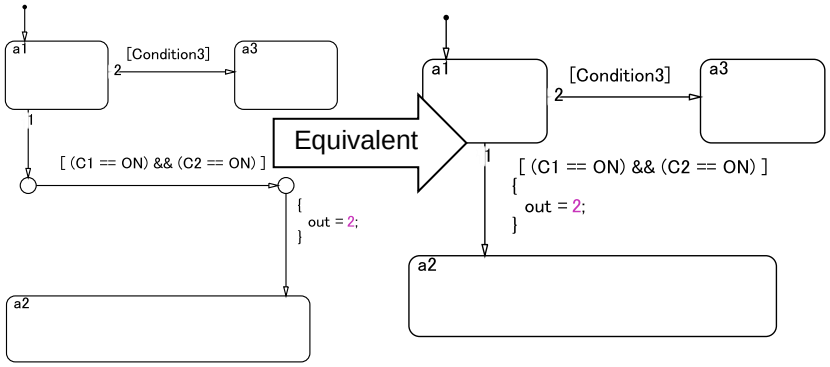
Incorrect:
The transition lines from state to state are connected by curved lines.



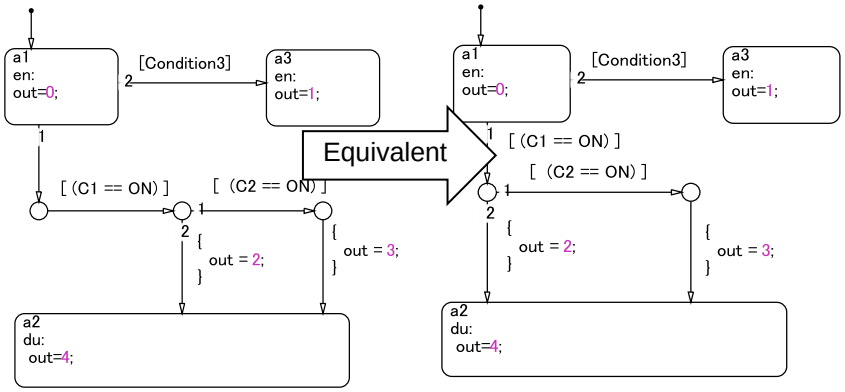
See also	MISRA AC SLSF 053E
Last Change	V4.0

5.5.2. jc_0751 : Backtracking prevention in state transition

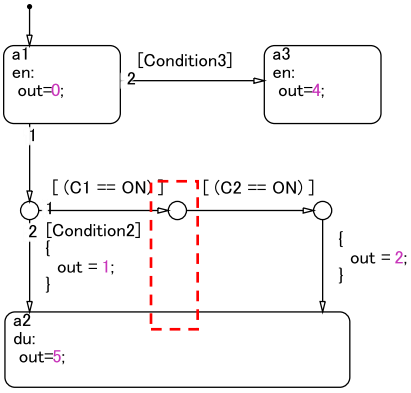
ID: Title	**jc_0751: Backtracking prevention in state transition**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Complex conditions must not be separated by connective junctions. (In order to prevent backtracking) Correct: Complex conditions are described all together. Incorrect: Complex conditions are separated by a connective junction.  Detailed patterns are descrived below. Correc  Correct: All connective junctions have branches.  Correct: Two conditions are described together.



Correct: Connective junction between conditions has branches.

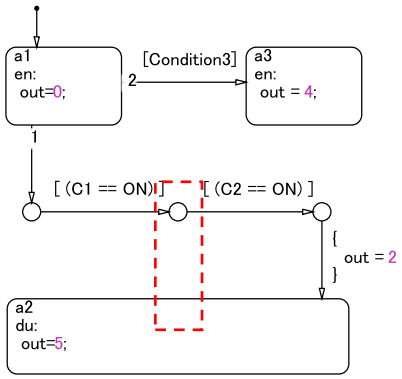


Incorrect: Connective junction between conditions has only one path.



In case of C1==ON and C2==OFF, transition seems to be terminated on the connective junction after [(C1==ON)]. However, in case of C2==OFF, backtracking occurs and {out=1} is executed.

Incorrect: Connective junction between conditions has only one path.

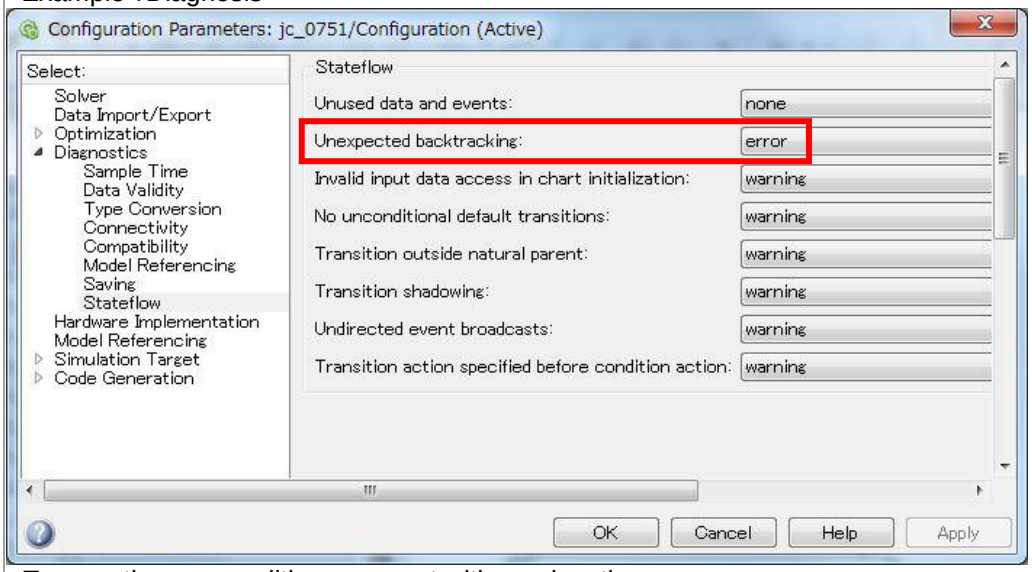


In case of C1==ON and C2==OFF, transition seems to be terminated on the connective junction after [(C1==ON)]. However, in case of C2==OFF, backtracking occurs and [Condition3] is evaluated. In that case, even If C1==ON is true, if [Condition3] is true, transition to a3 occurs.

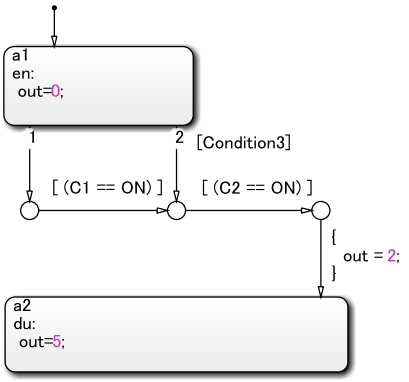
Notes

jc_0773 intends to prevent backtracking on flowchart. Complying to that rule, unconditional transition is designed to ensure transition reaches terminal junction.
However, on state transition, chart is intentionally designed so that transition doesn't reach to terminal junction. This means if condition is not met, transition does not occur.
First connective junction is not branched. This can be understood as intended transition inhibition.
However, in case not branched connective junction is placed between two conditions, unintended backtracking may occur. This rule is in order to prevent it.

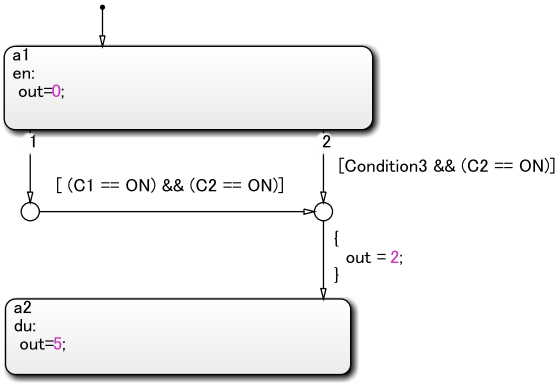
Description:
Example :Diagnosis



- Two continuous conditions connect with one junction.
- conditions exist further behind that.
This is detected as an unexpected back truck king.



This problem is detectable by diagnosis.
This rule is summarizing the beforehand continuous conditions to one, and aims at preventing this problem beforehand.
The upper model is designed as follows.

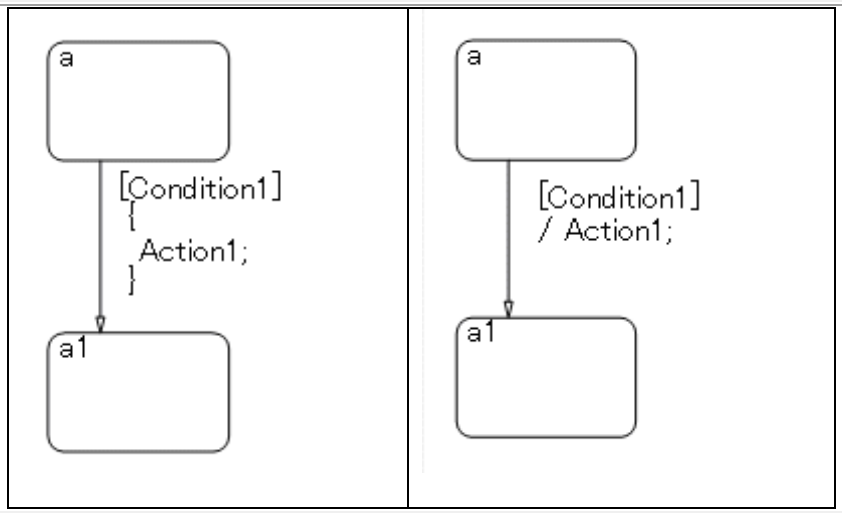


See also	MISRA AC SLSF 043C
Last Change	V4.0

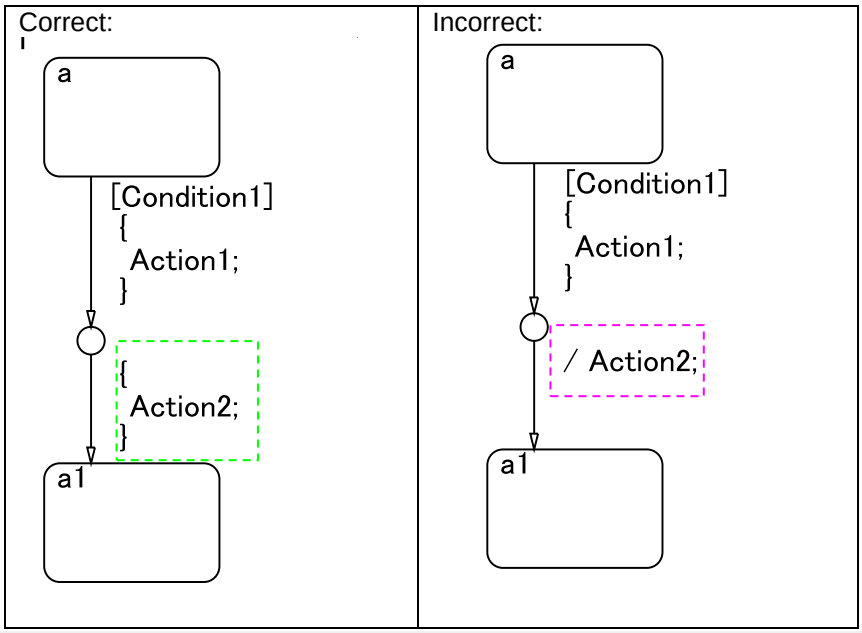
5.5.3. jc_0754: Transition actions in Stateflow

ID: Title	**jc_0754: Transition actions in Stateflow**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Transition actions must not be used.

Correct:	Incorrect:

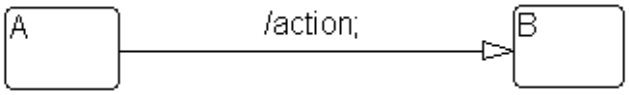
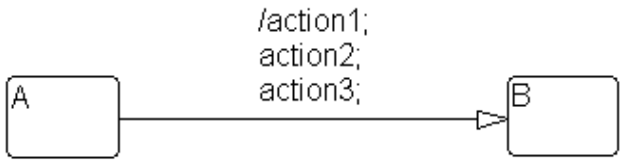
	
See also	MISRA AC SLSF 043B
Last Change	V4.0

5.5.4. jc_0753: Condition actions and transition actions in Stateflow

ID: Title	**jc_0753: Condition actions and transition actions in Stateflow**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Condition actions and transition actions should not be mixed within the same chart. They should be integrated into one. 
See also	MISRA AC SLSF 043 A
Last Change	V4.0

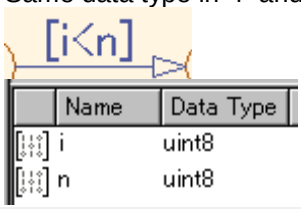
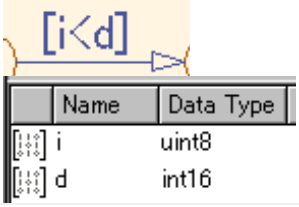
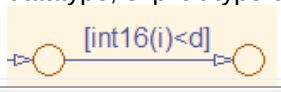
5.5.5. db_0151: State machine patterns for transition actions

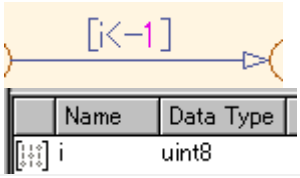
ID: Title	**db_0151: State machine patterns for transition actions**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The following patterns are used for transition actions within state machine patterns:
Last Change	V2.2

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;*	

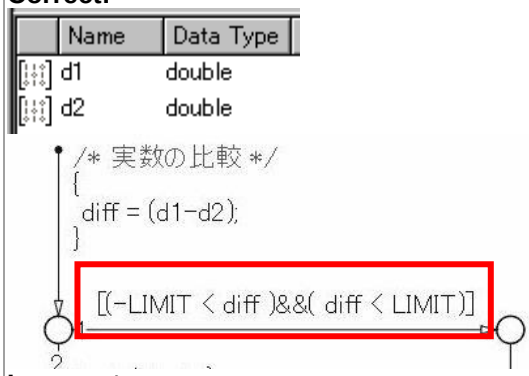
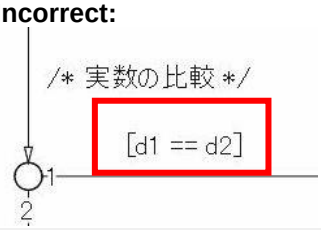
5.5.6. na_0013: Comparison operation in Stateflow

ID: Title	**na_0013: Comparison operation in Stateflow**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	● Comparisons should be made only between variables of the same data type. ● If comparisons are made between variables of different data types, the variables need to be explicitly type cast to matching data types.

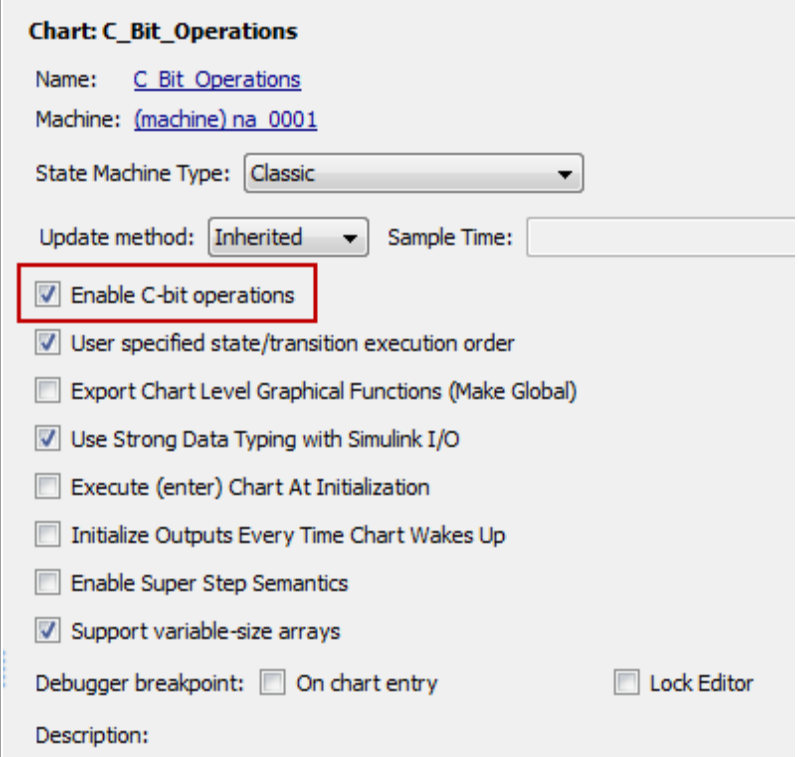
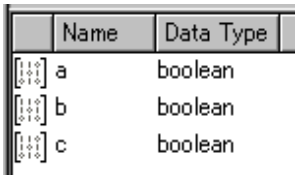
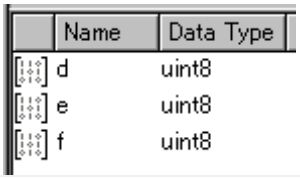
Correct: Same data type in "i" and "n" 	**Incorrect:** Different data type in "i" and "n" 
Correct: Although “i” and “n” have different datatype, explicit type cast is applied. 	

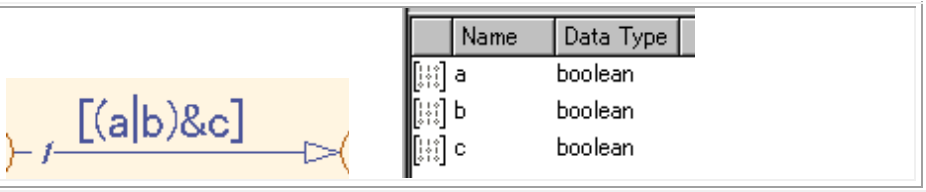
	 **Incorrect:** Do not make comparisons between unsigned integers and "negative numbers." 
Last Change	V2.1

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

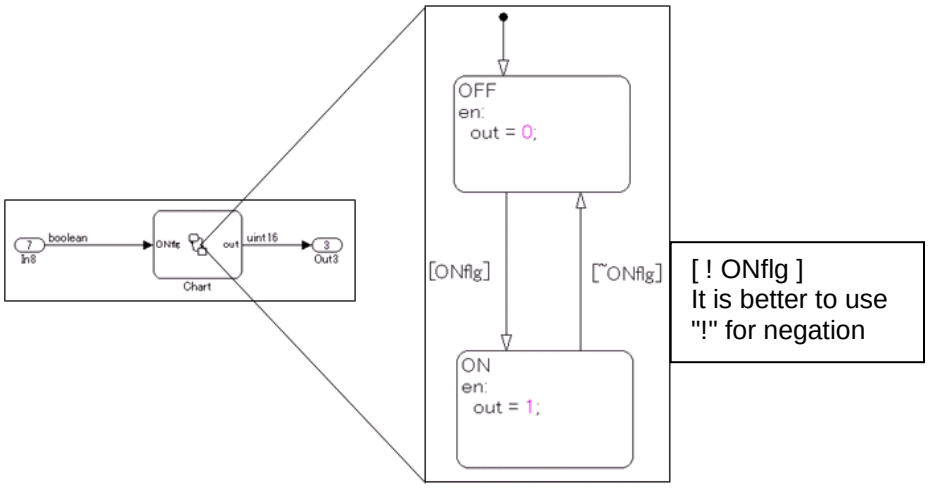
ID: Title	**jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	● Do not use hard equality comparisons (Var1 == Var2) or (Var1 != Var2) or (Var1 ~= Var2) with two floating point numbers. ● If a hard comparison is required, a margin of error should be defined and used in the comparison (LIMIT in the example). ● Hard equality comparisons may be done between integer data types. **Correct:**  **Incorrect:** 
Last Change	V2.0

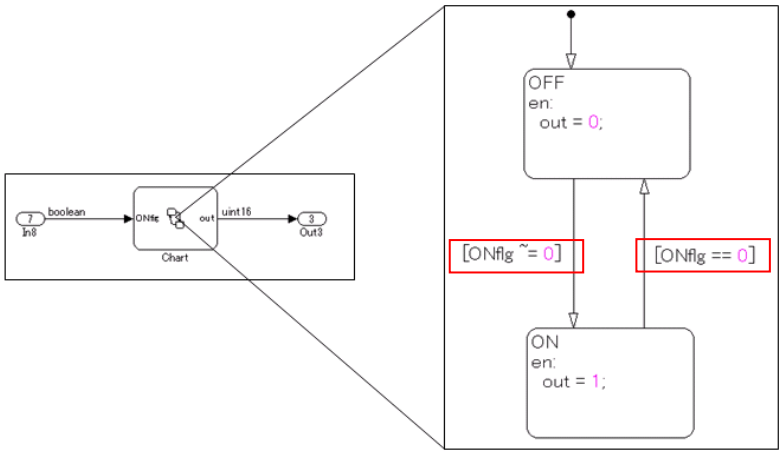
5.5.8. na_0001: Bitwise Stateflow operators

ID: Title	**na_0001: Bitwise Stateflow operators**
Priority	Strongly Recommended
Scope	MAAB
Prerequisites	
Description	Bitwise operators (*&”,”|”,”^”,”~”) should not be used other than bit operations. To enable bitwise operations, follow the steps below: 1. Select File > Chart Properties. 2. Select "Enable C-bit operations".  **Correct:** Use && and "||" for Boolean operation.   Use & and | for bit operation.   **Incorrect:** Use & and "|" for Boolean operation.

	
Notes	List of operational effect of each operator Operator | C-bit operations are enabled: OFF | ON a|b: Boolean OR of a,b | Bitwise OR of a,b a||b: Boolean OR of a,b | Boolean OR of a,b a&b: Boolean AND of a,b | Bitwise AND of a,b a&&b: Boolean AND of a,b | Boolean AND of a,b a^b: bth power of a | Bitwise XOR of a,b ! a: Boolean negation of a | Boolean negation of a ~a: Boolean negation of a | Complement of a
Last Change	V4.0

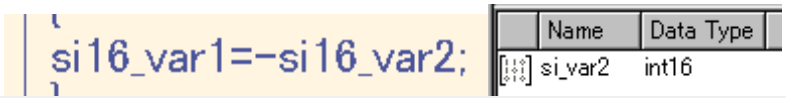
5.5.9. jc_0655: Prohibited comparison operation of logical type signal in Stateflow

ID: Title	**jc_0655: Prohibited comparison operation of logical type signal in Stateflow**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	jc_0757: Condition expressions should set a comparison operator na_0002: Appropriate implementation of fundamental logical and numerical operations
Description	● Logical operations must not be applied to boolean values. Boolean type signals must not be compared with numbers (0, 1) or logical values (true, false). ● Use Boolean operation (NOT) when inverting logical type signals. ● Usage of either ~ or ! for negation should at least be uniform in the chart. Preferably, specified rules should be made for each project pertaining to the writing method of negative statements, and they should be unified in the model. Correct:  Incorrect:

<table>
<tr><td></td><td> 

When using logical type signals as condition flags, it is not necessary to write the match with true or false. If performing code generation, an optimized code will be outputted regardless of whether it is described. This rule emphasizes readability by uniformity of appearance.</td></tr>
<tr><td>See also</td><td></td></tr>
<tr><td>Notes</td><td>na_0001: If rules of bitwise Stateflow operators are adopted and bitwise operations are made valid, "~" will be a complement of 2. If it is used in conjunction with na_0001, only "!" can be used for negation.
<table>
<tr><td rowspan="2">Operator</td><td colspan="2">C-bit operations are enabled</td></tr>
<tr><td>OFF</td><td>ON</td></tr>
<tr><td>a|b</td><td>Logical OR of a,b</td><td>Bitwise OR of a,b</td></tr>
<tr><td>a||b</td><td>Logical OR of a,b</td><td>Logical OR of a,b</td></tr>
<tr><td>a&b</td><td>Logical AND of a,b</td><td>Bitwise AND of a,b</td></tr>
<tr><td>a&&b</td><td>Logical AND of a,b</td><td>Logical AND of a,b</td></tr>
<tr><td>a^b</td><td>bth power of a</td><td>Bitwise XOR of a,b</td></tr>
<tr><td>! a</td><td>Logical negation of a</td><td>Logical negation of a</td></tr>
<tr><td>~a</td><td>Logical negation of a</td><td>Complement of 2</td></tr>
</table></td></tr>
<tr><td>Last Change</td><td>V4.0</td></tr>
</table>

5.5.10. jc_0451: Use of unary minus on unsigned integers in Stateflow

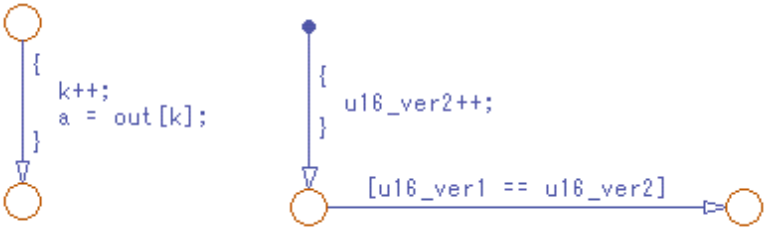
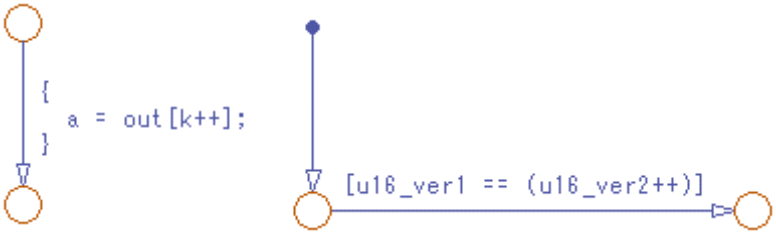
<table>
<tr><td>ID: Title</td><td>jc_0451: Use of unary minus on unsigned integers in Stateflow</td></tr>
<tr><td>Priority</td><td>Strongly Recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td></td></tr>
<tr><td>Description</td><td>Do not perform unary minus on unsigned integers.
Correct:


Incorrect:

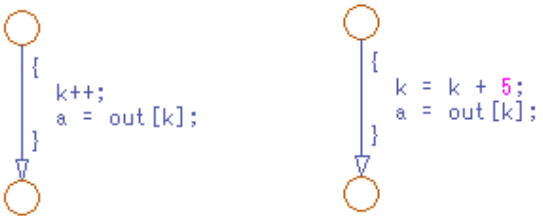
</td></tr>
</table>

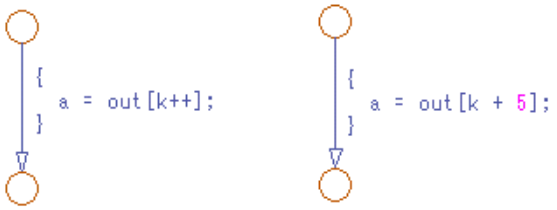
Last Change	V2.0

5.5.11. jc_0755: Guidelines for use of increments/decrements

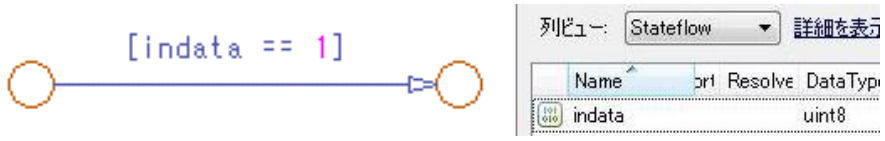
ID: Title	**jc_0755: Guidelines for use of increments/decrements**
Priority	Mandatory
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Increment/decrement operators should be used as one action. Correct:  Incorrect: 
See also	
Last Change	V4.0

5.5.12. jc_0756: Prohibited use of operation expressions in array indexes

ID: Title	**jc_0756: Prohibited use of operation expressions in array indexes**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Sequence numbers should not be calculated in the array indexes. Correct:  Incorrect:

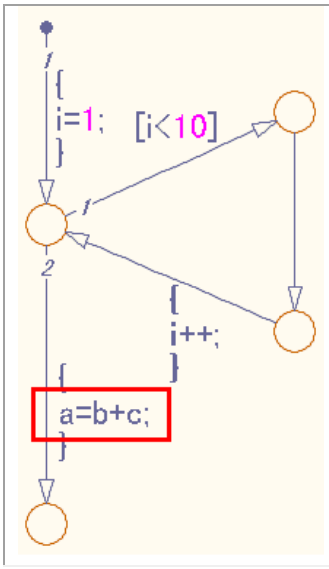
	
See also	
Last Change	V4.0

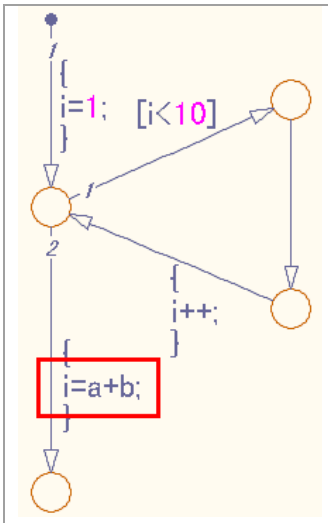
5.5.13. jc_0757: Guidelines for describing condition expressions

ID: Title	**jc_0757: Guidelines for describing condition expressions**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	jc_0655: Prohibited comparison operation of logical type signal in Stateflow
Description	Expressions which return boolean value should be used for condition expressions. Correct:  Correct:  Incorrect: 
See also	
Last Change	V4.0

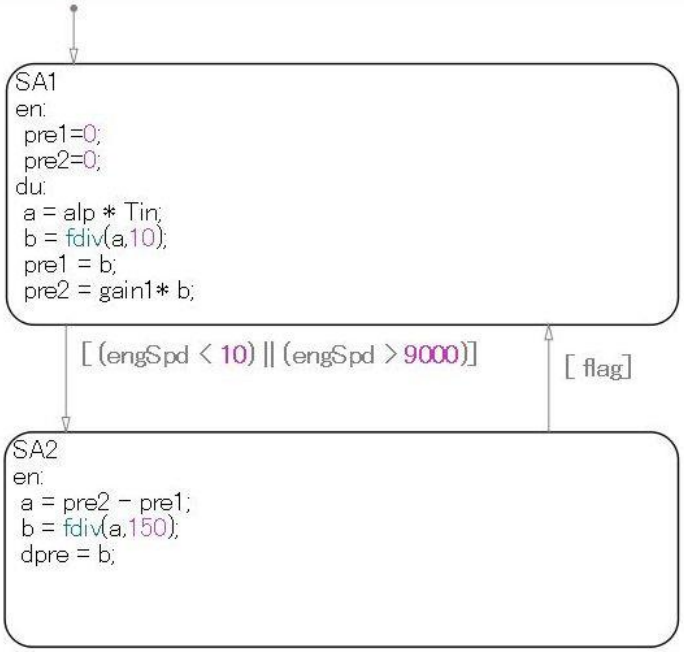
5.5.14. jc_0491: Reuse of variables within a single Stateflow scope

ID: Title	**jc_0491: Reuse of variables within a single Stateflow scope**
Priority	Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	The same variable should not have multiple meanings (usages) within a single Stateflow state. **Correct:** Variable of loop counter must not be used other than loop counter. **Incorrect:** The meaning of variable "i" changes from the index of the loop counter to the sum of a+b.

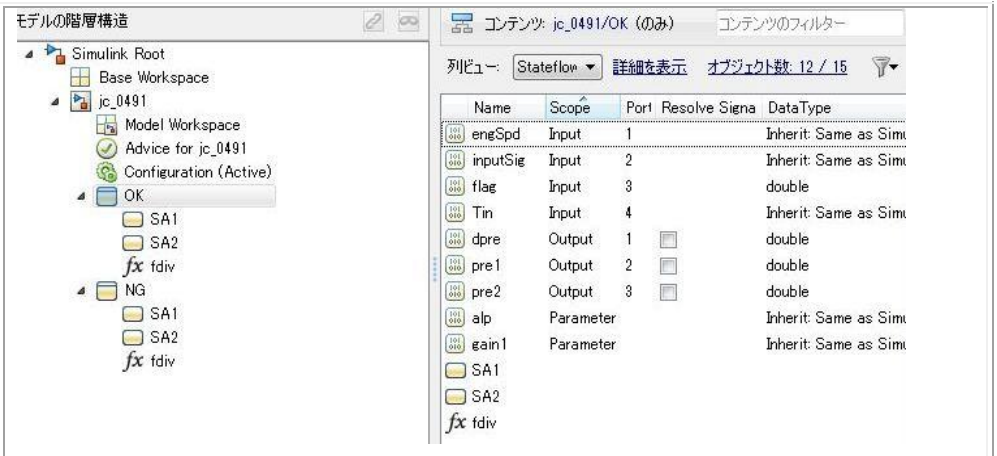


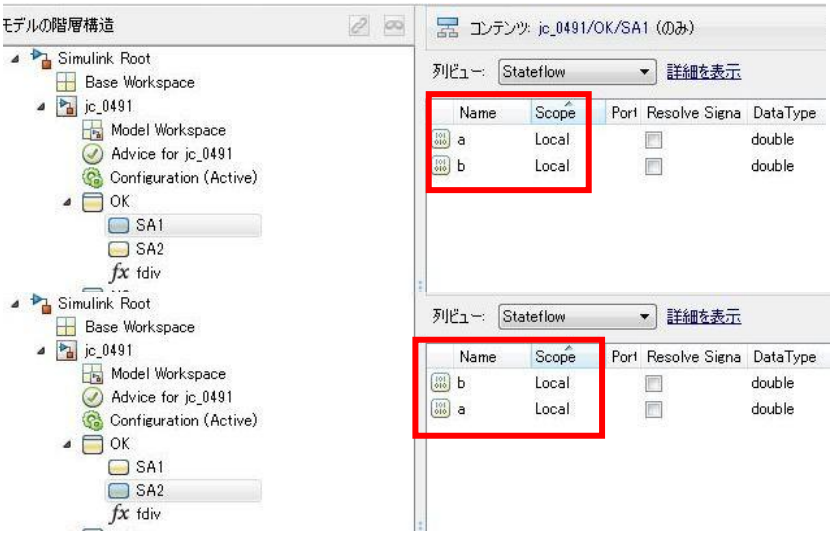


Examples in state transition
Definitions of temporary calculation result variables a,b which are used only in each state.

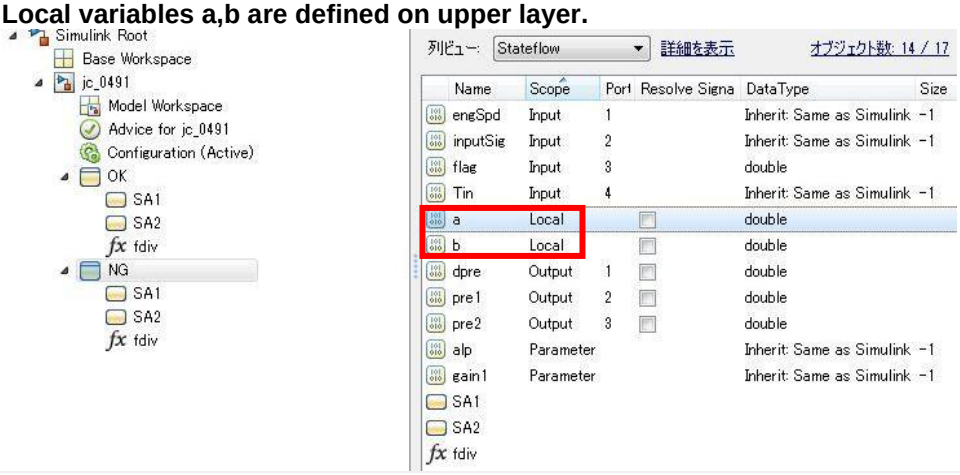


Correct:
Variables a,b are declared as local variable in each state.





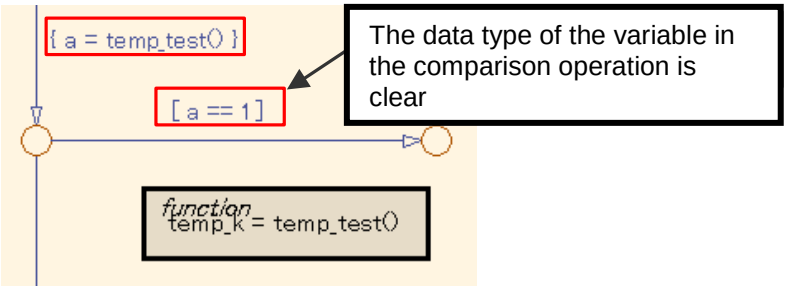
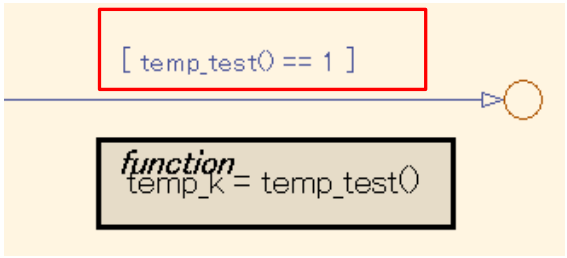
Incorrect:
Local variables a,b are defined on upper layer.



Last Change	V2.2

5.5.15. jc_0521: Use of the return value from graphical functions

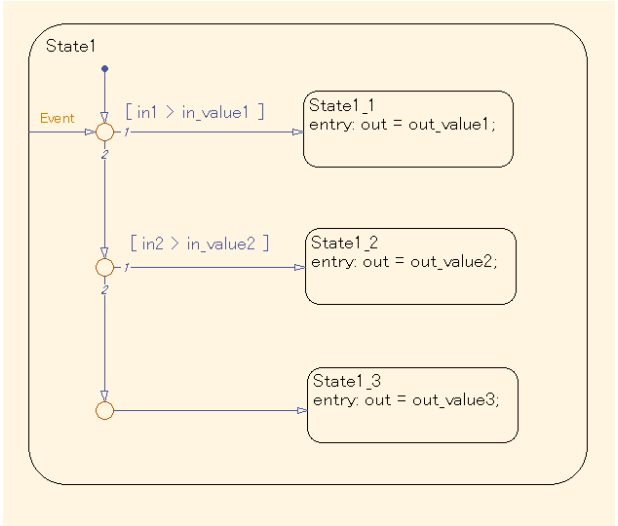
ID: Title	**jc_0521: Use of the return value from graphical functions**
Priority	Recommended
Scope	JMAAB

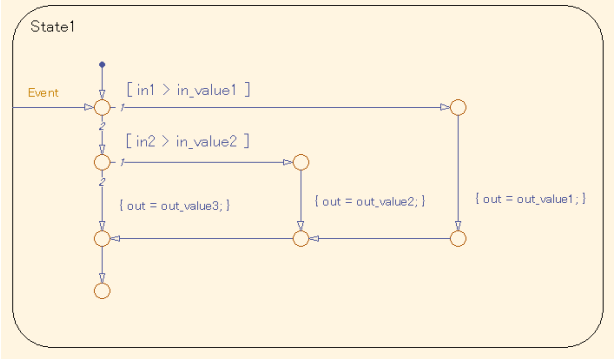
MATLAB Version	All
Prerequisites	
Description	The return value from a graphical function should not be used directly in a comparison operation. **Correct:** An intermediate variable is used in the conditional expression after the assignment of the return value from the function "temp_test" to the intermediate variable "a".  **Incorrect:** Return value of the function "temp_test" is used in the conditional expression. 
Last Change	V2.0

5.6. Internal transition of the state transition

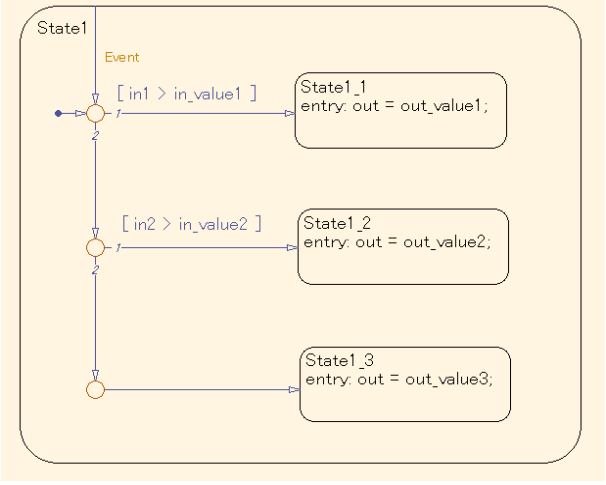
5.6.1. jc_0760: Starting point of internal transition in Stateflow

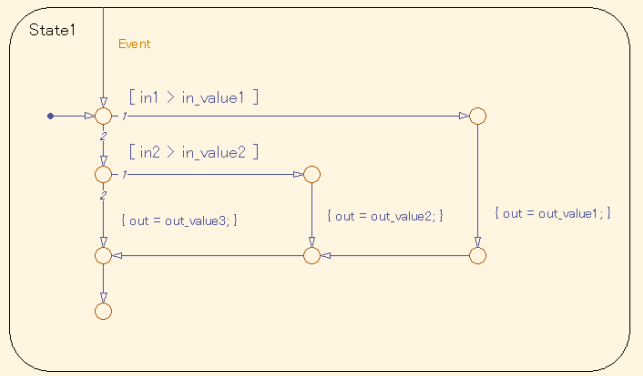
ID: Title	**jc_0760: Starting point of internal transition in Stateflow**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	In all state charts and Flow Charts, internal transitions from state boundaries should start from the left edge of the state. Correct:





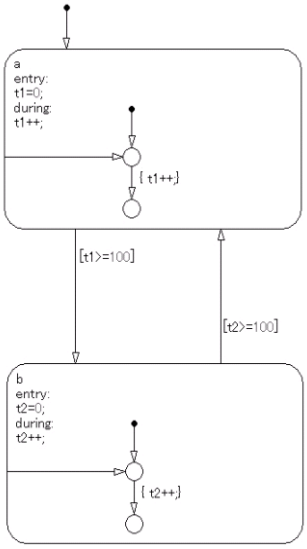
Incorrect:

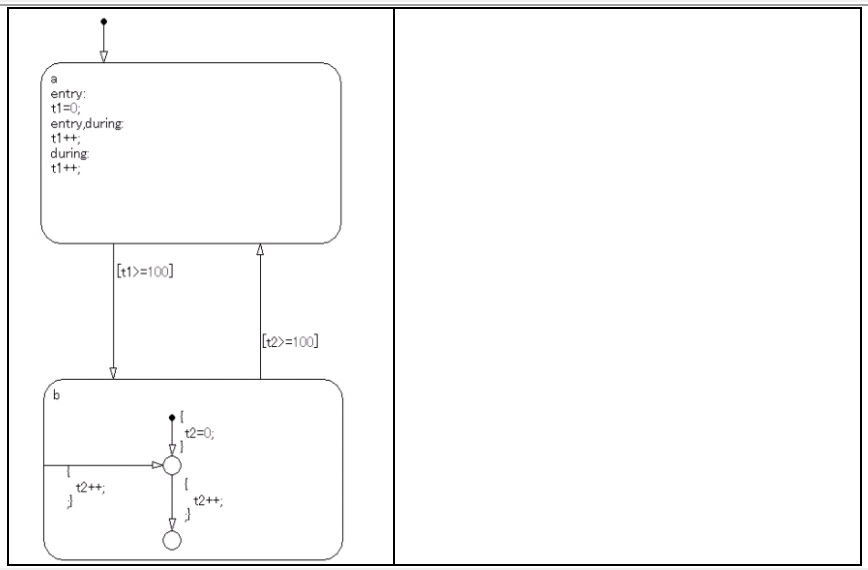


	
Notes	If the super state State1 used in the example above is a virtual state that does not exist in reality and was created in order to use the internal transition or unify transition lines, it is classified in the state referred to as virtual state or "pseudo state" expressed in UML. This state only unifies the transition lines, so it does not have a state action inside.
See also	MISRA AC SLSF 053F
Last Change	V4.0

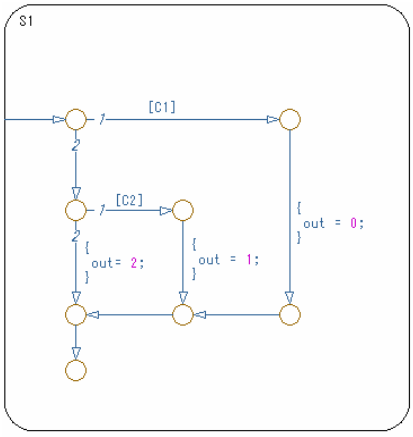
5.6.2. jc_0762: Prohibited combination of state action and Flow Chart

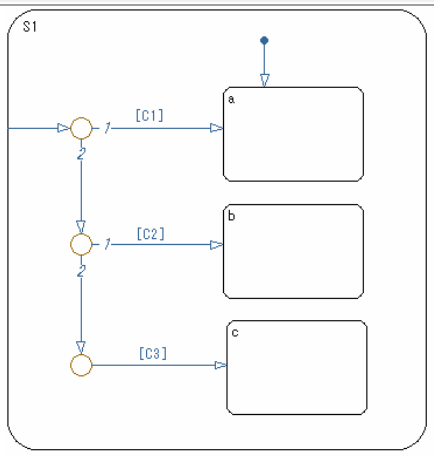
ID: Title	**jc_0762: Prohibited combination of state action and Flow Chart**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	State actions within states (starts with entry, during) and flow Chart statements should not be used in combination. Note The execution order is hard to understand in writing methods that use the two combined, and the behavior will not be understood intuitively. It should be unified to either a state action or Flow Chart statement.

Correct:	Incorrect:
	

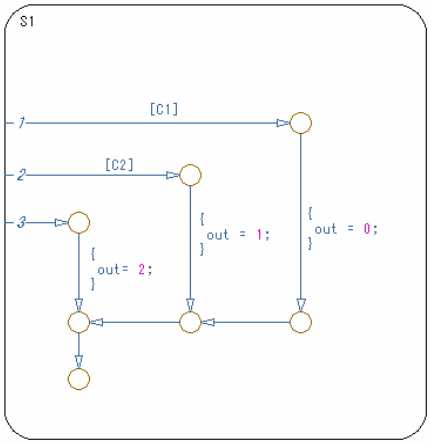
	
See also	
Last Change	V4.0

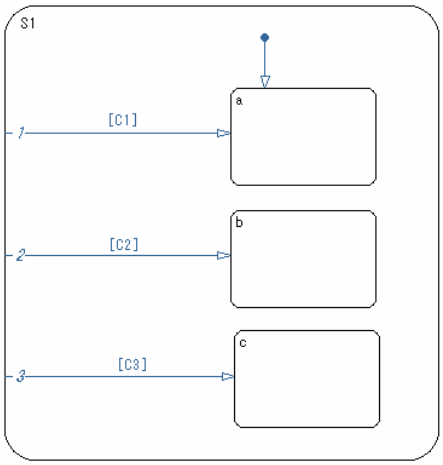
5.6.3. jc_0763: Usage restrictions of multiple internal transitions

ID: Title	**jc_0763: Usage restrictions of multiple internal transitions**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	Multiple internal transitions should not be used within a single state. Correct: 



Incorrect:

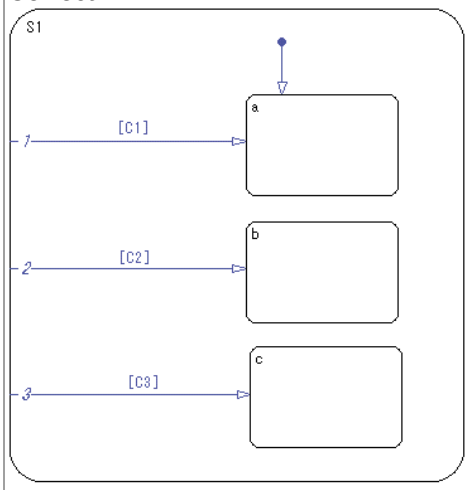
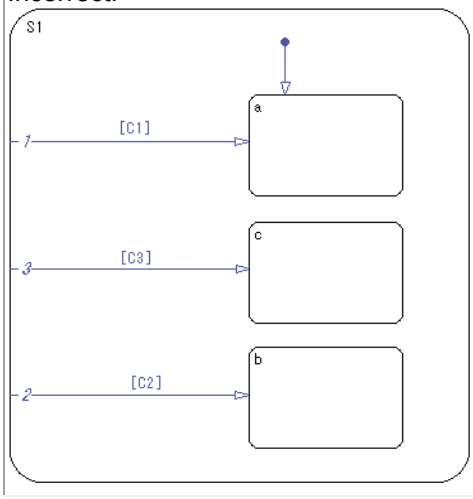




See also	MISRA AC SLSF 043F
Last Change	V4.0

5.6.4. jc_0761: Statement method when using multiple internal transitions

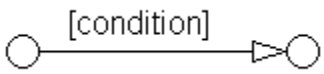
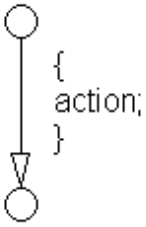
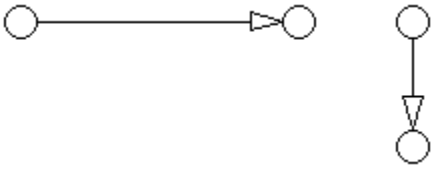
ID: Title	**jc_0761: Statement method when using multiple internal transitions**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB	All

Version	
Prerequisites	
Description	If multiple internal transitions are described out of necessity regarding an internal transition within a single state, they should be listed from top to bottom according to the order of execution of the internal transitions. Correct:  Incorrect: 
See also	
Notes	jc_0763: In the usage restrictions of multiple internal transitions, it is recommended that multiple internal transitions are not used. However, in some cases, using multiple internal transitions can prevent transition lines from crossing and simply represent state transitions. If multiple internal transitions will be used in such cases, use them in compliance with this rule.
Last Change	V4.0

5.7. Flow Chart foundation

5.7.1. db_0132: Transitions in Flow Charts

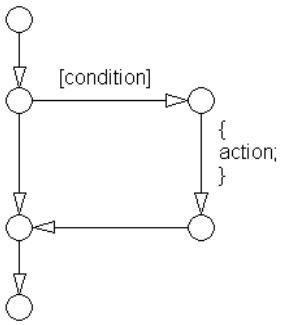
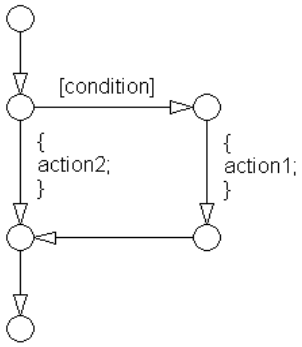
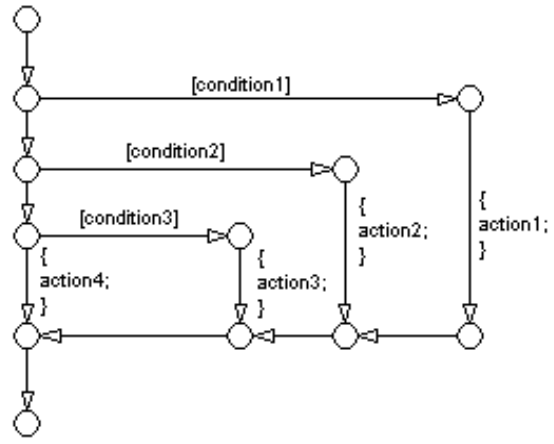
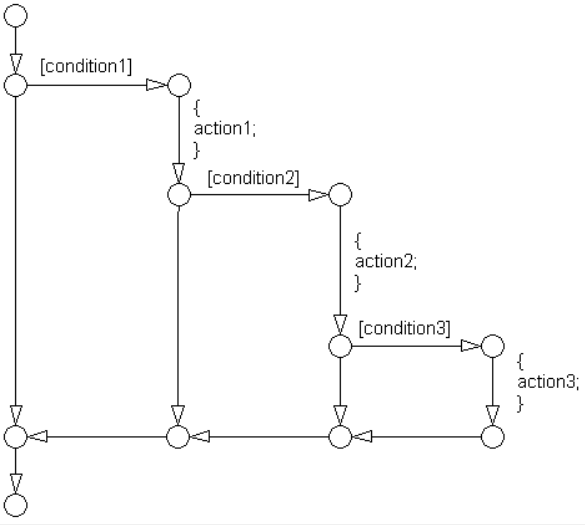
ID: Title	**db_0132: Transitions in Flow Charts**
Priority	Strongly Recommended
Scope	MAAB

MATLAB Version	All
Prerequisites	
Description	1. The following rules apply to transitions in Flow Charts: ● Conditions are drawn on the horizontal. ● Actions are drawn on the vertical. 2. Transitions labels of Flow Charts use a condition, condition action, or an empty transition. (Transition action must not be used in flow charts) Example Transition with condition:  Transition with condition action:  Empty transition:  Exception 5.7.4 db_0135: Flow Chart patterns for loop constructs
Notes	
Last Change	V4.0

5.7.2. db_0134: Flow Chart patterns for If constructs

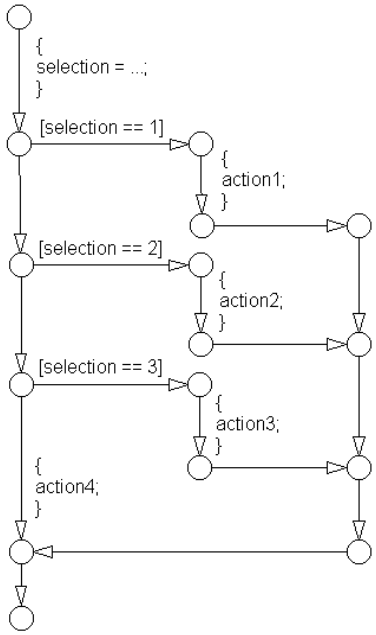
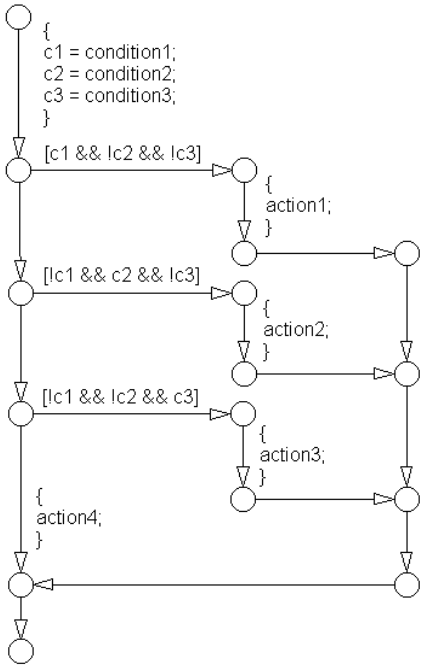
ID: Title	**db_0134: Flow Chart patterns for If constructs**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	jc_0742: Guidelines for writing Boolean operations to condition labels jc_0743: Guidelines for writing condition actions
Description	The following patterns are used for If constructs within Flow Charts:

Functionality	**Flow Chart Pattern**

	IF-THEN construct if (condition){ action; }	
	IF-THEN-ELSE construct if (condition) { action1; } else { action2; }	
	IF-THEN-ELSE-IF construct if (condition1) { action1; } else if (condition2) { action2; } else if (condition3) { action3; } else { action4; }	
	Cascade of IF-THEN construct if (condition1) { action1; if (condition2) { action2; if (condition3) { action3; } } }	
Last Change	V1.0	

5.7.3. db_0159: Flowchart patterns for case constructs

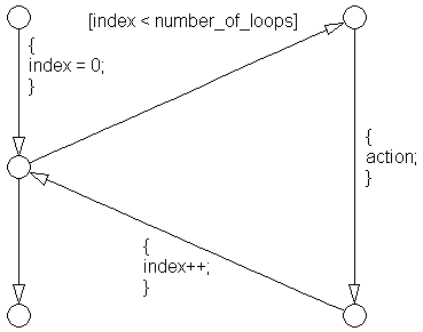
ID: Title	**db_0159: Flowchart patterns for case constructs**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	jc_0742: Guidelines for writing Boolean operations to condition labels jc_0743: Guidelines for writing condition actions
Description	The following patterns must be used for case constructs within Flow Charts:

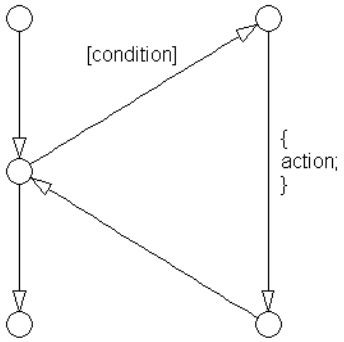
Functionality	Flow Chart Pattern
CASE construct with exclusive selection selection = ...; switch (selection) { case 1: action1; break; case 2: action2; break; case 3: action3; break; default: action4; }	
CASE construct with exclusive conditions c1 = condition1; c2 = condition2; c3 = condition3; if (c1 && ! c2 && ! c3) { action1; } elseif (! c1 && c2 && ! c3) { action2; } elseif (! c1 && ! c2 && c3) { action3; } else { action4; }	

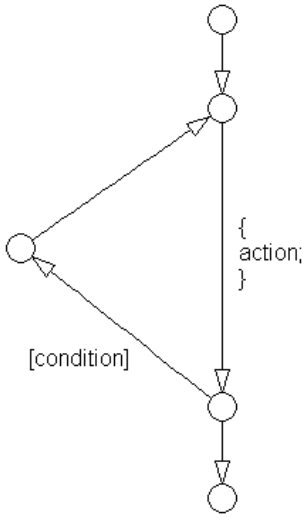
Last Change	V1.0

5.7.4. db_0135: Flow Chart patterns for loop constructs

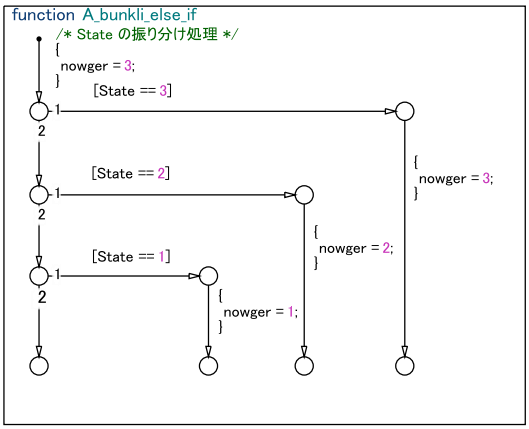
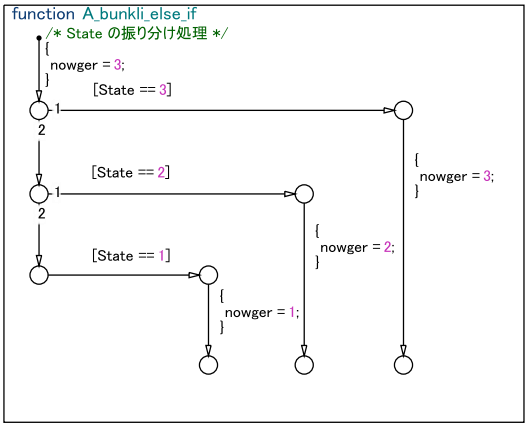
<table>
<tr><td>ID: Title</td><td>db_0135: Flow Chart patterns for loop constructs</td></tr>
<tr><td>Priority</td><td>Recommended</td></tr>
<tr><td>Scope</td><td>MAAB</td></tr>
<tr><td>MATLAB Version</td><td>All</td></tr>
<tr><td>Prerequisites</td><td>jc_0742: Guidelines for writing Boolean operations in condition labels
jc_0743: Guidelines for writing condition actions</td></tr>
<tr><td>Description</td><td>The following patterns must be used to create Loops within Flow Charts.
<table>
<tr><th>Functionality</th><th>Flow Chart Pattern</th></tr>
<tr><td>FOR LOOP construct

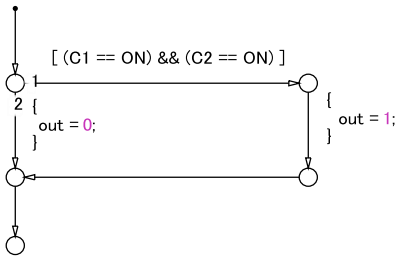
for (index=0;index<number_of_loops;index++) {
  action;
}</td><td> 
</td></tr>
<tr><td>WHILE LOOP construct

while (condition) {
  action;
}</td><td> 
</td></tr>
<tr><td>DO WHILE LOOP construct

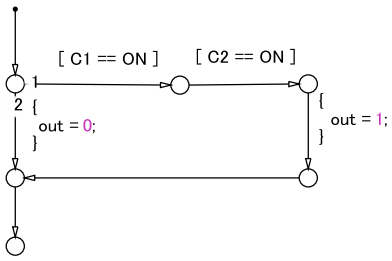
do {
  action;
}
while (condition);</td><td> 
</td></tr>
</table></td></tr>
<tr><td>Last Change</td><td>V1.0</td></tr>
</table>

5.7.5. jc_0773: Unconditional transition of a flow chart

ID: Title	**jc_0773: Unconditional transition of a flow chart**
Priority	Strongly Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	
Description	● All the flow charts, a graphical function, "Unconditional transition " when not fulfilling conditions is required for it. It is for preventing backtracking. ● The priority of unconditional transition is set as the last. In order to prevent backtrack, all flowchart and graphycal function provide unconditional taransition which will not meet all conditions. (Use complex conditions in one place) ● Priority of unconditional transition will be set at the very end. Correct:  Incorrect: It does not have transition line of "Unconditional transition".  Correct: When a complex condition is summarized to one.



Incorrect:
There are no unconditional transition in a central junction.



When the flow chart which outputs one by C1 == ON && C2 == ON is drawn as mentioned above.
The high skill engineer, C1 performs out=0, when C2 is OFF in ON, and he understands that processing is performed to a termination.
However, it is not easy to understand the course to a termination at a glance.
In this case, in order to avoid misunderstanding, following either is coped with and it leads to a termination.
"A complex condition is bundled to one."
"Unconditional changes are certainly prepared."

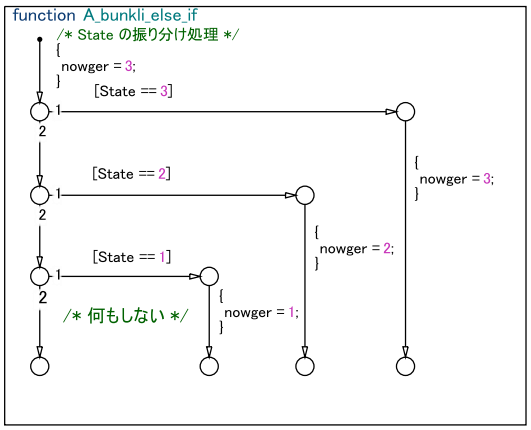
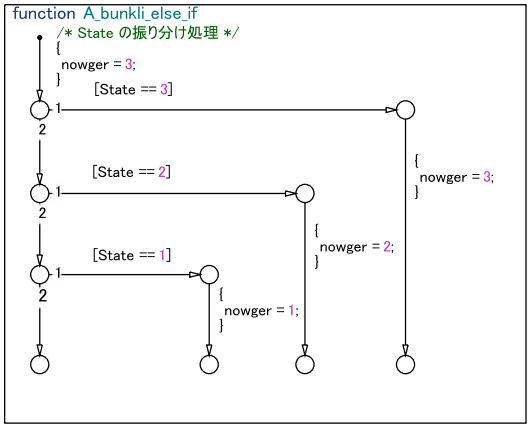
If you can understand Stateflow semantics,
When you drarw above chart which output 1 with C1 == ON && C2 == ON, you will know that the process is ran to the end when execute out=0. However, it is difficult to understand the path to the end at a glance.
In order to avoid misunderstanding, connecting to the end by using the methods below.

- Bundle complex conditions.
- Provide unconditional transition

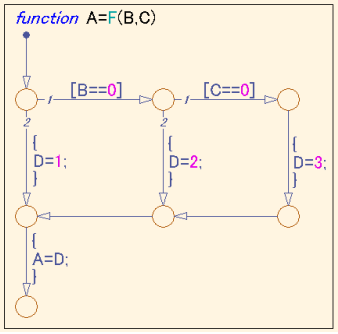
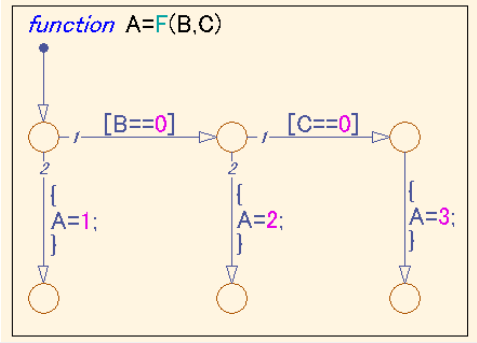
Notes	This is a backtracking prevention rule of a flow chart. Expression a flow chart "suspends processing in the middle of a junction in the case of condition disagreement" unlike condition changes is not used. Wire connection is carried out using unconditional transition array so that the last of processing may become clear, so that it may certainly flow to a termination. This is the rule to prevent backtrack of flowchart. This is different from conditional transition, which will not use the expression like “Stop the process in the middle of the junction in the case of mismatch conditions.” Wired it so that the end of the process becomes clear by using undonsitional transition line to flow to the end.
See also	
Last Change	V4.00

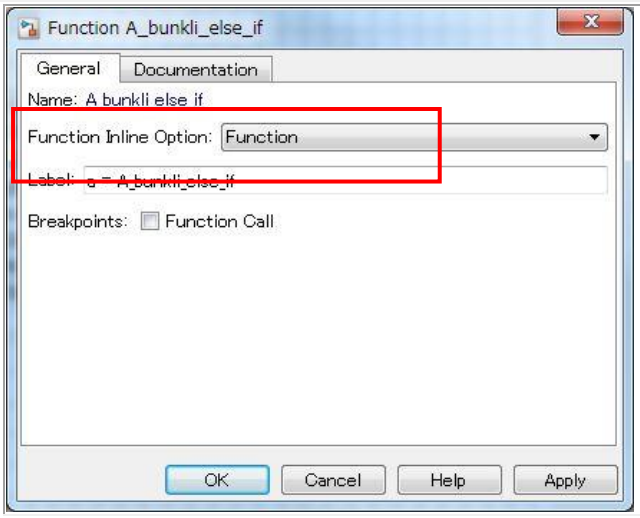
5.8. Flow Chart details

5.8.1. jc_0774: Comments on unconditional transition which has no process

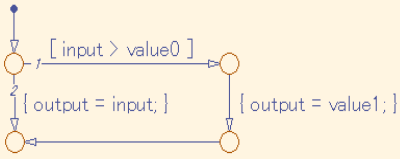
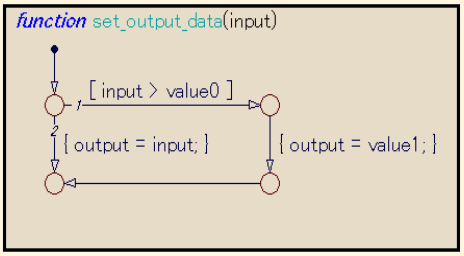
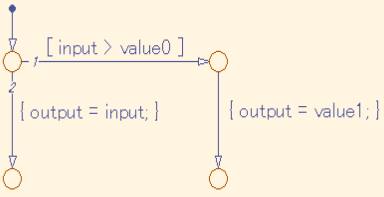
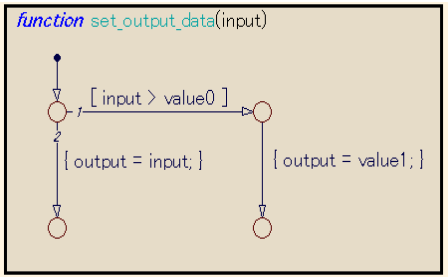
ID: Title	**jc_0774: Comments on unconditional transition which has no process**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	jc_0773: Unconditional transition of a flow chart
Description	When unconditional transition which has no action is used as exceptional processing in case no condition is met, comments must be described to show intentionally no process is described Correct: If there is unconditional transition which has no process, comment must be described.  Incorrect: Although there is unconditional transition, no comment is described. It is difficult to understand whether unconditional transition was intentionally described or description of conditions and actions was forgotten. 
Notes	
See also	
Last Change	V4.0

5.8.2. jc_0511: Setting the return value from a graphical function

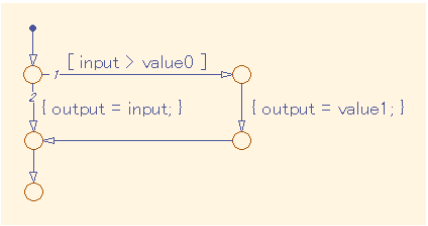
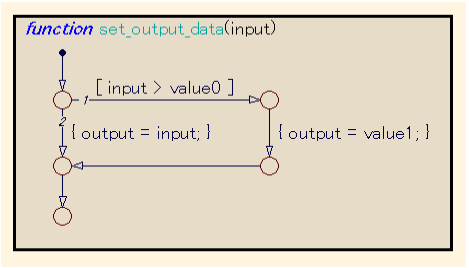
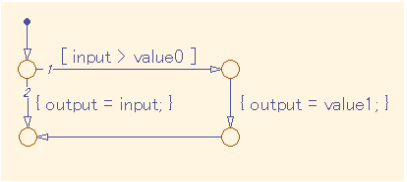
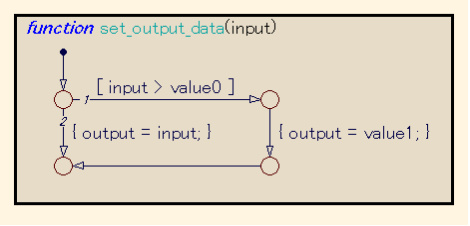
ID: Title	**jc_0511: Setting the return value from a graphical function**
Priority	Recommended
Scope	JMAAB
MATLAB Version	R2008a and later
Prerequisites	db_0134: Flow Chart patterns for If constructs
Description	The return value from a graphical function must be set in only one place. Correct: Return value A is set in one place.  Incorrect: Return value A is set in multiple places. 
Notes	Regarding R2007b and earlier, this rule has influence to code generation. If incorrect pattern is used, multiple return sentences are generated. This is violation to MISCA-C 1998 rule 82 and MISRA-C2004 rule 14.7. If earlier versions are adopted, please operate this rule as Mandatory as same as Ver2.0. In R2008a and later, C codes which has no violation to MISRA rules are generated. However, for the purpose of getting efficient code, in some cases, it is necessary that function setting of graphical functions are set not to "auto","inline" but to "function". In current versions of MATLAB, please operate this rule in order to unify appearances of graphical functions.

	
Last Change	V4.0

5.8.3. jc_0775: Number of terminal junctions in Flow Charts

ID: Title	**jc_0775: Number of terminal junctions in Flow Charts**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	db_0134: Flow Chart patterns for If constructs
Description	A unique terminal junction must exist in all graphical functions and Flow Charts described in states. Correct:   Incorrect:  
See also	MISRA AC SLSF 053J
Last Change	V4.0

5.8.4. jc_0776: Number of inputs to the terminal junction of Flow Charts

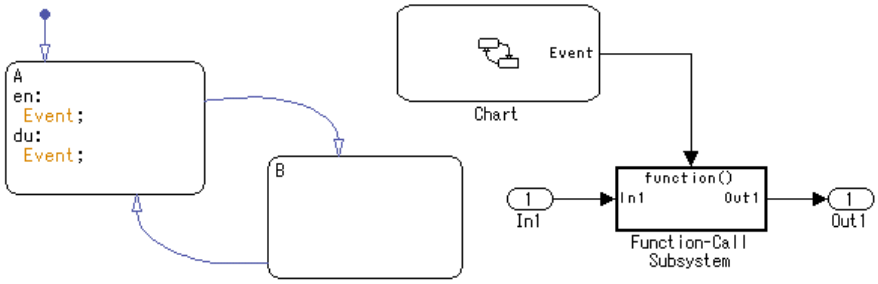
ID: Title	**jc_0776: Number of inputs to the terminal junction of Flow Charts**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	jc_0775: Number of terminal junctions in Flow Charts
Description	In all graphical functions and Flow Charts described in states, the number of transition lines inputted in terminal junctions within all Flow Charts and graphical functions should be one. Correct:   Incorrect:   The purpose of this rule is to explicitly indicate the point of completion.
See also	MISRA AC SLSF 053K
Last Change	V4.0

5.9. Event

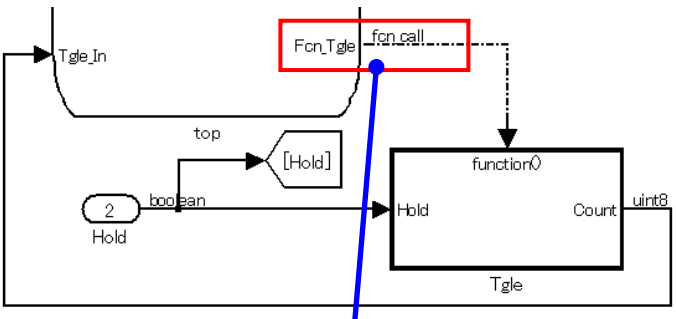
5.9.1. db_0126: Scope of events

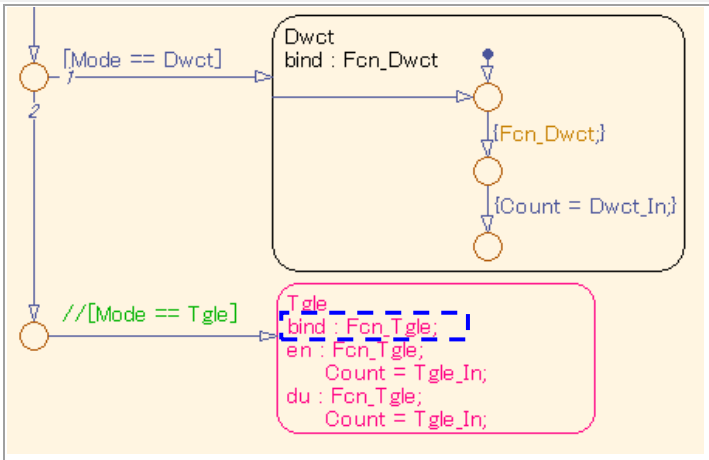
ID: Title	**db_0126: Scope of events**
Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisites	
Description	The following rules apply to events in Stateflow: ● All events of a Chart must be defined on the chart level or lower. ● There is no event on the machine level (that is, there is no interaction with local events between different charts).
Notes	It becomes the compilation error after R2009b.
Last Change	V4.0

5.9.2. jc_0780: Usage restrictions of events

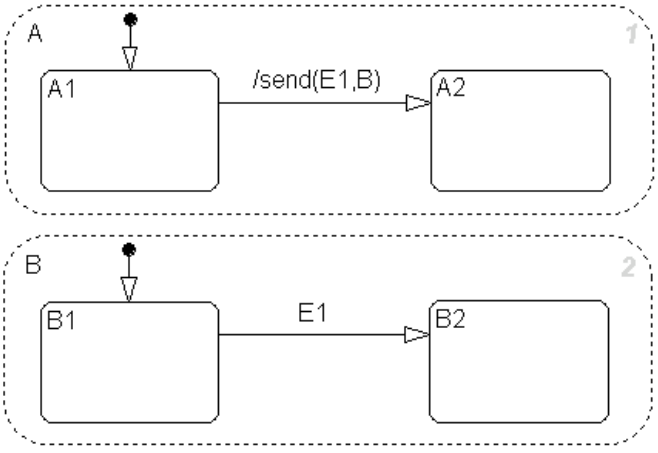
ID: Title	**jc_0780: Usage restrictions of events**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	db_0126: Guidelines for defining events
Description	Events should not be used for anything other than calls in the Function Call Subsystem. (State transitions by events should not be used.) Correct: 
Notes	If state transitions by events are used without fully understanding their operation, there are cases in which processing is unintentionally performed by recursive function and processing is performed twice in one cycle.
See also	MISRA AC SLSF 047A
Last Change	V4.0

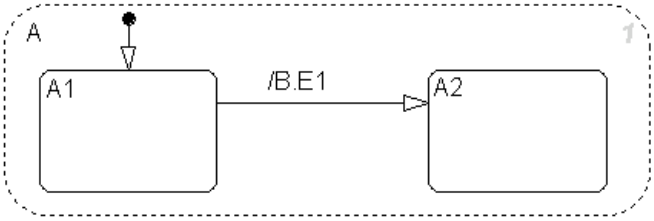
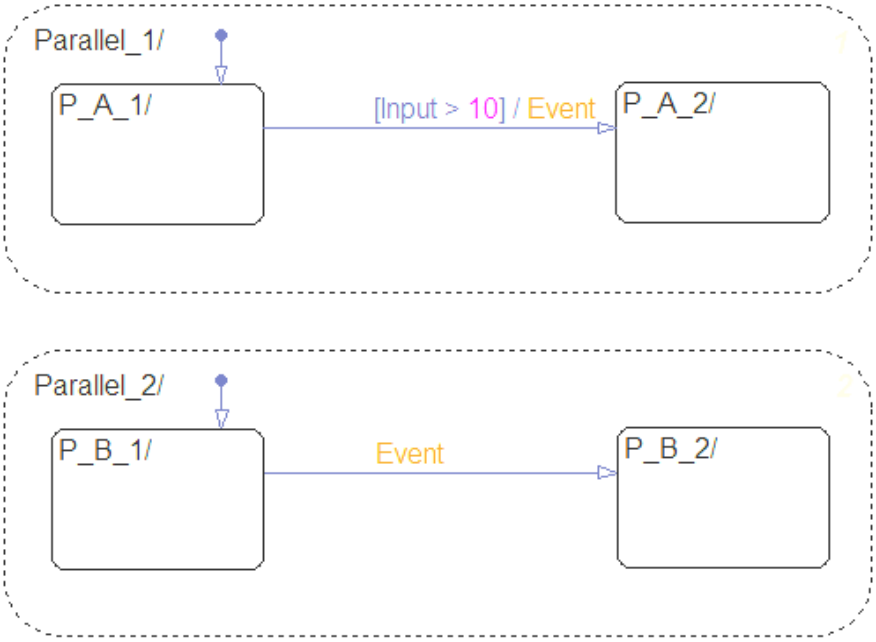
5.9.3. jc_0781: Function Call from Stateflow

ID: Title	**jc_0781: Function Call from Stateflow**
Priority	Recommended
Scope	JMAAB
MATLAB Version	All
Prerequisites	db_0126: Guidelines for defining events jc_0780: Usage restrictions of events na_0006: Guidelines for mixed use of Simulink and Stateflow
Description	If the "state exists" within the Function Call Subsystem of the call target and a "reset" of the state is required when the state of the caller becomes inactive, a bind action should be described by the caller. 

	
See also	
Last Change	V4.0

5.9.4. jm_0012: Event broadcasts

ID: Title	**jm_0012: Event broadcasts**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	All
Prerequisites	**db_0126: Guidelines for defining events**
Description	The following rules apply to event broadcasts in Stateflow: ● Directed event broadcasts are the only type of event broadcasts allowed. ● The send syntax or qualified event names are used to direct the event to a particular state. ● Multiple send statements should be used to direct an event to more than one state. Correct: Example using the send syntax:  Correct: Example using qualified event names:

	 Incorrect: Use of a non-directed event 
Last Change	V2.2

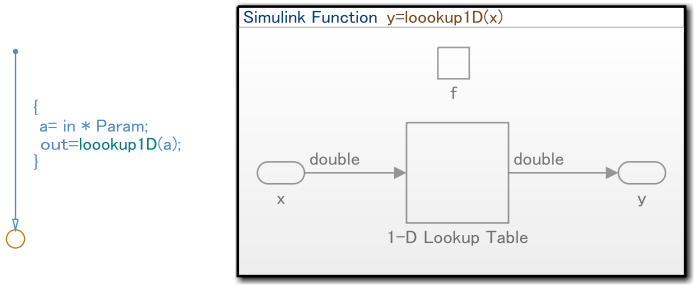
5.10. Functions within Stateflow

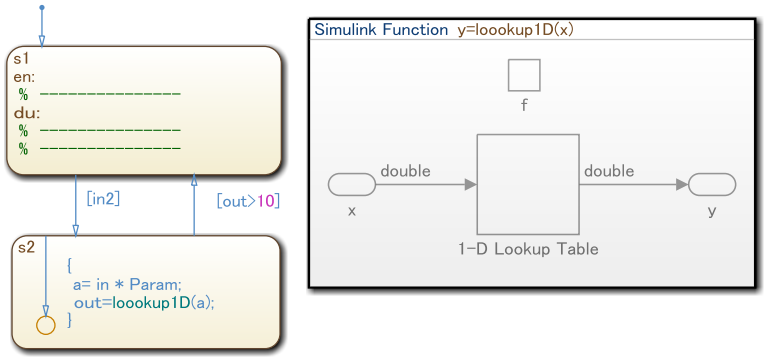
5.10.1. na_0041: Selection of function type

ID: Title	**na_0041: Selection of function type**
Priority	Recommended
Scope	MAAB
MATLAB Version	2010b and later
Prerequisites	
Description	The type of functions to be used should be selected depending on the required processing. ● Graphical functions ➢ If / then / else logic

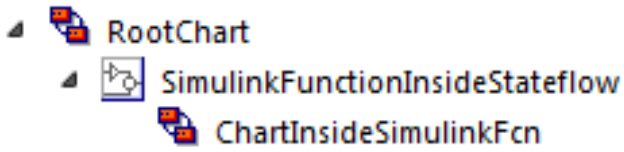
	• Simulink functions ➢ Transfer functions ➢ Integrators ➢ Table look-ups • MATLAB functions ➢ Complex equations ➢ If / then / else logic
Notes	Stateflow supports the following three types of functions: • Graphical functions • Simulink functions • MATLAB functions
Last Change	V3.0

5.10.2. na_0042: Location of functions

ID: Title	**na_0042: Location of functions**
Priority	Recommended
Scope	MAAB
MATLAB Version	2010b and later
Prerequisites	
Description	When deciding whether to embed Simulink functions inside a Stateflow chart, the following conditions make embedding the preferred option. If the Simulink functions: • Use only local Chart data or • Use a mixture of local Chart data and inputs from Simulink or • Are called from multiple locations within the chart or • Are not called every time step Incorrect  correct

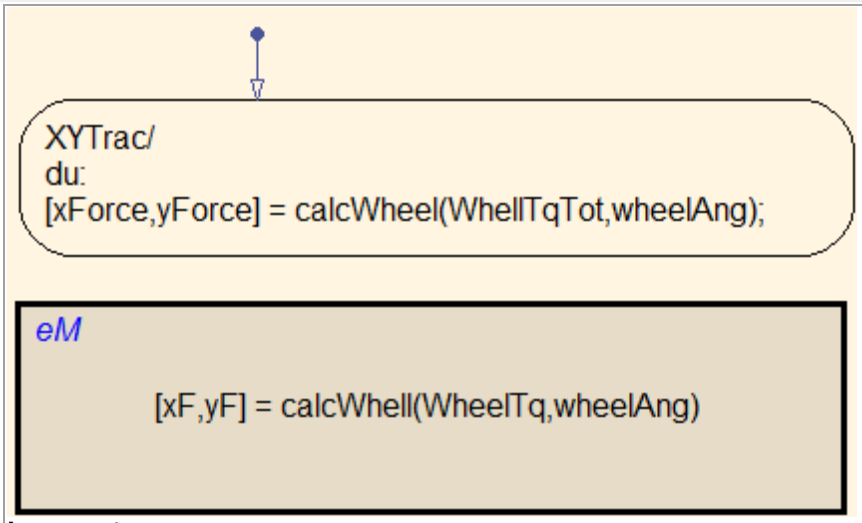
	
Last Change	V3.0

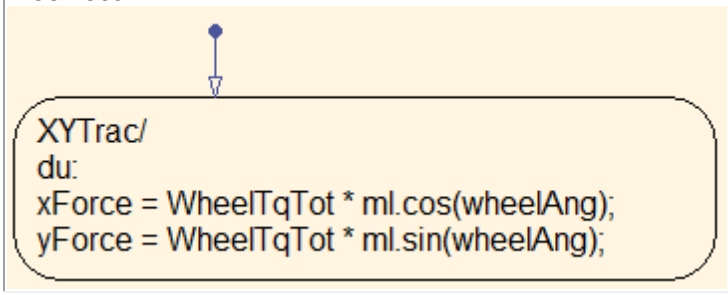
5.10.3. na_0039: Use of Simulink in Stateflow charts

ID: Title	**na_0039: Use of Simulink in Stateflow charts**
Priority	Recommended
Scope	MAAB
MATLAB Version	2010b and later
Prerequisites	
Description	The use of Stateflow charts is prohibited in Simulink functions that are included in Stateflow charts. Incorrect: 
Last Change	V3.0

5.10.4. db_0127: MATLAB commands in Stateflow

ID: Title	**db_0127: MATLAB commands in Stateflow**
Priority	Mandatory
Scope	MAAB
MATLAB Version	All
Prerequisites	
Description	Do not use the .ml syntax in Stateflow charts. Individual companies should decide on the use of MATLAB functions. If they are permitted, then MATLAB command should only be accessed through the MATLAB function . **Correct:**

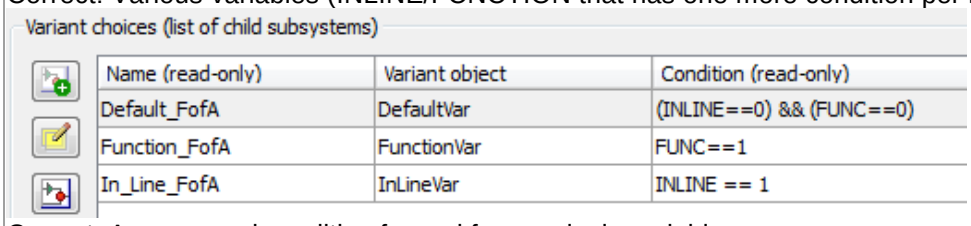
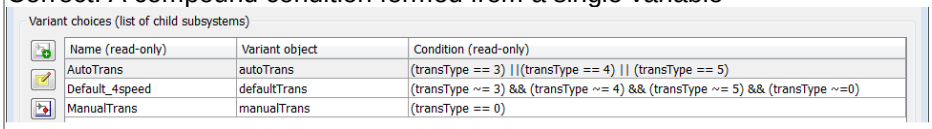
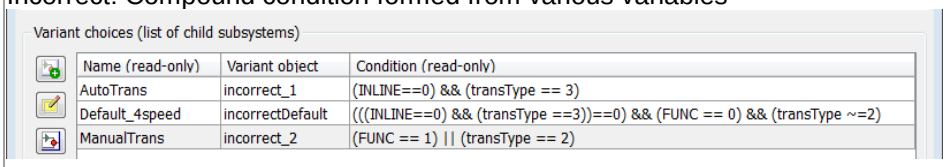
<table>
<tr><td></td><td> 

Incorrect:
 
</td></tr>
<tr><td>Notes</td><td>Code generation supports a limited subset of the MATLAB functions.
For a complete list of the support function, see the "MathWorks®" documentation.
Corresponding functions are described in the following two places.
・Functions Supported for Code Generation — Alphabetical List
(Functions Supported for Code Generation — Alphabetical List)
http://www.mathworks.com/help/simulink/ug/functions-supported-for-code-generation-alphabetical-list.html
・Functions Supported for Code Generation — Categorical List
(Functions Supported for Code Generation — Categorical List)
http://www.mathworks.com/help/simulink/ug/functions-supported-for-code-generation--categorical-list.html</td></tr>
<tr><td>Last Change</td><td>V2.2</td></tr>
</table>

6. Miscellaneous: Variants, enumerated type, MATLAB functions

6.1. Variant Subsystem

6.1.1. na_0037: Use of single variable variant conditionals

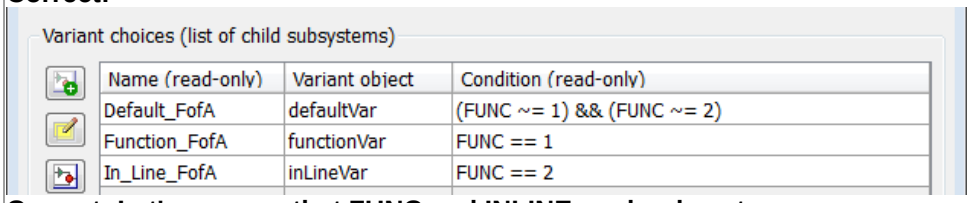
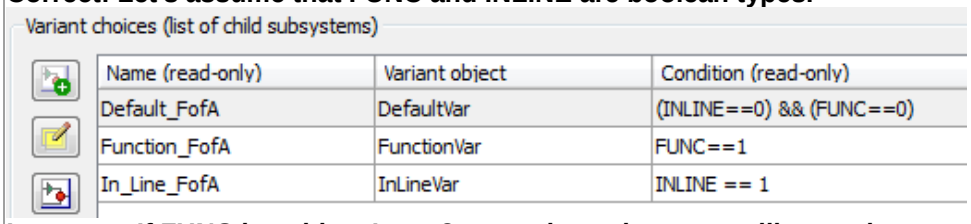
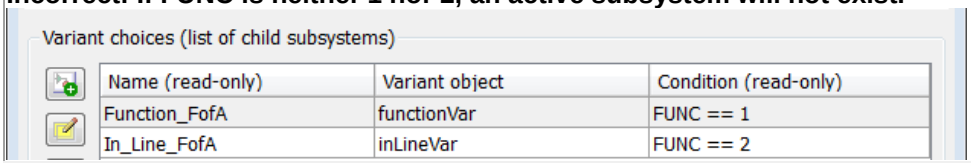
ID: Title	**na_0037: Use of single variable variant conditionals**
Priority	Recommended
Scope	MAAB
MATLAB Versions	ALL
Prerequisite	
Description	Variant condition equations must be composed from a compound condition formed from a single variable or a single condition formed from multiple variables. The provided variant is the exception to the second regulation. Correct: Various variables (INLINE/FCNCTION that has one more condition per line)  Variant choices (list of child subsystems) Name (read-only) / Variant object / Condition (read-only) Default_FofA / DefaultVar / (INLINE==0) && (FUNC==0) Function_FofA / FunctionVar / FUNC==1 In_Line_FofA / InLineVar / INLINE == 1 Correct: A compound condition formed from a single variable  Variant choices (list of child subsystems) Name (read-only) / Variant object / Condition (read-only) AutoTrans / autoTrans / (transType == 3) ||(transType == 4) || (transType == 5) Default_4speed / defaultTrans / (transType ~= 3) && (transType ~= 4) && (transType ~= 5) && (transType ~=0) ManualTrans / manualTrans / (transType == 0) Incorrect: Compound condition formed from various variables  Variant choices (list of child subsystems) Name (read-only) / Variant object / Condition (read-only) AutoTrans / incorrect_1 / (INLINE==0) && (transType == 3) Default_4speed / incorrectDefault / (((INLINE==0) && (transType ==3))==0) && (FUNC == 0) && (transType ~=2) ManualTrans / incorrect_2 / (FUNC == 1) || (transType == 2)
Notes	The usage of enumerated type variables is recommended in a condition equation. This example used numerical values in the screenshot to increase the readability.
Related	
Last Change	V3.0

6.1.2. na_0020: Number of inputs to variant subsystems

ID: Title	**na_0020: Number of inputs to variant subsystems**
Priority	Mandatory
Scope	MAAB
MATLAB Version	R2013b and earlier
Prerequisite	db_0081: Unconnected signals, block inputs and block outputs
Description	In Simulink, the same number needs to be inputted into Model Variants and Variant Subsystem that will be used in the variant system. However, this does not necessary mean that the variant subsytem will use all the input. Please connect the unused input with Terminator blocks to conduct termination processing.

Notes	Model Variants: Includes a model into another model as a block. Variant Subsytem: Represents subsytem that has several subsystems. A new function was added by R2014a. Even if the number of the ports is different, it is available.
See Also	
Last Change	V4.0

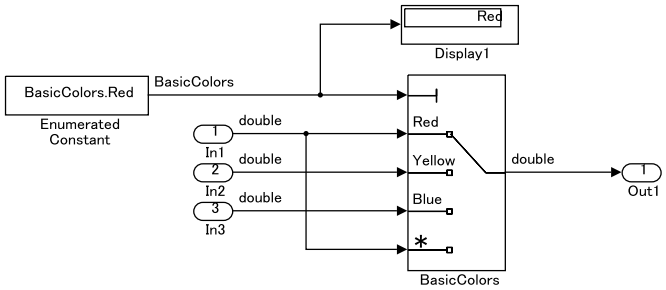
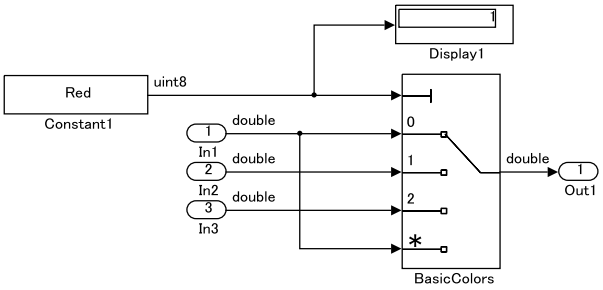
6.1.3. na_0036: Default variant

ID: Title	**na_0036: Default variant**
Priority	**Recommended**
Scope	**NAMAAB**
MATLAB Version	**ALL**
Prerequisite	**na_0037: Use of single variable variant conditionals**
Description	Model Variants and Variant Subsytem **are all constructed so that one subsytem will always be selected.** This can be achieved with one of the following methods. ● A default variant is used. ● The condition will be set so that all values that conditional variables may take will be covered. For example, a condition will be set for a situation in which the boolean-type variable's value is true and when it is false. **Correct:**  Variant choices (list of child subsystems) Name (read-only) | Variant object | Condition (read-only) Default_FofA | defaultVar | (FUNC ~= 1) && (FUNC ~= 2) Function_FofA | functionVar | FUNC == 1 In_Line_FofA | inLineVar | FUNC == 2 **Correct: Let's assume that FUNC and INLINE are boolean types.**  Variant choices (list of child subsystems) Name (read-only) | Variant object | Condition (read-only) Default_FofA | DefaultVar | (INLINE==0) && (FUNC==0) Function_FofA | FunctionVar | FUNC==1 In_Line_FofA | InLineVar | INLINE == 1 **Incorrect: If FUNC is neither 1 nor 2, an active subsystem will not exist.**  Variant choices (list of child subsystems) Name (read-only) | Variant object | Condition (read-only) Function_FofA | functionVar | FUNC == 1 In_Line_FofA | inLineVar | FUNC == 2
Notes	
Last Change	**V3.0**

6.2. Enumerated type data

6.2.1. na_0033: Enumerated Types Usage

ID: Title	**na_0033: Enumerated Types Usage**
Priority	Recommended

Scope	MAAB
MATLAB Version	R2010b and later
Prerequisite	**na_0002: Basic logical operation and the appropriate implementation of arithmetic operations**
Description	Signals and parameters serve as a finite set of integer values. If the values of these sequences correspond to a group formed from items with names, use the data of an enumerate type. Example: Usage example of red, yellow, and blue in a traffic light. Correct:  Incorrect:  Red is used as a regular unit 8 value.
Notes	4 byte will be used for the enumerate type in the C-code in the standard regulation.
See Also	
Last Change	V3.0

6.2.2. na_0031: Definition of default enumerated value

ID: Title	**na_0031: Definition of default enumerated value**
Priority	Recommended
Scope	MAAB
MATLAB Version	R2010b and later
Prerequisite	
Description	Default value of an enumerated type (getDefaultValue) always needs to be stipulated explicitly. 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_Violation < Simulink.IntEnumType enumeration Red(0) Yellow(1) Blue(2) end end
Notes	When the default value is not stipulated when using getDefaultValu, the text listed in the enumeration will be used as the initial value. For example, if "Yellow" is written first like in the below example, "Yellow" will be used as the initial value. enumeration Yellow(1) Red(0) Blue(2) end
Last Change	V3.0

6.3. MATLAB functions

6.3.1. na_0018: Number of nested if/else and case statement

ID: Title	na_0018: Number of nested if/else and case statement
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisite	
Description	The number of nests in if/else and case statements need to be restricted. Example: Say, the number hierarchies was up to 3 hierarchy.
See Also	Orion_jr_0002: The number of if/else and case statements block nests
Last Change	V3.0

6.3.2. na_0025: MATLAB function header

ID: Title	na_0025: MATLAB function header
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisite	
Description	MATLAB functions need to have a header that explains such functions. For example, the following types of information will be entered in a header. Item example: ・Function name ・Explanation of the function ・Prerequisite and restriction ・Modified points from the previous version ・List of input and output Implementation 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
%
% $Revision: 3.0$
% $Author: MAAB$
% $Date: July 24,2012$
%
%-----------------------------------------
```

See Also	Orion_jh_0073: eML header version
Last Change	V3.0

6.3.3. na_0034: MATLAB Function block input/output settings

ID: Title	**na_0034: MATLAB Function block input/output settings**
Priority	Strongly Recommended
Scope	NAMAAB
MATLAB Version	ALL
Prerequisite	
Description	It is required to explicitly stipulate the data type at the top of the model explorer or the function for all input and output toward MATLAB function block
Notes	
See Also	**Orion_jh_0063: Input and output setting of eML block**
Last Change	V4.0

6.3.4. na_0024: Global variable

ID: Title	**na_0024: Global variable**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisite	
Description	It is recommended to access the signal wire with common data between MATLAB functions. For example, if one side only merely consult the signal value, the connection is made by using the signal line without using the data store memory. In the following cases, it is possible to share the signal using a data store memory without connecting via signal line. ● It is required to share a specific signal, such as conducting writing updates toward the same signals within various MATLAB functions. Example: In this example, the same data store memory (ErrorFlag_DataStore) is shared between two different MATLAB functions. `function EngineFaultEvaluation(EngineData)` `%#codegen` `  global ErrorFlag_DataStore` `  if (EngineData.RPM_HIGH)` `  ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,HIGHRPMFAULT);` `  end` `  if (EngineData.RPM_LOW)` `  ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,LOWRPMFAULT);` `  end` `end`

	function WheelFaultEvaluation(WheelData) %#codegen global ErrorFlag_DataStore if (WheelData.SlipHigh) ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,WHEELSLIP); end if (WheelData.SlipHigh) ErrorFlag_DataStore = bitor(ErrorFlag_DataStore,LOWRPMFAULT); end end
See Also	**Orion_ek_0003: Global variable**
Last Change	V4.0

6.3.5. na_0022: Recommended patterns for Switch / Case statements

ID: Title	**na_0022: Recommended patterns for Switch / Case statements**
Priority	Mandatory
Scope	MAAB
MATLAB Version	ALL
Prerequisite	
Description	Switch / Case statements must use constant values for the “Case” arguments. Input variables cannot 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 ```
See Also	Orion_jh_0026: Switch/Case statement
Last Change	V3.0

6.3.6. na_0016**:** Source lines of MATLAB Functions

ID: Title	**na_0016: Source lines of MATLAB Functions**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisite	
Description	The length of MATLAB functions should be limited, with a recommended limit of 60 lines of code. This restriction applies to MATLAB Functions that reside in the Simulink block diagram and external MATLAB files with a .m extension. If sub-functions are used, they may use additional lines of code. Also limit the length of sub-functions to 60 lines of code.
See Also	Orion_im_0008: Source line of eML
Last Change	V3.0

6.3.7. na_0017: Number of called function levels

ID: Title	**na_0017: Number of called function levels**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisite	
Description	The number of levels of sub-functions should be limited, typically to 3 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.
Notes	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.

See Also	Orion_im_0009: Hierarchy number of function to be called out
Last Change	V3.0

6.3.8. na_0021: Strings

ID: Title	**na_0021: Strings**
Priority	Strongly Recommended
Scope	MAAB
MATLAB Version	ALL
Prerequisite	
Description	The use of strings is not recommended. MATLAB Functions store strings as character arrays. The arrays cannot be resized to accommodate a string value of different length, due to lack of dynamic memory allocation. Stings are not a supported data type in Simulink, so MATLAB Function blocks cannot pass the string data outside the block. For example, the following code will produce an error: *name = 'rate_error'; %this creates a 1 x 10 character array* *name = 'x_rate_error'; %this causes an error because the array size is now 1 x 12, not 1 x 10*
Notes	If the string is being used for switch / case behavior, consider using enumerated data types.
See Also	Orion_jh_0024: Character string
Last Change	V3.0

7. Basis, list of rule parameters

7.1. Basis

7.1.1. Basis category

For the basis, one or more reasons from the following reasons that the guideline recommends will be selected.

1. **Readability**
    - **Improvement of graphical understandability**
    - **Improvement of readability of functional analysis.**
    - **Prevention of connection mistake**
    - **Comments and so on**

2. Simulation and verification
    - **System to enable simulation**
    - **Easy testing**

3. Code generation
    - **Improvement of efficiency of generation code.(ROM,RAM efficiency)**
    - **Securement of robustness of a generation code**

4. Others
    - Maintainability and operatability
    - Template
    - Not correspond to basis described above (Basis is unclear)

7.1.2. List of rule basis

Rule ID	Readability	Simulation and verification are effective	Effective/efficient to built-in code generation	Others
ar_0001		○		
ar_0002			△:In the past	
jc_0241	○		○	
jc_0242	○			
jc_0201			○	
jc_0211			○	
jc_0222			○	
jc_0232		○		
jc_0231	○			
jc_0243	○		○	
jc_0244	○		○	
jc_0245	○		○	
jc_0246	○		○	
jc_0247	○			
na_0035	○			
jc_0251			○	
na_0014		△:In the past		
na_0006	○		○	
na_0007		○		
db_0143	○			
db_0144	○			
na_0004	○			
db_0043	○			

db_0042	○			
jm_0002	○			
db_0142	○			
jc_0061	○			
db_0140	○			
db_0032	○			
db_0141	○			
jc_0110	○			
jc_0111	○			
jc_0653	○	○		
jc_0171	○			
jc_0602	○			
db_0146	○			
jc_0281	○			
jc_0603	○			
jc_0604	○			
na_0010		○		
na_0008				
na_0009				
na_0005				
jc_0082				
jc_0083				
db_0097	○			
db_0081		○		
na_0003	○			
na_0002		○		
jm_0001			○	
hd_0001			○	
na_0011	○			
jc_0141		○		
jc_0121	○			
jc_0610	○			
jc_0611			○	
jc_0131	○			
jc_0161	○	○		
jc_0621	○			
jc_0011			○	
jc_0629			○	
jc_0622	○			
jc_0626			○	
jc_0627		○	○	
jc_0628	○			
jc_0650		○		
jc_0630	○	○		
jc_0631			○	
jc_0632	○	○		
jc_0625	○			
jc_0640		○	○	
db_0112			○	

db_0110			○	
jc_0645			○	
jc_0641				Improvement of maintainability and operatability
jc_0642			○	
jc_0643			○	
jc_0644				Improvement of maintainability and operatability
db_0114				Template
db_0115				Template
db_0116				Template
db_0117				Template
na_0012				Not correspond to basis described above
na_0028				Not correspond to basis described above
jc_0658	○	○	○	
jc_0623	○			
jc_0624	○		○	
jc_0651	○		○	
jc_0652	○			
jc_0659	○	○		
jc_0656	○			
jc_0657	○		○	
db_0123	○			
jc_0700				Improvement of maintainability and operatability
db_0122		○		
db_0125				Improvement of maintainability and operatability
jc_0701				Improvement of maintainability and operatability
jc_0702			○	
jm_0011				○
db_0129	○			
db_0137	○		○	
jc_0711		○	○	
jc_0531	○	○	○	
jc_0712	○	○	○	
na_0038	○			
na_0040	○			
jc_0720	○			
jc_0721	○		○	

jc_0722				Improvement of maintainability and operatability
jc_0723	○	○		
jc_0730		○	○	
jc_0731	○			
jc_0732	○			
jc_0733	○	○		
jc_0734	○	○		
jc_0740	○	○		
jc_0501	○			
jc_0735		○		
jc_0736	○			
jc_0737	○			
jc_0738	○			
jc_0739	○			
jc_0741	○	○		
jc_0742	○			
jc_0770	○			
jc_0771	○		○	
jc_0772	○			
jc_0752	○			
jc_0743	○			
jc_0750	○			
jc_0751		○	○	
jc_0754	○			
jc_0753	○			
db_0151	○			
na_0013			○	
jc_0481		○	○	
na_0001	○		○	
jc_0655	○			
jc_0451		○	○	
jc_0755	○			
jc_0756	○			
jc_0757	○			
jc_0491	○			
jc_0521	○			
jc_0760	○			
jc_0762	○			
jc_0763	○			
jc_0761	○			
db_0132	○			
db_0134				Template
db_0159				Template
db_0135				Template
jc_0773	○			
jc_0774	○			
jc_0511	○		△	

jc_0775	○			
jc_0776	○			
db_0126	△	△		
jc_0780	○	○		
jc_0781		○		
jm_0012	○			
na_0041	○			
na_0042	○			
na_0039	○	○		
db_0127			○	
na_0037	○			
na_0020		○		
na_0036	○			
na_0033	○			
na_0031	○		○	
na_0018	○			
na_0025	○			
na_0034				Not correspond to basis described above
na_0024	○			
na_0022		○		
na_0016	○			
na_0017	○			
na_0021	○	○	○	

7.2. Selectable parameters of each rule

7.2.1. Interpretation

In several rules, it is clearly described that is is selectable. However not all rules include that description. Regarding the others, there is no need to accord completely to description. This guideline provides templates for practical use of rules in projects. Numeric values and block types described in guidelines are not absolute. They need adaptation to characteristics of each project. In this section, least choices which must be decided based on characteristics of each project are described. As other elements, development processes of each project, conditions of controlling object, average of skill levels of relating engineers should be taken into comprehensive consideration. Appropriate operation based on understanding of what guidelines really mean is expected.

7.2.2. List of rule parameters

This list does not completely include all selectable parameters.

Rule ID	Parameters
ar_0001	Extensions which are subject to this rule is decided. In case limite to MATLAB related files, following extensions are subject to this rule. {m,p,mdl,slx,fig,c,h,mexw64,mexw32} Current version does not use dll. In case all files are subject to this rule, kinds of extensions should not be limited.
ar_0002	
jc_0241	Total number of characters

jc_0242	Total number of characters
jc_0201	
jc_0211	
jc_0222	
jc_0232	
jc_0231	Kinds of subsystems Expansion to function declaration.
jc_0243	Total number of characters
jc_0244	Total number of characters
jc_0245	Total number of characters
jc_0246	Total number of characters
jc_0247	Total number of characters
na_0035	All of naming conventions
jc_0251	
na_0014	Places in which using local language is inhibited. Processes adoption.
na_0006	
na_0007	
db_0143	● The list of blocks which are allowed to use on all layers. ● The list of blocks which is used depends on layers. ● Definitions of layers
db_0144	
na_0004	The type of options and setting values which should be selected.
db_0043	Kind of the font, size and style. Simulink: Standard setting should be decided for each of block, line and annotation. Stateflow: Standard setting should be decided for each of state label and transition label.
db_0042	
jm_0002	Blocks which are subject to this rule. And their sizes. Regarding block sizes, tolerances should be decided.
db_0142	
jc_0061	For each process subject to this rule, following lists should be decided. ● The list of block types whose names are always displayed ● The list of block types whose names are always undisplayed. ● The list of block types whose names are selectable to be displayed or undisplayed
db_0140	For each process subject to this rule, following lists should be decided. ● Block types subject to this rule, options to be displayed and conditions to display options. ● How to display and displaying characters.
db_0032	
db_0141	
jc_0110	Block types which are allowed to be rotated.
jc_0111	
jc_0653	
jc_0171	
jc_0602	
db_0146	Regarding detailed position of blocks, it is selected from the following

	patterns. ● Anywhere of top portion ● Rightside of top portion ● Center of top portion ● Leftside of top portion In case model information is described according to jc_0603, relative position of conditional input blocks and them should be clarified. Positions of following blocks also should be decided. ● For Each ● For Iterator
jc_0281	Which of block name or subsystem name inherit names of blocks
jc_0603	Decide the kind of the block which is used for model description. ● Annotation ● ModelInfo ● Both can be used Detail of position should be decided. Examples: ● The most upper left ● Anywhere of top portion ● Right-side of the whole ● Center of the whole ● Left-side of the whole In case db_0146 is also applied, relative position of conditional input blocks and model informations should be decided. Examples: ● Horizontally same position ● The upper position than conditional input blocks The keyword string should be decided. 例: ● Prerequisite ● Outline ● Function
jc_0604	Blocks which are allowed to set block shading.
na_0010	
na_0008	
na_0009	
na_0005	Which of jc_0082 or jc_0083 is adopted.
jc_0082	
jc_0083	
db_0097	
db_0081	How to enable distinguishment of automatically added blocks and intentionally added ones should be decided
na_0003	
na_0002	Block types are registered to following lists.. ● List of block types that are awaiting logical values. ● List of block types that are awaiting numerical values.
jm_0001	Prohibited block types
hd_0001	Prohibited block types

na_0011	
jc_0141	
jc_0121	
jc_0610	
jc_0611	
jc_0131	
jc_0161	
jc_0621	Which of the Logical Operator block icon shape "square" or "characteristics" is adopted.
jc_0011	
jc_0629	
jc_0622	
jc_0626	
jc_0627	
jc_0628	
jc_0650	
jc_0630	
jc_0631	
jc_0632	
jc_0625	Unified rule for initial value is decided.
jc_0640	
db_0112	Which of 0 based indexing or 1 based indexing is adopted.
db_0110	
jc_0645	
jc_0641	
jc_0642	
jc_0643	
jc_0644	
db_0114	
db_0115	
db_0116	
db_0117	
na_0012	
na_0028	The nest level of switch blocks. Total nest level?
jc_0658	
jc_0623	
jc_0624	
jc_0651	Kinds of blocks that are used for Cast. How to describe Cast.
jc_0652	
jc_0659	
jc_0656	
jc_0657	Whether comments are described or not. In case comments are described, contents and positions should be decided.
db_0123	
jc_0700	
db_0122	

db_0125	
jc_0701	
jc_0702	
jm_0011	
db_0129	
db_0137	
jc_0711	
jc_0531	
jc_0712	
na_0038	The maximum number of layers within a single viewer.
na_0040	The maximum number of layers within a single viewer.
jc_0720	
jc_0721	
jc_0722	
jc_0723	
jc_0730	
jc_0731	
jc_0732	
jc_0733	
jc_0734	
jc_0740	
jc_0501	
jc_0735	
jc_0736	
jc_0737	
jc_0738	
jc_0739	
jc_0741	
jc_0742	The number of conditions written in a line.(An example number is 3) In case of multiple lines, the position of operators. (They are written on start of lines or end of lines.)
jc_0770	Positions of conditions and actions in flow chart. ● Near the starting point of transitions ● Near the center of transitions
jc_0771	It should be decided that comments are written above lines or written below lines.
jc_0772	
jc_0752	
jc_0743	
jc_0750	
jc_0751	
jc_0754	
jc_0753	
db_0151	
na_0013	
jc_0481	
na_0001	
jc_0655	Which of "~" or "!" is used as negation. "!" is recommended.

jc_0451	
jc_0755	
jc_0756	
jc_0757	
jc_0491	
jc_0521	
jc_0760	
jc_0762	
jc_0763	
jc_0761	
db_0132	
db_0134	
db_0159	
db_0135	
jc_0773	
jc_0774	
jc_0511	
jc_0775	
jc_0776	
db_0126	
jc_0780	
jc_0781	
jm_0012	
na_0041	
na_0042	
na_0039	
db_0127	
na_0037	
na_0020	
na_0036	
na_0033	
na_0031	
na_0018	
na_0025	
na_0034	
na_0024	
na_0022	
na_0016	Number of lines in MATLAB function is 60. It should be decided whether comments are also counted or only execution lines are counted.
na_0017	Number of maximum layers.
na_0021	

Common

- Let the masked inside be targeted search?
- Is the kind of function setup by which atomic was carried out limited?
- Please determine the subsystem classified into annotation, and the kind of S-function.
  For example, is the following kind classified into annotation?
  - Do an input and the block without an output port correspond?
  - What kind of block type name corresponds?

- What kind of mask type name is applicable striped soot?

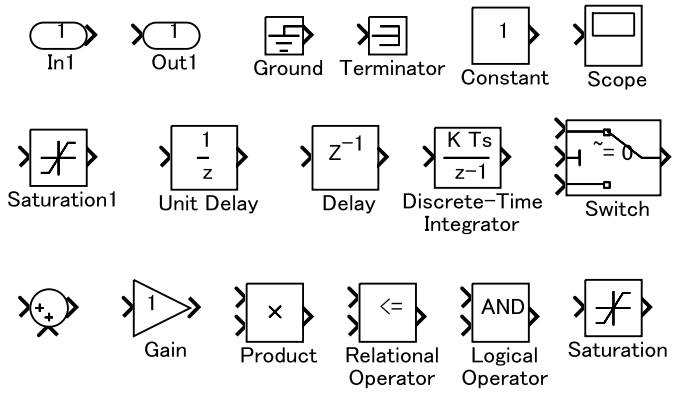
8. Terminology/supplementary explanation

JMAAB's own supplementary information not published in MAAB guideline (English) will be published here. This resource material includes content that requires supplement particular to Japan. Although the Help section in MATLAB has everything in the English translation, the Japanese Help section does not have all. Therefore, there are various sections that need to be explained just for Japan. This chapter added its own supplementary explanation on items that should be originally be read minutely on Help Definitions of terminologies used in the guideline and the commentary on the functions

8.1. Commentary on Simulink terminologies

8.1.1. Definition of basic blocks

In this guideline, the built-in blocks of standard Simulink library are defined as “basic blocks”
Below are the examples of basic blocks.



Related ID：db_0110、db_0143,jc_0641,db_0146,jc_0281

8.1.2. Definition of port blocks.

When the term "port block" is used in this guideline, it is referring to the input and output port of the subsystem.
Ports used for the conditional system are referred to as "condition input blocks". The block groups that include port block and condition input blocks are referred to as a "port block group".

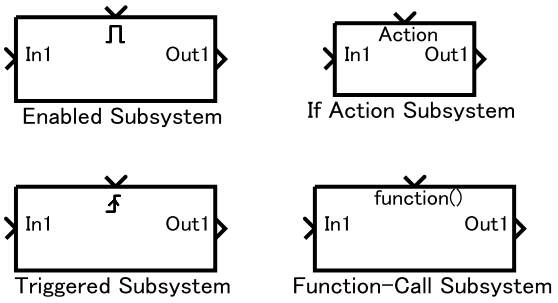
		Block type
Port block group	Port block	Inport,Outport
	Condition input block	Enable For Iterator Action Port Switch Case Action Trigger While Iterator

Related ID：na_0005,jc_0082,jc_0083,

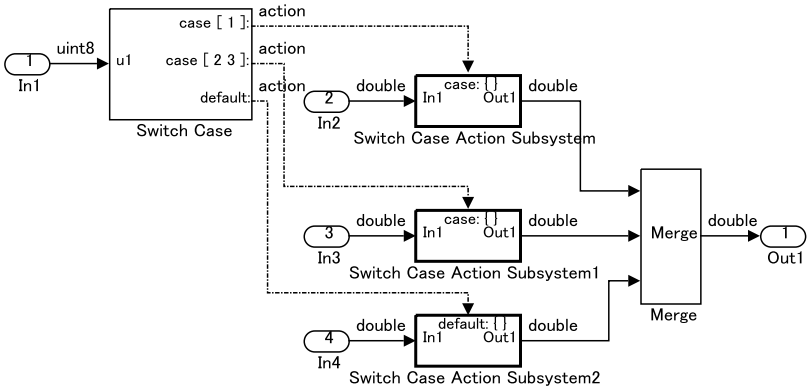
8.1.3. Conditional control flow

Flow listed using conditional subsystem that includes condition input block is referred to as “conditional control flow”

An example of a conditional subsystem



An example of conditional control flow

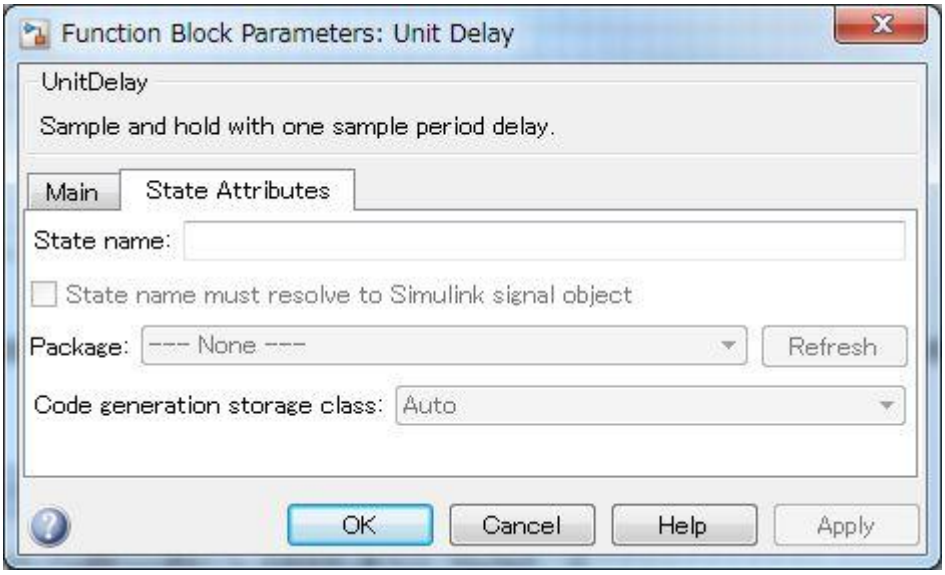


Conditional control flow indicates flow listed using a conditional subsystem. However, it does not indicate a function in which only one subsystem operates. A system that conducts calculation for several times for for iterator and while iterator also exist. The flow of original block in which the signal is input into a conditional subsystem and the conditional subsystem and how it's used form a pair, known as a "conditional control flow".
Related ID:na_0012,na_0028,jc_0658,jc_0656,jc_0657

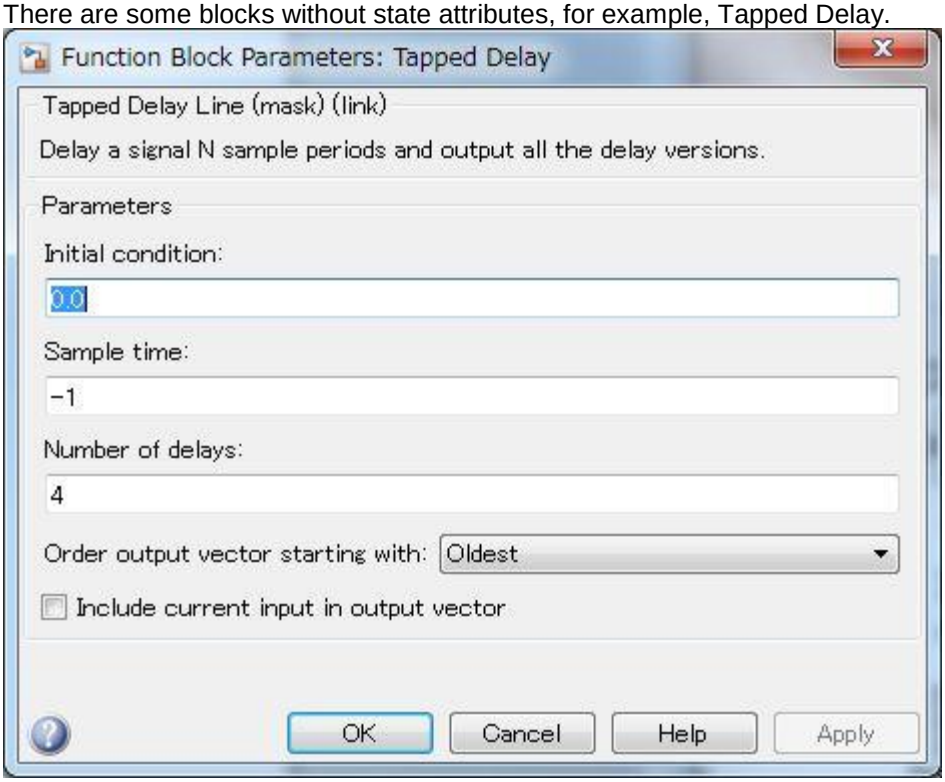
8.1.4. Blocks with State Variables

Block with state variables is a block that keeps values of the past in memory.
The blocks are stored under <Simulink><Discrete>.
Blocks with state variables have initial value(s). Blocks with state variables are blocks in which initial values setting is enabled. Also, most of blocks with state variables have the State Attributes property within the block properties.

Example of Block with State Attributes Property



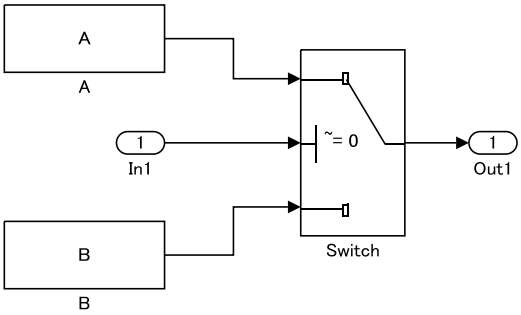
There are some blocks without state attributes, for example, Tapped Delay.



Note that a conditional control flow may have state variables depending on the flow's structure pattern.
Related ID：jc_0658,jc_0625,jc_0640

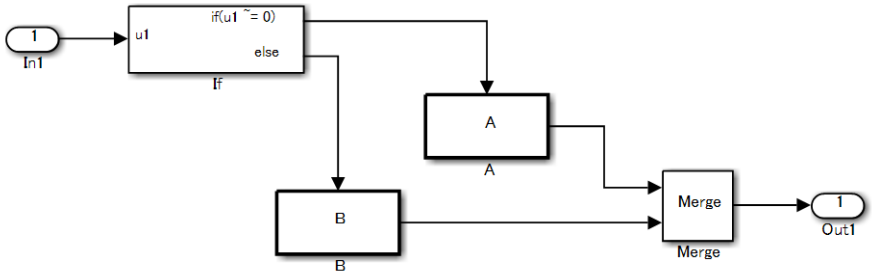
8.1.5. Branch Syntax with State Variables

Switch and Conditional Control Flow behave differently when they have a state variable.
Depending on the configuration setting, when any state variable exists, the Switch block generally executes subsystem A if the condition of control port is satisfied, and if not, 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 if the conditions of control port do not hold.

On the other hand, in the Conditional control flow, the subsystem A is calculated if the condition holds, and if not, the subsystem B is calculated instead of subsystem A, regardless of existence of state variables in subsystem A.



Reset action in recalculation can be specified by Action Port setting.
The behaviors of subsystem A using Switch and Conditional control flow are listed in the following tables:

Behavior of subsystem A

Control port condition	State variables (in subsystem A)	Switch	Conditional control flow
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 control flow	Hold	First time only
	Reset	At returned by condition

Understand the behaviors above to determine the more suitable structure to use, Switch block or conditional control flow, according on the intended use.

Related ID:na_0012,na_0028,jc_0658,jc_0656,jc_0657,db_0114,db_0115

8.1.6. The definition of subsystem

Subsystem is used for compiling various blocks and subsystems. However, they can also be used for other purposes. Below, usage methods that are not functional subsystems will be listed.

- Open function of the subsystem will be used.
  - This is used with the purpose of running several tools or displaying an explanatory text separate to the model.
- Mask display of the subsystem will be used.
  - This describe the outline or display fixed form documents, such as "classified"
- Turning block composition into groups using a subsystem bracket
  - From R2012b onward, subsystems can be place in the back of the block. Using this, the foreground of subsystem can be set to a slightly lighter color, such as grey. A subsystem can be used for compiling several blocks that do not require to be turned into a subsystem into a group.

When taking usage methods other than the above usage of compiling functions, generally such subsystem is set to exclude code generation as targets with blocks that do not have any input or output. Furthermore, if possible, these subsystems should not display the block name or use a determined block name if it is displayed, making it clear that it is not a general subsystem. When the expression "subsystem" is used in this guideline, it covers subsystems that "use functions separately", which always cover code generation. Categorically speaking, other subsystems have rules on the annotation side applied.
Furthermore, there are also subsystems that have had its setting changed to a mask subsystem (a subsystem that was simply set to NoReadOrWrite), in which a general user cannot see the content. This change could be made by the upper level user certified by the organization mask the subsystem after designing or reviewing it. This subsystem is excluded from the guideline's inspection target. A list should be created on exclusion targets and should be managed within the project.
Related ID:jc_0201,jc_0231,jc_0243,db_0144,jc_0111,jc_0653

8.1.7. The definition of a dictionary

The actual state of the title data dictionary differs for each project. MathWorks provide various methods, such as data management method that uses m-file and data management method that uses a model explorer. For example, MWJ proposes a tool that can manage data with a description method that follows data dictionary format stipulated by JMAAB (provided at MATLAB CENTRAL "SDOxlsIF: Excel Interface API for Simulink Data Object) Other than this, there are also companies that use DCM file based on ASAM as a data dictionary. Of course, it is possible to make a company's own format. As such, even the term "data dictionary" can be perceived differently depending on the project.

When the term “data dictionary” is used in this document, it refers to the list of signals and parameters managed by the various methods mentioned above.
Related ID：jc_0644,na_0035,jc_0251

8.1.8. Signal

The RAM value that appears in the data dictionary is referred to as a signal.
This refers to the variable used in the code generation that uses Simulink or mpt object. It is also referred to as signal object at times.
When the label name is added to the signal line before and after the block, and this is used for code generation, it is then referred as a signal.
In cases where only the label name is stipulated without having any Simulink or mpt object, it is a name that has an annotative nuance or was given to differentiate signals to use bus. Therefore, strictly-speaking, signals that do not appear in the data dictionary are not covered by he guideline.
Related ID:jc_0222,jc_0245,na_0035,jc_0251

8.1.9. Parameter

It refers to a RAM or ROM value that appears in a data dictionary with a fixed number.
Parameters that have values that are used for code generation that use Simulink or mpt object are either variables or constants. It is also referred to as a parameter object.
Parameters do not become altered during a single simulation. However, a variable-type parameter that has been stipulated as an adaptable RAM can have its values altered while Simulink is executed or after being implemented by using an external tool. This is called a relevant constant and can be altered during operation.
Parameters that do not appear in data dictionary are usually not treated as the target in this guideline. The parameters that become targets are few in number, such as the switching constant.
Related ID:jc_0232,jc_0246,na_0035,jc_0251

8.1.10. Signal label and signal name

Signal label is used to make the functions of the Simulink block diagram model easier to understand. Furthermore, it can also be used to manage the variable names used in simulation and code generation.
Signal name is inputted only once (at the time the signal is emitted). When displaying the inputted signal name at a different location of the model, it will be displayed as a propagation signal if the signal has not been converted functionally (Signals can be functionally converted by having it go through an integrator. Signals will not be converted functionally even if they go through a subsystem import that had become a nest). If the signal with a name given was functionally converted, have a new name be related to it.
Unless not specified clearly elsewhere, the “signal” guidelines can be applied to various types of signals.
For details of the representation of Simulink model signal, please refer to "How to Handle Signals", a Simulink Documentation
Related ID:jc_0222,jc_0245,na_0035,jc_0251

8.1.11. Control Characters

Control Characters are special characters used to control display and printer, including carriage return (CR), escape (ESC), tab (TAB) and so on. They are not be displayed on the screen.

See Also
http://e-words.jp/w/E588B6E5BEA1E69687E5AD97.html
http://www.c-tipsref.com/words/control_character.html
Related ID: jc_0222,jc_0232

8.1.12. Commentary vector signals/path signal

Vector

- Individual scholar signals that compose a vector need to have common functions, data type, and units.
- The most typical examples of a vector signal include sensor data grouped to a sequence with a location index and actuator data.

Bus

As mentioned previously, signals that do not fulfill the conditions as a vector can only be grouped as a bus signal.
Bus Selector block is only used with bus signal input. Do not use it to extract 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]

Types of vector	Size
Wheel speed subsystem	[1 wheel number]
Cylinder vector	[1 cylinder number]
Location vector based on a 2-dimensional coordination	[1 2]
Location vector based on a 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 $[\Theta_{fl}, \Theta_{fr}, \Theta_{rl}, \Theta_{rr}]$
	Acceleration
	Pressure
Controller bus	Sensor bus
	Actuator bus
Serial data bus	Circulating water temperature
	Engine speed, front passenger seat door open

Related ID：na_0010,db_0117,jc_0222,jc_0245,db_0097,jc_0630,jc_0659

8.1.13. Boolean type and boolean value

Boolean type refers to a Boolean type variable, which is characteristic to MATLAB.
This document uses the term Boolean type to mean that it is a signal that cn be perceived as either true or false. Within Simulink or in C programming language, there are times where these take a form of double, uint8, or boolean, depending on the configuration or the setting of the block. However, Boolean type, per semantics, refer to the calculation result of blocks that "deal with authenticity".
Boolean value displays true or false values.
Related ID:na_0002,jc_0141,jc_0655,jc_0757,na_0037

8.1.14. On 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.
Description per types of blocks that can be appointed is in “Simulink composition that supports the enumerated type”, under Help Simulink – Model – model composition – data type.
In order to use an enumerated type, it is necessary to define enumerate type using m file on MATLAB as seen in the example below.

Example: BasicColors.m
In this example, the characters of Red, Yellow, and Blue (Green) can be used.

```
classdef(Enumeration) BasicColors < Simulink.IntEnumType
  enumeration
   Red(0)
   Yellow(1)
   Blue(2)
    Green(2)
  end
```

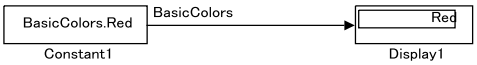
```
  methods (Static = true)
    function retVal = getDefaultValue()
      retVal = BasicColors. Blue;
    end
     function retVal = getDescription()
      retVal = 'This defines an enumerated type for colors';
     end
    %    function retVal = getHeaderFile()
    %  retVal = 'imported_enum_type.h';
    % end
    function retVal = addClassNameToEnumNames()
     retVal = true;
If it is set as % true, it will be shown as BasicColors_Red on C-code.
If % is not appointed or false is selected, it will be written as Red on C-code.
    end
  end
end
```

A method to customize the data types below will be provided here.

getDefaultValue	Except for the first value on the allowed value list, default enumerated value will be appointed.
getDescription	An explanation on the data type of Simulink® Coder™ generation code will be provided here.
getHeaderFile	It enables import of custom header file including the enumerated type definition of Simulink Coder generation code.
addClassNameToEnumNames	It avoids the competition of name with the identifier of Simulink Coder, making it easier to read.

For example, if a Display block is used, the display of 0,1, 2 is usually used for constant. However, a character can be displayed if an enumerated type is used.

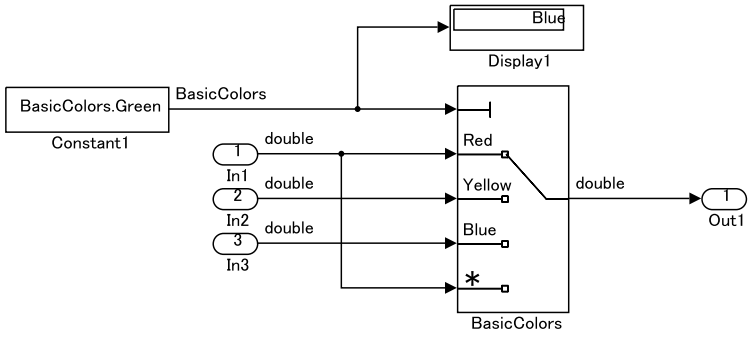
- Description method that stipulates the constant for enumerated type in a Constant block.



Simlink Coder can generate code by also using enumerated type.
In the default setting, the enumerated type data within generated code is stipulated within a header file model_types.h generated for a model.
For example, the default code for BasicColors will be as follows:

```
#ifndef _DEFINED_TYPEDEF_FOR_BasicColors_
#define _DEFINED_TYPEDEF_FOR_BasicColors_

  typedef enum {
    BasicColors_Red = 0,
    BasicColors_Yellow = 1,
    BasicColors_Blue = 2,       /* Default value */
    BasicColors_Green = 2
  } BasicColors;  /* This defines an enumerated type for colors */
#endif
```

You can choose either blue or green as the color for the signal. As shown in the example, the letters Blue or Green can be set to the same value 2 for enumerated type.
If the example case is used in SImulink, a Green setting is interpreted as Blue.
If two letters are set for 1 constant in this way, the letters written in the m file takes precedence as with setting the initial value.



8.2. Stateflow terminology commentary

8.2.1. Operators available for Stateflow

Operators available for use with Stateflow

Operator	Description
a * b	Multiplication
a / b	Division (Conditional use is available)
a %% b	Reminder
a + b	Addition
a - b	Subtraction
a >> b	Shift operand a right by b bits
a << b	Shift operand a left by b bits
a > b	Compare whether the 1st operand is greater than the 2nd operand
a < b	Compare whether the 1st operand is smaller than the 2nd operand
a >= b	Compare whether the 1st operand is equal to or more than the 2nd operand
a <= b	Compare whether the 1st operand is less than or equal to the 2nd operand
a == b	Compare whether the two operands are equal
a ~= b	Compare whether the two operands are not equal
a != b	Compare whether the two operands are not equal
a <> b	Compare whether the two operands are not equal

Operator 演算子	C language bit operation is available	
	OFF	ON
a|b	Logical OR of a, b	Bitwise OR of a, b
a||b	Logical OR of a, b	Logical OR of a, b
a&b	Logical AND of a, b	Bitwise AND of a, b
a&&b	Logical AND of a, b	Logical AND of a, b
a^b	b power of a	Bitwise XOR of a, b
!a	Logical NOT of a	Logical NOT of a
~a	Logical NOT of a	Two's complement

C chart supports the following unary actions

Operator	Description
a++	Increment a
a--	Decrement a

You can perform element-wise assignment operations on assignment operation vector and matrix operands.

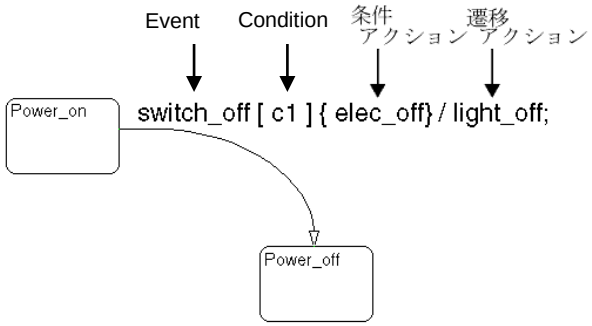
Assignment operation	Equivalent expression
a = expression	
a += expression	a= a + expression
a -= expression	a= a - expression
a *= expression	a= a * expression
a /= expression	a= a / expression

Related ID:jc_0737,jc_0742,na_0001,jc_0655,jc_0755,

8.2.2. Transition line condition, condition action, transition action

The entire descriptions on the transition line is referred to as the "transition label".
The following four descriptions are possible for the transition label.

1. Event
2. Condition
3. Condition action
4. Transition action



Related ID：jc_0754,jc_0753,db_0151

8.2.3. State Actions and Action Types

`entry`, `during`, `exit`, `bind` and `on` actions are called as action type.

List of Action Types

Action Type	Short name	Description
entry	en	Executes when the state becomes active
exit	ex	Executes when the state is active and a transition out of the state occurs
during	du	Executes when the state is active and a specific event occurs
bind	-	Binds an event or data object so that only that state and its children can broadcast the event or change the data value

Action Type	Short name	Description
on event_name	-	Executes when the state is active and it receives a broadcast of event_name
on after(n, event_name)	-	Executes when the state is active and after it receives n broadcasts of event_name
on before(n, event_name)	-	Executes when the state is active and before it receives n broadcasts of event_name
on at(n, event_name)	-	Executes when the state is active and it receives exactly n broadcasts of event_name
on every(n, event_name)	-	Executes when the state is active and upon receipt of every n broadcasts of event_name

The actions for states are assigned to an action type using label notation with this general format:

```
name
entry:
 entry actions
during:
 during actions
exit:
 exit actions
bind:
 data_name, event_name
on event_name:
 on event_name actions
```

Related ID：jc_0760,jc_0762

8.2.4. State Transition and Flow Chart

Stateflow can represent two features of state transition diagram and flowchart.
State transition diagram is a flow where states exist and state transition is made when conditions hold.
Flowchart is a flow where an action is executed at the change of condition regardless of changes of state.
Stateflow software allows a flowchart to be designed within a state transition diagram.
An entry action can be represented as flowchart in a state, which starts from default transition and moves to junctions through transition lines, as in the following example. Starting from an inner transition anebles `during` action by flowchart.

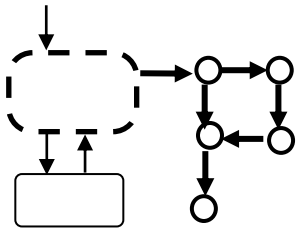
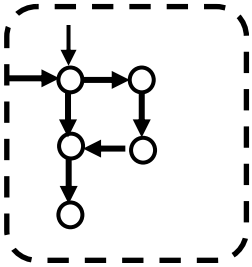
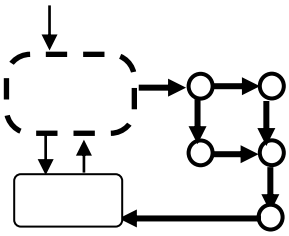
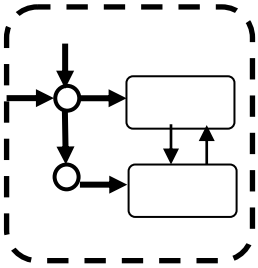
Additional information:
A flowchart cannot maintain its active state between updates. As a result, a flow chart always ends at a “terminating junction” (a junction that has no valid outgoing transitions).
By contrast, a state transition diagram stores its current state in memory to preserve local data and active state between updates. AS a result, state transition diagrams can begin executing where they left off in the previous time step, making them suitable for modeling reactive or supervisory systems that depend on history.

Flowchart and state transition diagram

	Start point	End point
Flowchart	Default transition or State	All endpoint are connected to the junctions
State transition diagram	Default transition or	Any of end points is connected to a state

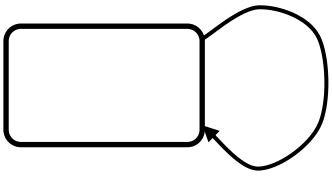
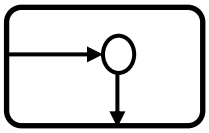
	State	

Difference from common flowchart and state transition diagram

	Flowchart outside state	Flowchart within state
Flowchart		
	State transition outside state	State transition within state
State transition diagram		

Mixture of flowcharts and state transition diagrams with self-transition is subjected of more strict constraints from both.

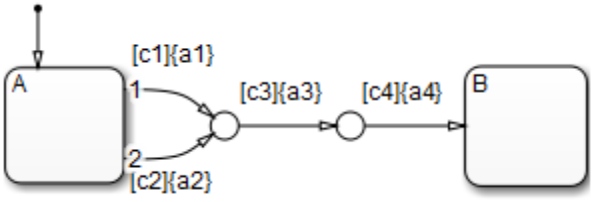
Example of flowchart with self-transition

	Self-transition outside state - Form self-transition outside state, reset after execution	Self-transition within state - Form self-transition in state, reset with `during` action
State transition		

Related ID:db_0132jc_0752

8.2.5. Backtrack

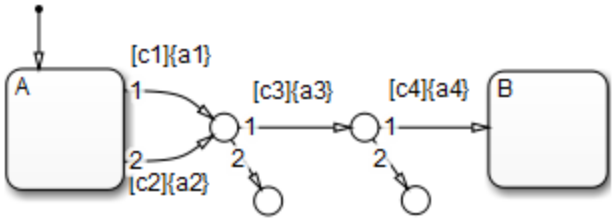
This example shows the behavior of transitions with junctions that force backtracking behavior in flow charts. The chart uses implicit ordering of outgoing transitions (see “Implicit Ordering of Outgoing Transitions").



Initially, state A is active and conditions c1, c2, and c3 are true, 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 condition c1 from state A to a connective junction.
2. Condition c1 is true and action a1 executes.
3. Condition c3 is true and action a3 executes.
4. Condition c4 is not true and 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 condition c2 from state A to a connective junction.
6. Condition c2 is true and action a2 executes.
7. Condition c3 is true and action a3 executes.
8. Condition c4 is not true and control flow backtracks to state A.
1. The chart goes to sleep.

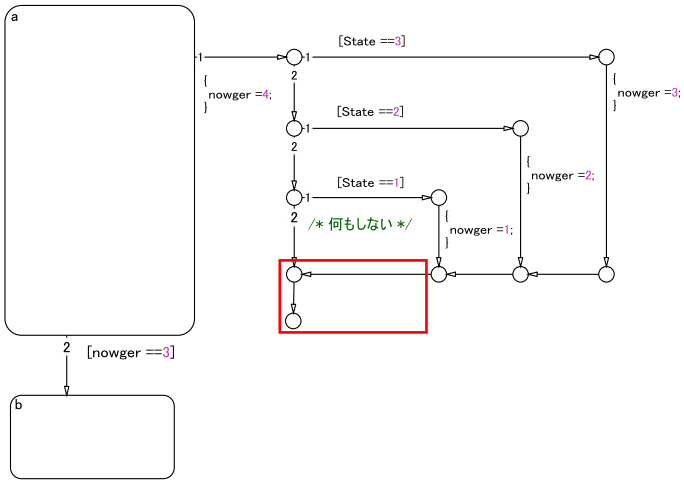
To resolve this problem, consider adding unconditional transitions to terminating junctions.
The terminating junctions allow flow to end if either c3 or c4 is not true. This design leaves state A active without executing unnecessary actions.



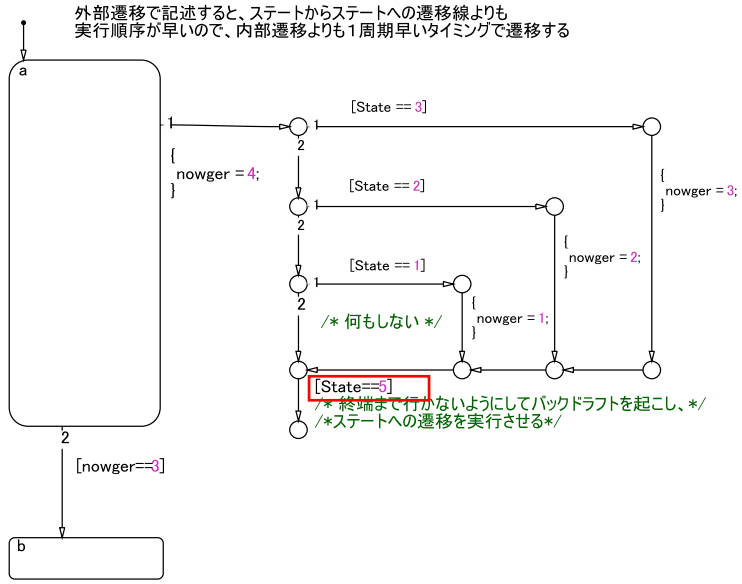
Related ID:jc_0751,jc_0773

8.2.6. Note on flowchart outside state

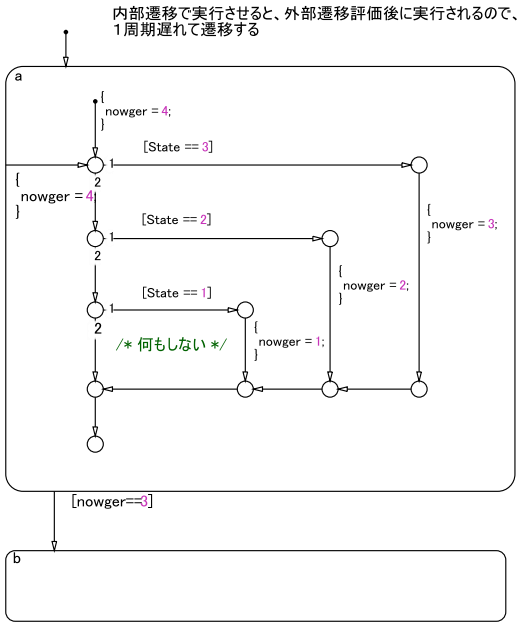
Writing a flowchart associated with a state is available either inside or outside of the state, however writing the execution order and backtracking require attention.
The following flowchart, which evaluates transition from `a` to `b` after executing flowchart outside state, appears to execute transition with the period same as that of `nowger` calculation.
However, transition line to `b` is not evaluated if the terminating point is reached via calculating transition outside state; this is a state transition diagram which stays on `a`.



The flowchart should be written as follows: adding a condition that does not stand at the end of flowchart outside state by design to make the transition line from `a` to `b` evaluated after executing flowchart outside state.
This enables the flowchart outside state to be executed before transition and to be evaluated with the latest value at the instant of transition. Note that this chart contains a dead path where condition never hold, which may cause a bug when the specification is changed in the future.



In contrast, the following flowchart where the internal flowchart is always calculated with execution of the state `a`, is written as easily comprehensible structure without dead paths.
Note, however, that it has such performance characteristic as evaluates transition from `a` to `b` in the next period of internal flowchart calculation period.
Due to this characteristic, calculation execution and transition may not be processed timely for the external flowchart. Use with sufficient attention.



Related ID:jc_0751,jc_0773

8.2.7. How to use custom C code

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.

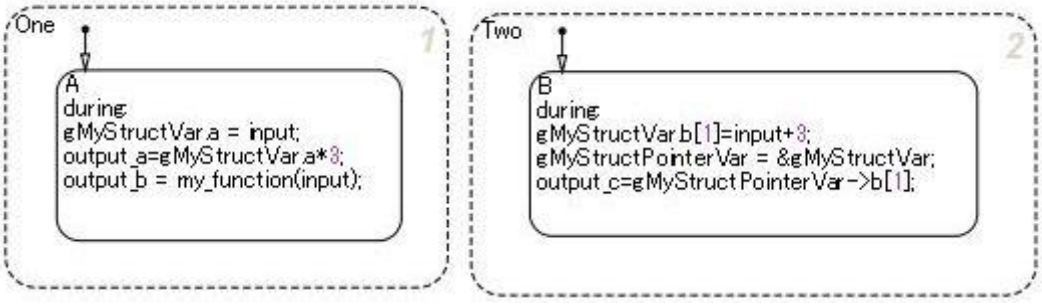
---------my_header.h----------

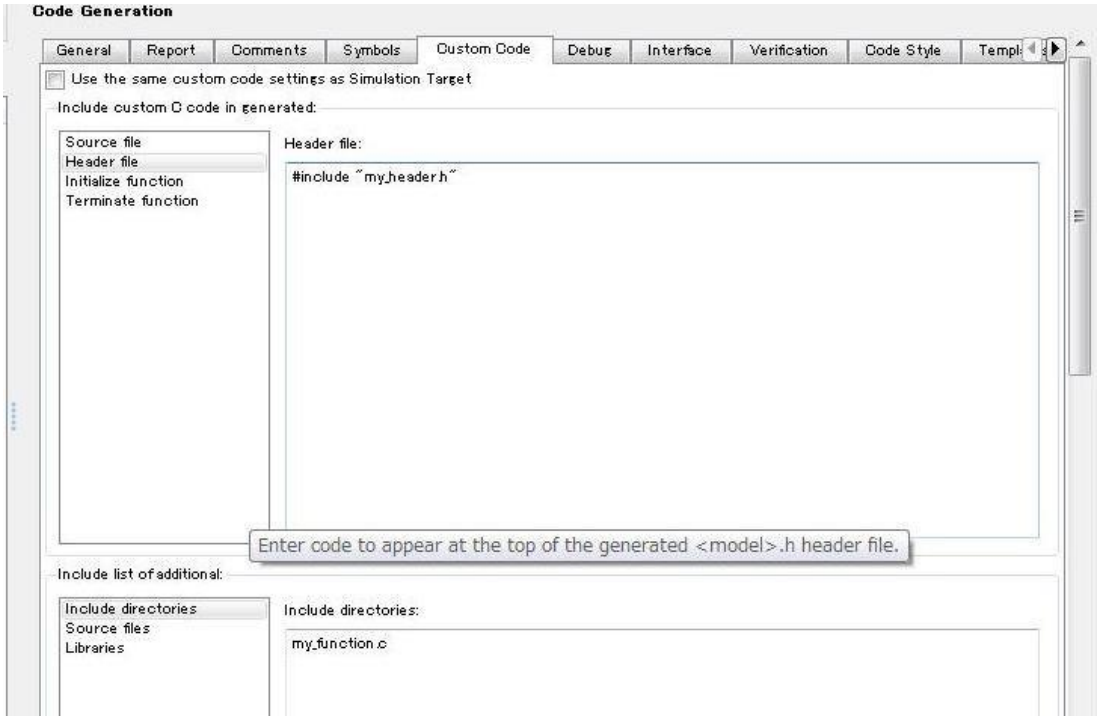
```
#include "tmwtypes.h"

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 ----------





Related ID：jm_0011

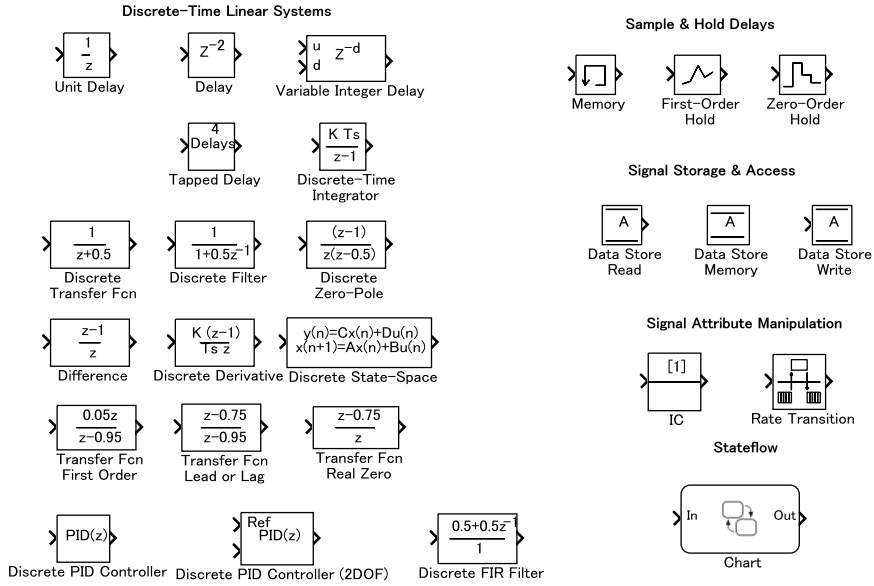
8.3. Initialization

8.3.1. Initial value setting in initialization

When a signal needs to be initialized, the initial values should be set correctly.
Cases that require initial values are the following.

1. When state variables are defined.
   - ① When blocks that have state variables are used.
     - A) Use the internal block settings.
     - B) Use the external input values.
   - ② When initial values are enabled for a block when a specific configuration is performed.
     - A) Set initial values in Merge blocks.
     - B) Use signals registered in the the data dictionary.
2. When signal settings (with RAM) have been defined that can be referenced from the outside.
   - A) Use signals registered in the the data dictionary.

8.3.2. List of blocks that have internal initialization values



- When initial values have been set inside a block, an initial value list using annotations is useful to allow you to visually confirm the input initial values.

(db_0140: Display of basic block parameters)

8.3.3. Initial values of signals registered in the the data dictionary

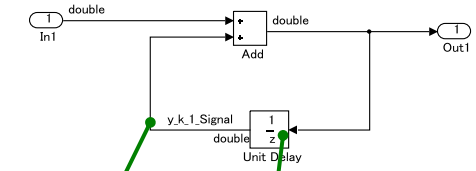
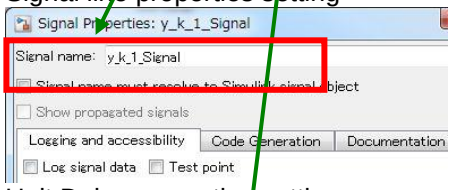
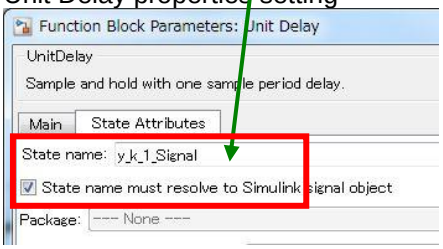
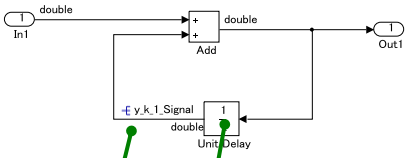
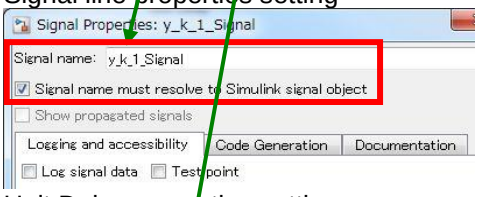
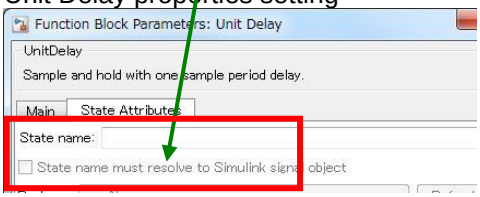
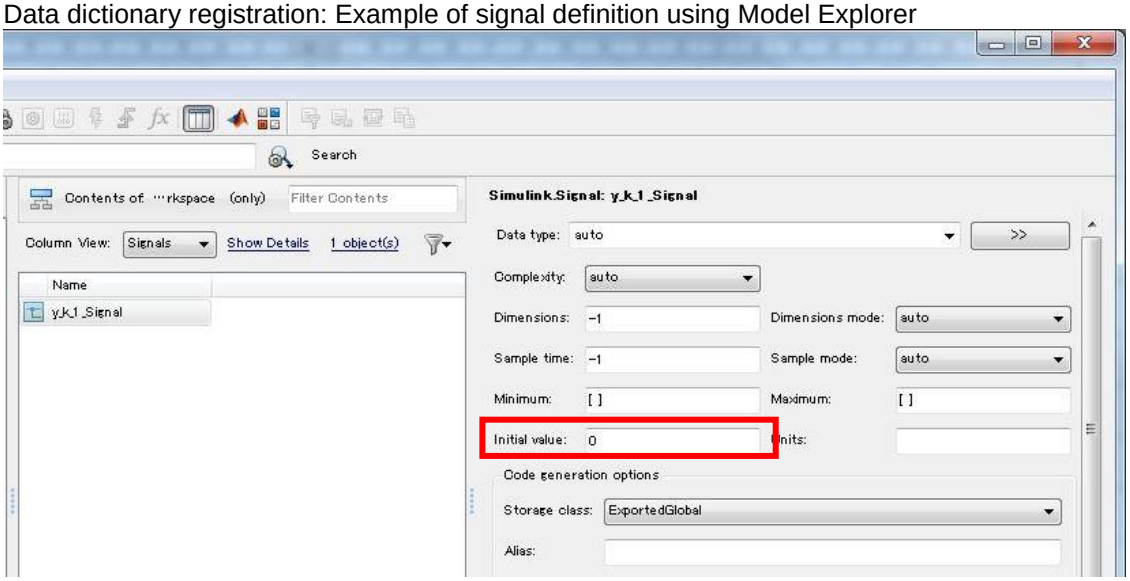
Set initial values for signals registered in the the data dictionary.

- Discrete block groups, such as UnitDelay, 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. When using a signal defined in the data dictionary for a state variable, the respective initial values should be conformed to the same value.

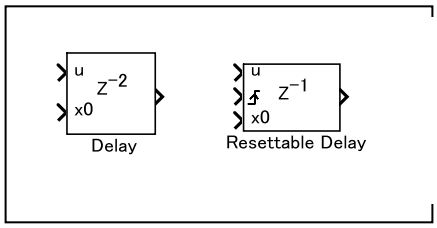
- When using a signal defined in the data dictionary for a state variable.

For Discrete blocks, such as a UnitDelay, 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 if the signal name of the data dictionary is assigned to the signal line, RAM will be reserved in duplicate, which would be a waste of RAM. Please use the label name in the sense of an annotation.

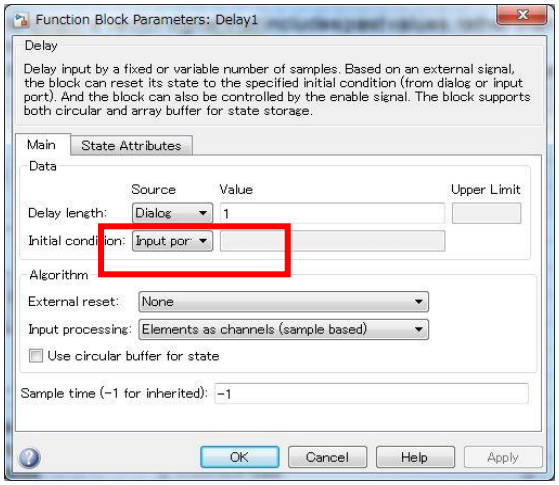
Correct: When the signal is defined for the state variables inside the block.	Incorrect: When the signal is defined for the output signal of the block that has state variables.
 Signal line properties setting  Unit Delay properties setting 	 Signal line properties setting  Unit Delay properties setting 
Data dictionary registration: Example of signal definition using Model Explorer 	

Signal objects which have been defined in Workspace can be automatically associated with signal objects and signal names of the same name, by using the disableimplicitsignalresolution (model name) command. However, for the above mentioned state variables inside the block, they get associated with the state variables inside the block and the signal name of the same name. If a globally set signal is associated with 2 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, because the model becomes unexecutable.

8.3.4. Example of a block where the external input value is the initial value

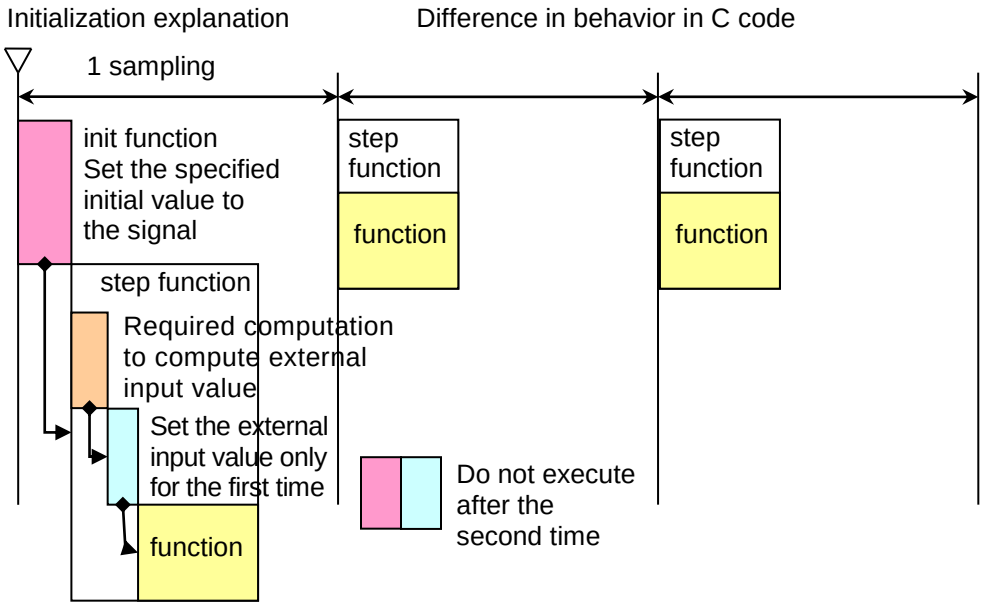


If the initial condition is set as the input port, the port name will not be displayed unless the block size is made slightly bigger than standard.



- Initialization behavior

If the initial value is input from the outside, the initial value of the signal in the data dictionary and the initial value of the model will differ.
In setting the initial value in initialization, the init function is called to set to the signal either the value set inside the block or the initial value defined in the data dictionary.
Next, the step function which is the data flow executive function is executed. When the external input value is set as the initial value, the initial value setting is executed only for the first time.
Please be aware in your modeling that in C code the executive function and the execution timing both differ.



8.3.5. Initial value settings in a system configuration that would enable initialization parameters

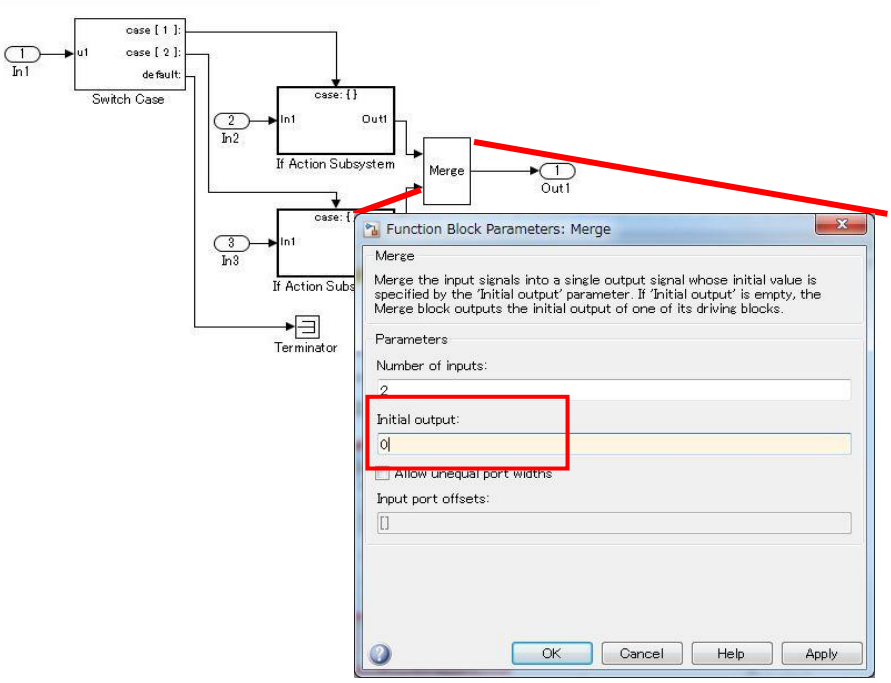
There are system configurations where, depending on their settings, initialization parameters are enabled for combinations of conditional subsystems and Merge blocks. If initial values are required in theses combinations of conditional subsystems and Merge blocks, either of the following modeling is performed.

For instance, either of the following methods can be used for conditional subsystem Outport + Merge

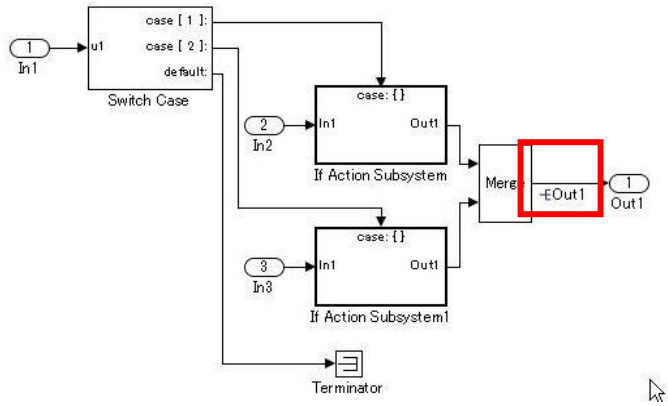
- set in Outport
- set in Merge
- if an mpt signal is defined behind Merge, set in mpt signal

Exception:
When there are successive blocks with initial values and settings for each block are unnecessary for clearly showing the signal's initial value.

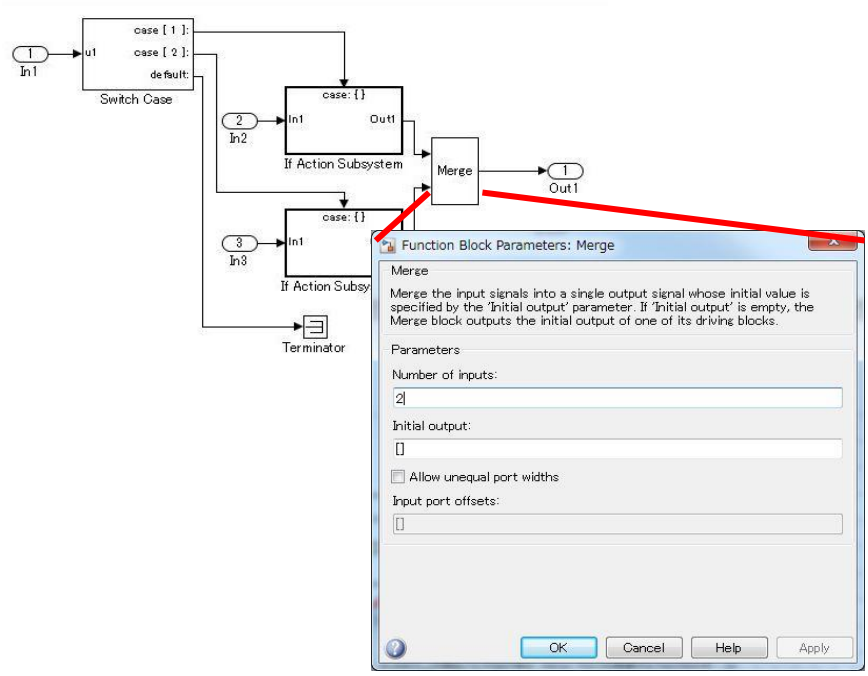
Correct: Initial value set in Merge



Correct: Initial value set in mpt object



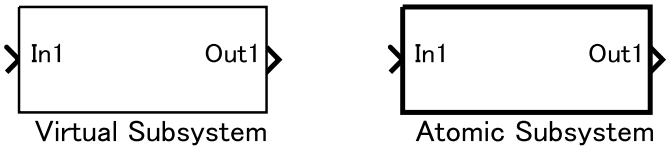
Incorrect: Despite the requirement for an initial value setting, it is not shown anywhere.



8.4. Supplement: Commentary on functions

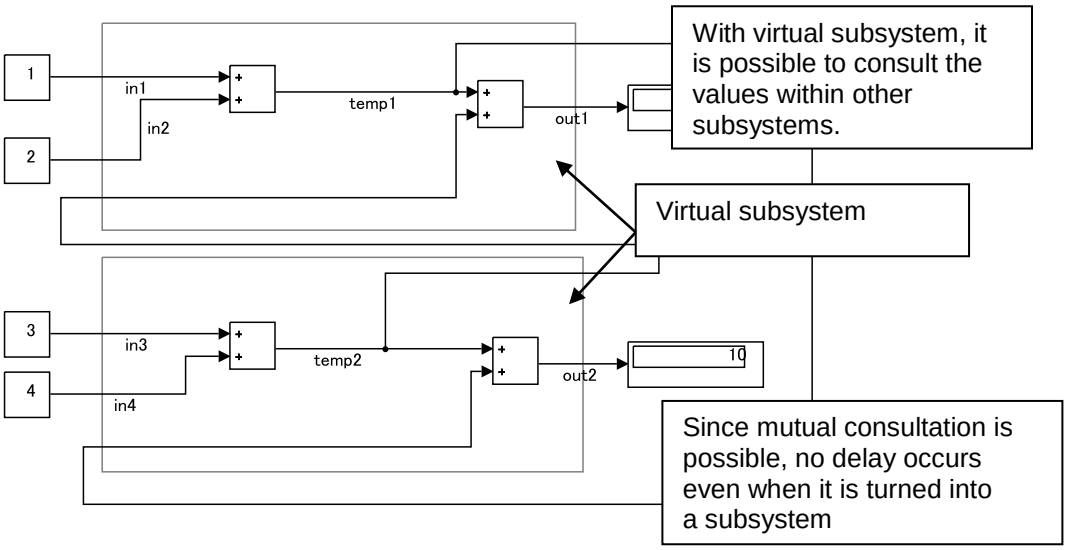
8.4.1. About Atomic Subsystem

Subsystem has a two types of settings: a setting referred to as "Atomic Subsystem" and another as "virtual". The difference between the virtual subsystem (a subsystem block in a default setting) and Atomic Subsystem is whether a subsystem is treated as a block or not.
It does not have a practical meaning in a mathematical or physical sense, but a block that simply provides visual expression is called a "virtual block".
For example, Mux block that compiles several signal line, From block that hands out the signal, and Goto block all correspond to a virtual block. Since the subsystem block in the default setting only constitutes a merely visual hierarchical structure, it also falls under virtual blocks. This subsystem is referred to as a virtual subsystem.
The line of the virtual system external bracket is displayed thinly while the one for Atomic Subsystem is displayed thickly.

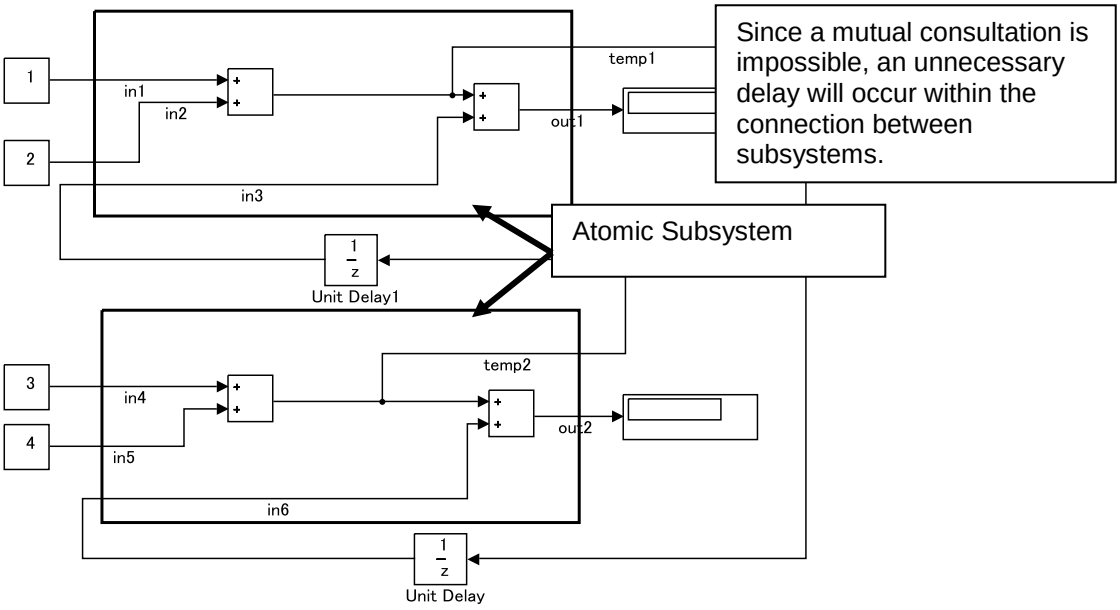


The major difference between Atomic Subsystem and virtual system is that the Atomic Subsystem is detached from the external system, being not subjected to cross-border optimization.
For example, let's suppose a subsytem that consuls the external calculation result within a subsytem like in the example below. This system is calculated from the four equations below.
temp1= in1 + in2
temp2= in3 + in4
out1=in1 + in2 + temp2
out2= temp1 + in3 + in4

Example of a virtual subsystem definition



However, Atomic System does not use internal calculation results for each subsystems. Therefore, interium output value will use a calcualtion result that is delayed by a session.
temp1= in1 + in2
temp2=in4 + in5
out1=in1+ in2 + in3
out3=in4+ in5 + in6
in3=temp2
in6=temp1
Atomic Subsystem is prohibited from directly referincing the interium calculation result to other subsystems.



Atomic Subsystem can select factor settings of C-source.
With Atomic Subsystem, as explained above, the inside of subsystem will become encapsulated (objectified). Depending on the relationship between before and after, one should acknowledge that a static RAM field can be secured for the output signal. Atomic Subsystem (including the addition of factor setting) should not be used carelessly for reasons such as to merely make the test easier to do. Setting that conducts factor setting will not simply have a factor name inserted within a C code. It is necessary to acknowledge that it is described as a mathematically independent system and to review under which cases Atomic Subsystem should be used.
Including the relation with the structure layer that will be mentioned later on, it is necessary to determine an operation rule per project and to determine its relationship with the guideline rules.

The difference between Atomic Subsystem and virtual subsystem (Japanese)
http: //www.mathworks.co.jp/support/solutions/ja/data/1-CYPFSL/index.html? product=SL&solution=1-CYPFSL
Atomic Subsystem (Japanese)
http：//www.mathworks.co.jp/jp/help/simulink/slref/Subsystem.html
Explanation of algebra loops (Japanese)
http: //www.mathworks.co.jp/jp/help/simulink/ug/simulating-dynamic-systems.html#f7-19688

9. Determining guideline operation rules

Describe the deployment rules and processes for the guideline implementation.

9.1. Necessity of process definition

Automobiles need to be safe. In order to develop a safe product, various initiatives will become necessary. The model base development that utilizes simulation is suitable for developing a safer system. However, this doesn't mean that a safe system will be made just because simulations were used. Although the development of good control and good functions are necessary, process definition and the development environment that will be used will be equally important. A safe system planning will be conducted after implementing various agreements when starting the development.

9.2. A version of MATLAB/Simulink

When starting a project, the version of MATLAB/Simulink to be used will be determined.
This includes mixing various MATLAB versions for each process.
For example, if a version that conducts automatic code generation was “R2011b”, it is possible to generate code and conduct verification with R2011b by downgrading test cases by generating test cases by having Simulink Design Verifier (SLDV), a verification tool box that uses a formal method, use R2013a.
For each project, one should decide upon which software version to be used at which stage. At that specific process, the version that was decided upon should be used by everyone.
Furthermore, it is necessary to check the latest bug report on a regular basis. Depending on the bug, one may need to change to the latest version. It doesn't 't mean that one cannot change once after making a choice. One needs to appropriately evaluate the risks of malfunctioning occurring due to a bug and risks from upgrading the version. It is necessary to always have a structure in place that allows to be changed to the latest version and to appropriately evaluate and judge what is the safest option.

9.3. MATLAB/Simulink setting

The setting of MATLAB/Simulink specifically set for each MATLAB should be operated in a unified manner with the project. In particular, Simulink setting that affects the appearance setting requires unification. The option name to be unified will be listed below.

- Displayed standard value of a new model
  - The display of a mask subsystem
  - The display of a library link
  - Displaying non-scholar line with a wide-width lines
  - The display of a data-type terminal
- Font setting of a new model
  - Block/line/annotation
- Standard value of an editor
  - Using the traditional block diagram theme

9.4. Usable blocks

jm_0001 and hd_0001 display the blocks that are prohibited to use. These rules are rules determined by whether the code generation is enable/disable. However, usable blocks are not only able/disable to generate codes, they also change depending on the education level.

There are many blocks in Simulink. Depending on the block, an efficient code can be generated or a combination of several basic blocks can be represented using one function. However, when there's a difference within the SImulink skill level within an organization, one should limit the blocks and design within a designated range. However, decreasing the block number too much can deteriorate the readability. Adverse effects include increasing the user library and variation within the descriptions for the same function.

An engineer that possesses a skill level that the organization sees as the standard should be set. A list of usable Simulink block should be made and operated.

When an advanced practitioner uses an unsupported block, it should be stored within a mask subsystem, concealing it so that it cannot be seen by a general user.

In conjunction to the eduction structure, the operation rules will be determined when starting a project.

9.5. Setting of the configuration to be used

9.5.1. Optimization parameters

Optimization options highly affect codes generated through automatic code generation. With a good understanding of your own product characteristics, these options should be configured so that the setting match to the security level suitable for the product. Optimization should not be applied easily for the products that require utmost consideration to security.
In general, for automotive built-in products, computing speed is critical, and also less RAM/ROM is thought to be ideal. For example, for auto-industry products, optimization settings are enabled on the “Conditional Input Branch Execution” pane. This improves computation rate by executing only the side where the condition holds during execution of the conditional branch using Switch.
In contrast, for aviation industry, the pane is disabled since stabilization of execution speed is critical, and calculating in both sides is preferred in order to keep stable calculation period even if calculation is needed only on the side where the condition holds.
These optimization settings are also deeply related with the SIL level of function safety, as described above vary in adoption criterion depending on industries, need to be determined with understanding of your own product characteristics.

9.5.2. Other configurations

✧ Hardware implementation parameter settings

Describes model system hardware characteristics, including products and test hardware configuration setup for simulation and code generation.
Configure appropriately to be compatible with the microcomputer the project uses. Especially mind unintended utility function might be inserted unless signed integer division rounding is defined.

✧ Model reference parameter settings

Specified when using model references.
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 parameter settings

Configures a simulation target of a model including a MATLAB Function block, Stateflow chart, or Truth Table block.

9.5.3. Configuration settings

For configuration settings, see the hisl and cgsl guidelines developed by MathWorks. The guideline describes recommended patterns for each version. Determine to accept or reject according to the needs of individual projects.

hisl_0040: Configuration Parameter > Solver > Simulation Time
hisl_0041: Configuration Parameter > Solver > Solver options
hisl_0042: Configuration Parameter > Solver > Tasking and sample time options
診断
hisl_0043: Configuration Parameter > Diagnostics > Solver
hisl_0044: Configuration Parameter > Diagnostics > Sample Time
hisl_0301: Configuration Parameter > Diagnostics > Compatibility
hisl_0302: Configuration Parameter > Diagnostics > Data Validity > Parameters
hisl_0303: Configuration Parameter > Diagnostics > Data Validity > Merge Block
hisl_0304: Configuration Parameter > Diagnostics > Data Validity > Model Initialization
hisl_0305: Configuration Parameter > Diagnostics > Data Validity > Debug
hisl_0306: Configuration Parameter > Diagnostics > Connectivity > Signal
hisl_0307: Configuration Parameter > Diagnostics > Connectivity > Bus
hisl_0308: Configuration Parameter > Diagnostics > Connectivity > Function calls
hisl_0309: Configuration Parameter > Diagnostics > Type Conversion
hisl_0310: Configuration Parameter > Diagnostics > Model Referencing

hisl_0311: Configuration Parameter > Diagnostics > Stateflow
最適化
hisl_0045: Configuration Parameter > Optimization > Implement logic signals as Boolean data (vs. double)
hisl_0046: Configuration Parameter > Optimization > Block reduction
hisl_0048: Configuration Parameter > Optimization > Application lifespan (days)
hisl_0051: Configuration Parameter > Optimization > Signals and Parameters > Loop unrolling threshold
hisl_0052: Configuration Parameter > Optimization > Data initialization
hisl_0053: Configuration Parameter > Optimization > Remove code from floating-point to integer conversions that wraps out-of-range values]
hisl_0054: Configuration Parameter > Optimization > Remove code that protects against division arithmetic exceptions
hisl_0055: Prioritization of code generation objectives for high-integrity systems

Modeling Guideline
cgsl_0301: Prioritization of code generation objectives for code efficiency
cgsl_0302: Diagnostic settings for mutilate and multitasking models

9.6. Guideline rules that are used

The numerical values and the list in the rules are recommended standard values. They are not numerical values that must be adhered to. For example, the hierarchizing of the na_0038 state is written to be up to 3 hierarchy level. However, there is no need to necessary operate by limiting it to 3 hierarchy. It can be altered to 5 hierarchy level.

Within the guideline, there are types of blocks to become the subject and parameters that can be changed within rules, and not just numerical values. These parameters will be listed in the "Rule Parameter List" as an attached resource material.

Furthermore, with a state in which all rules can be checked by an automatic checker as a precondition, a list has summarized which rule should be effective in which situation.

9.6.1. The adoption of the guideline rule and the setting of the process

It is necessary to determine which rule to be adopted in what sort of composition. It should be determined at the start of the project as to which adopted rule will be used at what sort of process. The guideline requires an appropriate operation rule that matches with the development process, such as: will the evaluation only be done at the final stage where the automatic code generation conducted or whether the adopted rules be switched according to the stages starting from the initial development phase?

9.6.2. The setting of the guideline rule application field and the clarification of the exclusion condition

It is necessary to determine the field to adopt the rule. For example, many rules should limit to the adoption of the model that represented the Autostar application field. With models that achieved interruption used in the basic software field or models that add process that prohibits interruption during calculation execution, there are many processes that cannot be achieved without using several special custom S-function or Data Store Memory blocks. Furthermore, with fields that only professional who specialize in said field writes down, such as the designing of a custom library block that many users use, is not a restricted area that this guideline is aiming for to begin with.

Many rules in this guideline are made by having the field in which several engineers with a moderate level edit as the target. The rules were made with the intention that a model with a high intelligibility will be made within that field. A field that can be achieved by a selected few professionals using specialized techniques should be excluded from the restriction target of the guideline by limiting said field and establishing a unique system in which only the professionals touch the field.

Furthermore, when having a control model for the entire model that is operated with RCP as a guideline rule subject, the entire model should not be set as a target easily; instead, the field needs to be limited. It it necessary to conduct a code generation and pay attention to the areas that will be implemented to the built-in microcomputer and areas that will not. Scheduler model that won’t be implemented and made only for RCP, PWM signal that is only for operating the real machine, and the interface section that includes

blocks that correspond to the drivers such as CAN signal, are not the control models that this guideline applies to.

As mentioned above, when changing the application field of the guideline within the same model, a model structure that separates code generation target from fields that are not is required. Furthermore, their unique rules also need to be added.

9.6.3. The decision on the parameter that is stipulated in the guideline

This guideline or ones that the users set should not simply be adopted as they are. Instead, various parameters need to be reviewed in accordance to the characteristics of the product and the development environment tools that are being used.

For example, "in the jc_0061: display", there are parts where the organization's education state determines the block type in which the block name should be displayed, block type that should not be displayed, and the block type that could be either. There are also times where different setting values are set due to the difference in the group process of the users.

9.6.4. Guideline checker adoption process determination

Whether to adopt an automatic checker or to check by eyes during the review session for the checking process should be determined first.

It is possible to used a checker created in one's own company.

Having many automatic check items will reduce the time for review. However, even if everything can be automatically checked, a review should always be conducted by a highly skilled member. Checks should not only done by an automatic checker but it is effective when combined with a review.

The rule adoption is determined by the organization’s education level (i.e., which process is being adopted) and is not only determined by the functions that the project should achieve or their size.

9.6.5. Addition of the model analysis process

The designed model preferably should be set when reviewing the list of rules to be adopted by analyzing the usage tendency of the block and the school of the description style. If possible, the rule review period should be set in advance during the initial stage of the project. For example, the frequency of used block of an analysis of a simple model can be investigated by using sldiagnostics. Adjust the operation rules list by identifying blocks that are frequently used and those that aren’t. Furthermore, measure such tendencies such as at which coordination plane the block that has status variables such as UnitDelay are located at, whether to have UnitDelay outside or inside of the subsystem, whether to set abs block to the output side of the subsystem, and whether to process it at the input side after receiving a signal. The addition of rule to unify the schools and anticipating in advance the modification labor hour will lead to the improvement of re-usability later on.

9.6.6. Rule alteration procedure

Rules that have been decided upon once do not require to be strictly adhered to for eternity.

When changing the rule, a correct procedure and process are required. Listen to the needs of the designer and review what needs to be changed. After that, if the root issue for the alteration is caused by misunderstanding of the usage, the addition and execution of training is necessary, rather than revising the rule. However, if there is a restriction arising from the control specifications and objectives of the company or hardware (i.e., implemented microcomputer), a procedure to relax the rule according to the needs should be set.

9.6.7. Arrangement of development environment

Using CMM and SPICE as reference, adopt a process in accordance to the level of each project and make stipulations in accordance to the level.

Levels may refer to the maturation level of MBD infiltration, training level, skill level, and the size of the model. Otherwise, if the target product is subjected to function safety (ISO26262), SIL level will also become involved. When conducting a system design with a high SIL level, traceability should be secured for various parts within the process.

For example, if there is a project with difference in various data sets, there should be a management chart that dictates which data dictionary should be used for that project. When conducting automatic code generation, it is necessary to prove whether the operation was conduct according to the management chart.

Immediately before an automatic code generation, read the management chart into it, automatically read that data in according to the chart, and read in the correct appointed configuration set before conducting the code generation. Within this process, the following will become necessary: the appointed data dictionary, data within the work space, used configuration, and lastly, the storage of all the logs on people who packaged the models and codes and stored, people and the files that were read into the PC, and the types of codes, and the report output.

Instead of creating this system manually, an automatic generation using a tool is effective. This is because concerning the numerical value selection mistake of the data, a third-party other than the person who set the value cannot easily identify which value should be used by which project. However, if there's a document link that displays the basis, it can be determined by a third party or automatically. Combining it with the data, using the original chart with the intention of the designer written on it will decrease the possibility of generating mistakes. Such automatic system is needed to begin with even if the SIL level is not high.

Of course, a system that automatically checks the guideline rules should aslo be utilized. A system that checks the rule according to the unique decisions of the company will also become necessary. Write down an account for checker to modify an area with issues or exclude areas with no issues after checking the areas detected as errors when checking with a checker, Naturally, unique rules for the checker to determine exclusion will be required. There is a necessity to develop tools that customized these areas.

10. Model Architecture Explanation

This chapter describes only the outline on model architecture suitable for model-based development to share the concept, since it is difficult to establish standards for model architecture which includes combination of the existing software of individual companies with the model architecture explained in the JMAAB Guideline Ver. 1.0, and Simulink also provides a variety of features appropriate for the unique circumstances of each company.

10.1. The roles of Simulink and Stateflow

It is possible to describe all systems to be compatible with either Simulink or Stateflow.
When Stateflow alone is used, Simulink is required for in/outputs and structuring only, but within Stateflow a variety of formula processing is possible. When using Simulink, it is possible to realize complex state variables through methods such as the use of Switch-Case blocks.
Accordingly, whether Simulink or Stateflow is used in modeling specific parts of control algorithms comes down to subjective views on which one is easier to understand. The technique to realize this should be selected depending on the training level within organizations.
In most cases RAM efficiency is worse for Stateflow than it is for Simulink. Therefore, Simulink has an advantage in computations that use simple formulas. Apart from that, Simulink is also more advantageous in instances such as state variables that can be operated with simple flip-flops and Relay blocks. When describing things with Stateflow that can be described with Simulink, the most suitable technique should be investigated in consideration of the following risks.

- Static RAM must be ensured to allow visualization of Stateflow inputs, outputs and internal variables.
- When general computational formulas are used internally, the user designs the overflow prevention.
- When the computations are done externally, the whole gets segmentalized, reducing the level of understanding of the whole.

There are cases when Stateflow obtains more efficient sources than Simulink for optimum expressions that are close to C source, but these kinds of models do not have a good appearance nor are they very easy to understand. In these kinds of cases, it is more beneficial to use S-functions instead of using Stateflow modeling.
Stateflow can note computations where specific arrangements are specified, or computations using for-loops, more efficiently than Simulink, but in recent years the use of MATLAB language for descriptions in the latest MATLAB has also become very convenient.

When modeling using Stateflow, if dealing with states as described below, readability improves by describing them as state transitions.

1. Different output values are output for identical inputs.
2. Multiple states exist. (if possible, from 3 or more)
3. Where a meaning of a state is defined, that is not an infinite number but a discrete value.
4. Inside a state, initialization (first time) and differentiation during execution (after the second time) is required.
5. Apart from state variables, input and output variables are signals that can be visualized.

For instance, in flip-flop circuits, different output values are outputted for inputs. Moreover, state variables are limited to 0, 1. However, in the sense that for the input/output variables 0, 1, both minimum and maximum state variable values 0, 1 are used, there is the possibility of classification in infinite numbers. Also, there is no differentiation between initialization and during execution inside a state. In other words, only 1 flip-flop applies out of the 4 above, so Simulink can be said to be more advantageous.
The question as to whether Simulink or Stateflow must be used for the design should be answered in consultation with several people, depending on the problems that must be implemented. Whether implementation in Stateflow is with state transitions or with flow charts should also be determined in consultation.
Things that should be handled as states are state transitions and conditional branches that are not states are flow charts. Truth tables are also classified as a conditional branch implementation method.
Moreover, when designing the above mentioned states as state transitions using Stateflow, Classic mode should be used in order to implement it as software into the control system’s embedded micro controller.

Stateflow is HDL coder supported. Mealy and Moore modes should be used when implementing as HDL coder. Moreover, when protection is required against internal electric leaks, the Moore mode is more appropriate.
These guidelines do not describe cases of use as HDL coder. Please note that these are guidelines for Simulink and Stateflow that are implemented as software in control systems.

10.2. Hierarchical structure of a controller model

Shows the separation concept, or the layout concept, for the hierarchical structure of a controller model, as reference examples. This is not a clear standard as a rule, but it is a basic approach to modeling.

10.2.1. Types of hierarchies

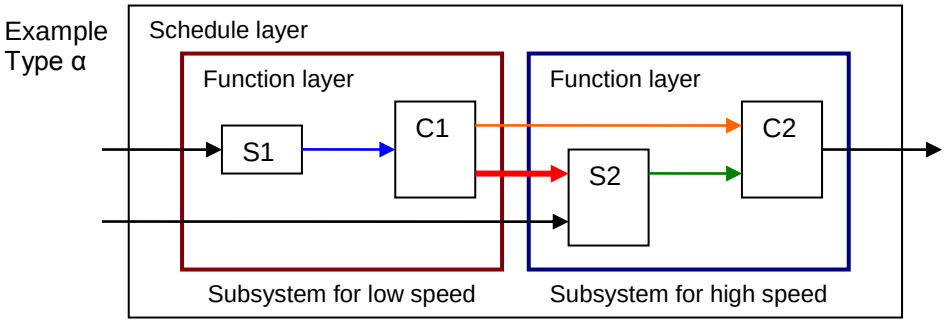
- Building method of hierarchies
  - Division into subsystems with the main purpose of space adjustments within the layer should be avoided.
  - The following layer concepts should be allocated to the layers, and subsystems should be divided based on that.
  - Unnecessary layer concepts do not need to be allocated to a layer.
  - Multiple layer concepts may be allocated to one layer.
- Layer concept

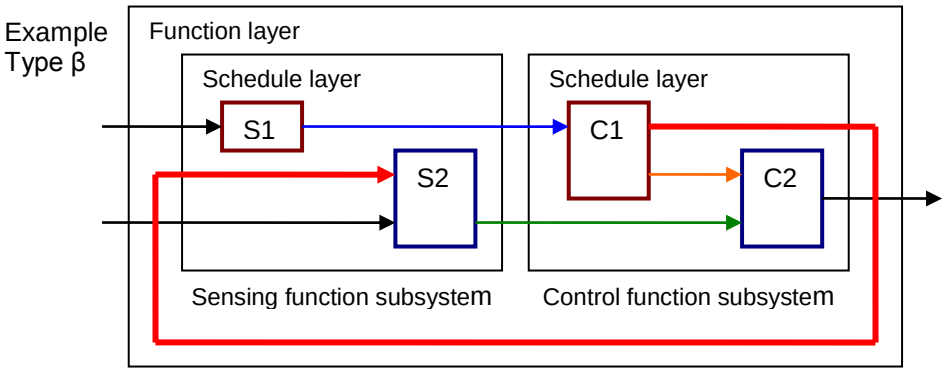
	Layer concept	Layer purpose
Top Layer	Function layer	Broad functional division
	Schedule layer	Expression of execution timing (sampling, order)
Bottom Layer	Sub function layer	Detailed function division
	Control flow layer	Division according to processing order (input → judgment → output, etc.)
	Selection layer	Divide (select output with Merge block) into format that switches the active subsystem and execute
	Data flow layer	Layer for non-separable computations

10.2.2. Layout method for top layer

There are principally 3 types of layout methods for the top layer.

- Simple control model
  Represents function layer and schedule layer in the same layer. Here, function = execution unit.
  Example: When the control model only has one sampling cycle, and all functions are arranged in execution order
- Complex control model Type α
  Schedule layer is placed at the top.
  Makes integration with the hand-written code easy, but functions are divided and the readability as a model is reduced.
- Complex control model Type β
  The function layers are arranged at the top, and schedule layers are built below the individual function layers.





The subsystem indicated in bold is set to be an atomic subsystem.

10.2.3. : Modeling method for function layers and sub-function layers.

- Division into subsystems by function. The respective subsystems represent "1 function".
- "1 function" is not necessarily an execution unit. For that reason, the respective subsystems cannot necessarily be made into Atomic Subsystems.
  (For type β in the example above, it is appropriate to make the function layer subsystems into virtual subsystems. If they are changed into an Atomic Subsystem, algebraic loops are created.)
- Using annotation, the function overview must be either described on the layer or included in the subsystem overview and displayed as an annotation.
- If there are several big functions, partitioning of the model, using model references for each function, should also be considered.

10.2.4. Modeling method for schedule layers

Sampling intervals and priority order should be set.
The previous guideline corresponds to the approach that uses "jc_0321: Trigger layer".

・ Point for attention when setting multiple sampling intervals

In connected systems with varying sampling intervals, a signal is required for the fast cycle for times even when the signal for the slow cycle has not been computed. When connecting using different sampling intervals, a pinned RAM area is always required. For that reason, always split systems for each different sampling times in the top layer, without connecting different sampling times in the bottom layer.

・ Setting priority ranking

This is important when designing multiple different independent functions. It is advisable that computation sequences are freely determined as much as possible depending on all subsystem connections.

For the priority order, the following two need to be set: priority ranking for different rates and priority ranking within an identical sampling rate.

・ Implementation method for sampling interval and priority ranking

The described methods can broadly be divided into 2 types.

1. Perform setting of sampling times and priority rankings for subsystems or blocks.
2. Using conditional subsystems, the user sets independent rankings to match the scheduler.

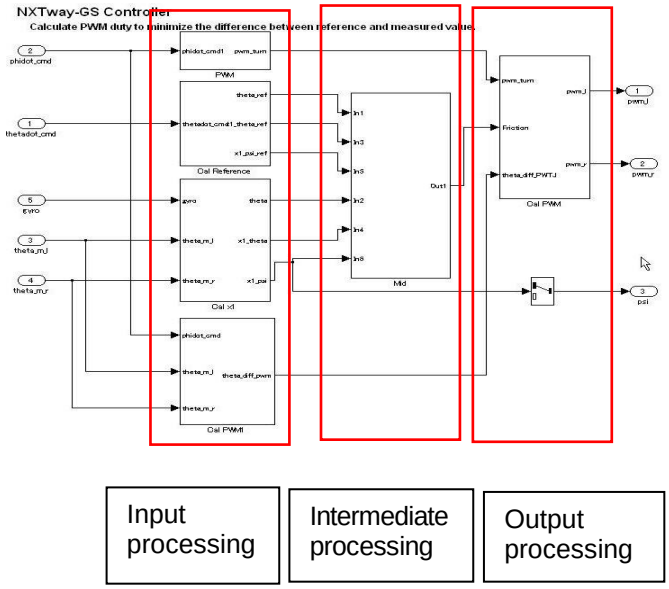
Patterns exist here with various conditions, such as configuration multi-rate and single rate, Atomic Subsystem setting, use/non-use of model references. Which among these are employed is closely linked to the C code implementation method, and substantially varies depending on the project status.
The typical factors that are substantially affected are listed below.

- On the model side
  - Do several sampling times exist in the model?
  - Is it a model that realizes several independent functions?
  - Use of model references
  - Number of models (whether there are multiple sources with code generated in Simulink)
- On the source side
  - Use/non-use of real-time OS
  - Consistency of usable sampling intervals and computation cycles to be implemented
  - Applicable area (application domain or basic software)
  - Source code type: AUTOSAR conform - not conform - not supported.
  - RAM, ROM specifically RAM margin

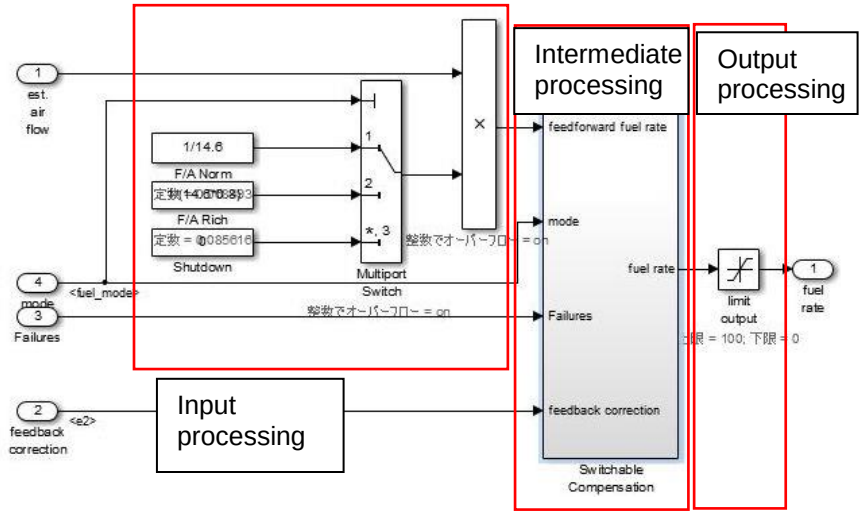
In consideration of the above, the corresponding patterns will vary depending on the use case, so we will introduce the patterns in the appendix material.

10.2.5. Modeling method for control flow layers

The arrangement of the control layer is a layer used to express all input processing, intermediate processing and output processing in one function. Significance is attached to the arrangement of blocks and subsystems. Multiple mixed small functions are grouped by dividing them between the 3 biggest stages of input processing, intermediate processing and output processing, which form the conceptual basis of control. The general configuration image is close to the data flow layer, and it is represented in a horizontal line. The difference with a data flow layer is its construction from multiple subsystems and blocks.
In control flow layers, the horizontal direction indicates processing with different significance, and blocks with the same significance are lined up vertically.



Block groups are arranged horizontally and as a whole are arranged by being given a provisional meaning.
The red borders signify the delimiter for the processing that is not visible, and the red borders correspond to objects called virtual objects. Using annotations to mark the delimiters makes it easier to understand.

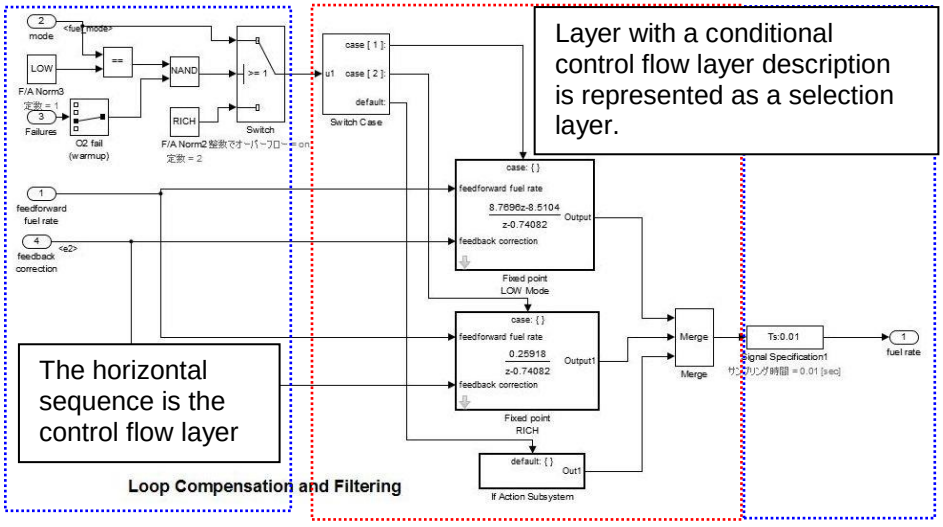


Control flow layers can co-exist with blocks that have a function.
They are positioned in the middle area between the sub-function layer and the data flow layer.
Control flow layers are used when the number of blocks becomes too large when all is described in the data flow layer and when units that can be given the minimum partial meaning are made into subsystems.
Attaching significance to the placement organizes the internal layer configuration and makes it easier to understand. It is also effective in improving maintainability by avoiding the creation of unnecessary layers.
Even if it consists of only blocks, and not a mix of subsystems and blocks, if the horizontal layout can be split into input/intermediate/output, it is a control flow layer.

10.2.6. Modeling method for selection layers

Selection layers can be written vertically or horizontally. (There is no significance to which orientation is chosen)
Selection layers are mixed with control flow layers.
Because there are switch functions for subsystems where only either one runs depending on the conditional control flow inside the red border, this is termed a selection layer. It is also described as a control flow layer because the whole lines up initial processing/intermediate processing (conditional control flow)/output processing. In the control flow layer, the horizontal direction indicates processing with different significance, and parallel processing with the same significance is lined up vertically. In selection layers, no significance is attached to the direction they are arranged in, but they show layers where subsystem groups are described where only either one runs.
Example:

- Switching of coupled functions between running upwards or downwards, changing in chronological order.
- Switching to setting where the computation switches after the first time (immediately after reset) and second time.
- Switching between destination A and destination B.

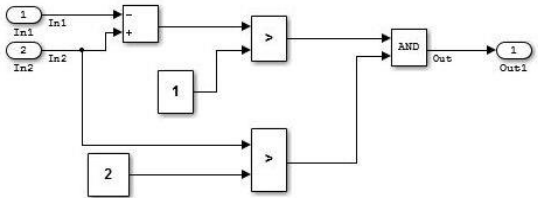


10.2.7. Modeling method for data flow layers

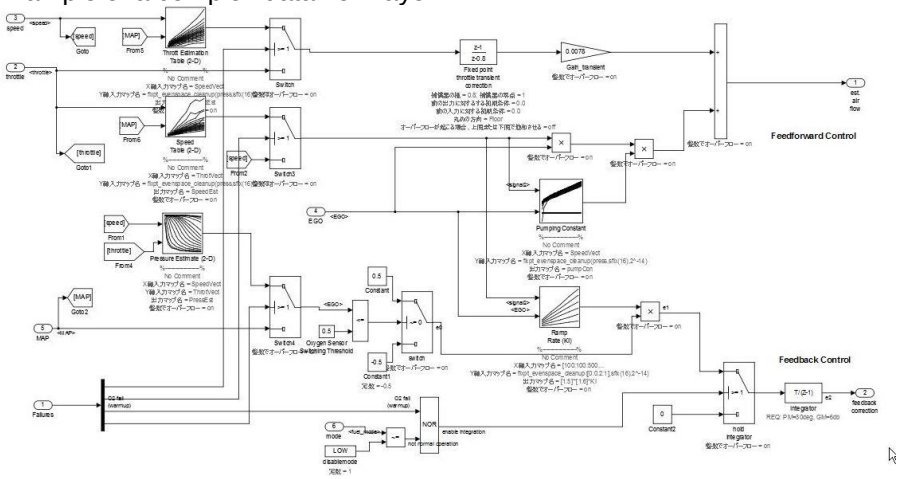
A data flow layer is the layer below the control flow layer and selection layer.
When it represents one function as a whole, and the roles of input processing, intermediate processing and output processing cannot be divided, it is a data flow layer. For instance, systems performing one continuous computation that cannot be split. Data flow layers do not permit co-existence with subsystems apart from those where exclusion conditions apply.
Exclusion conditions: Co-existence with the following subsystems is allowed.

- Subsystems where reusable functions have been set.
- Masked subsystems that are registered in the Simulink standard.
- Masked subsystems 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 in a layout as the one above, they are represented as a data flow layer.

A data flow layer becomes complicated when both the feed forward reply and the the feedback reply from the same signal are computed at the same time. Even when the number of blocks in this type of cases is a bit large, the creation of a subsystem in between should not be included in the design when the functions cannot be clearly divided. When meaning is attached through division, please design as a control flow layer.

10.2.8. Relation between embedded implementation and Simulink models

Running with the actual embedded micro controller requires embedding the C code generated from the Simulink model into the micro controller. This will substantially affect the Simulink model configuration, depending on to what extent the Simulink model will model the functions concerned, on how it is embedded, and on how the schedule on the embedded side is set.
There will be a significant effect if the tasks of the embedded micro controller to be implemented and the tasks used by the Simulink model are different.

10.3. AUTOSAR Concept

Here, we will not explain the AUTOSAR standard, but rather we will explain the concept of AUTOSAR. Users do not have to conform fully to AUTOSAR, but they must have an understanding of it and use it as a reference in modeling.

10.3.1. What is the AUTOSAR software platform concept?

When designing a control model, you must use the AUTOSAR software platform concept and examine whether the model you are designing classifies as an application or as basic software.
A model that mixes application and basic software must be split at the design stage.
The AUTOSAR software platform concept

- High capacity, low speed, regular processing is dealt with in the application layer.
- High speed or irregular driver types are dealt with in the basic software layer

The AUTOSAR software platform is represented as the configuration in Fig. 10.1.

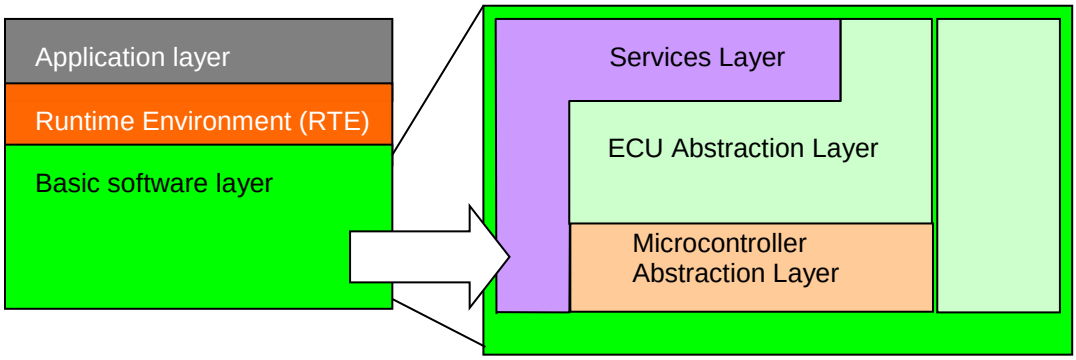


Fig. 10.1 System configuration example (see Architecture – Overview of Software Layers Top and view Coarse view
AUTOSAR Release 4.0 Document Title: Layered Software Architecture.)

For instance, in designing an engine control model, no model is built where all computations are executed with the interrupt as a base point, but computations that are shared for all cylinders are performed through regular computations in the application area. For instance, computations of the current emissions status or the target torque. And computation results, which have been computed with the application through the RTE when an irregular interrupt occurs from the basic software area, are received, and the actuator is actualy activated. It is the concept of AUTOSTAR for computations of the basic software area to be as simple as possible and for shared computation functions to be placed in the application.
When all the modeling is done in Simulink, it is advisable to have as many single computations in the interrupt area as possible. A design is required that places controls that are as simple as possible on the interrupt side, reduces the computation volume for that instant, and acquires results that, when possible, are computed at regular intervals. If possible, PID standard computations should not be included.
Functions that only execute the designated actions are ideal. However, necessary computations should

not be excluded. For instance, for fault diagnosis, computations where a conclusion must be drawn at that instant should be performed even if they are complex computations.
For those parts that run in a slower layer than the interrupt processing and receive commands to an actuator which is faster than the application execution speed, the direct execution code should not be given, but a way should be devised so that the target value or gradient until the the next command is delivered is obtained during sampling through linear interpolation.

10.3.2. RCP and AUTOSAR software platform

Modeling using devices such as RCP is pretty similar to the AUTOSAR software update in concept. Of course, generated codes do not conform to the AUTOSAR specifications. For example, I/O software of RCP allows vender-provided S-functions to be linked, and a user designs the application domain. Custom functions in the application domain and S-functions are wired in the Simulink block diagram, which does not require consciousness of interaction with RAM and so on.
The output C code runs on the real-time OS, and I/O software and applications created by Simulink are output into different source files, the real-time operation part and the part handled as interrupt are separated naturally. Users do not have to be conscious of those platforms; I/O S-function created by vender is executed when needed, and application model is modeled without consciousness of content and timing of I/O processes and behaves.

The actual control model/software which has such software structure has more advantages. Since RCP is capable of concentrate on developing application without so much regard to software structure, those use AUTOSAR software platform naturally. In other words, if your own product does not conform the AUTOSAR platform in the development using the AUTOSAR platform on RCP, you must customize the generated code and held back from sharing in the fruits of model-based development.

10.4. Single-task and multi-task

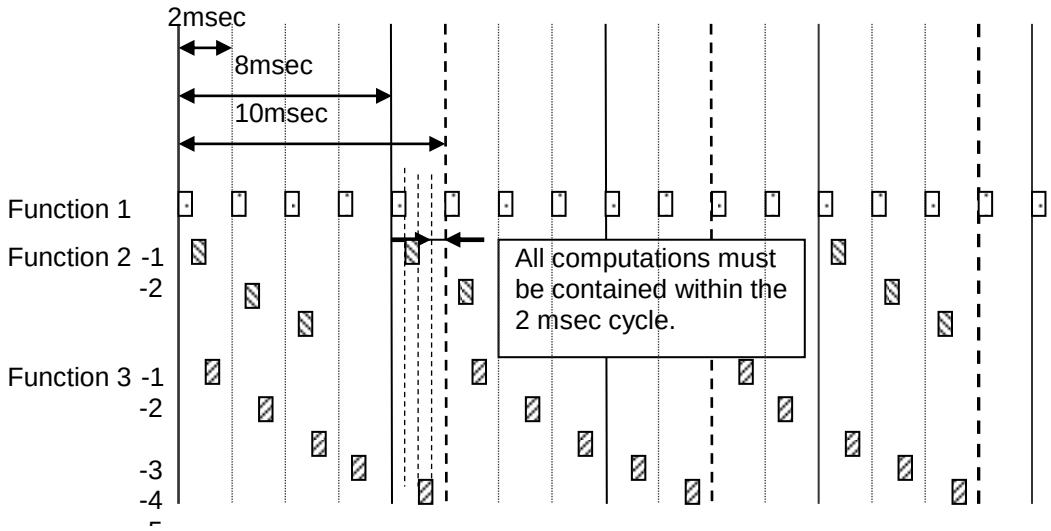
The realization method for the scheduler in embedded software has single-task and multi-task settings.

10.4.1. Single-task

For single-task, basic sampling is 2 msec, and when sampling rates of 2 msec, 8 msec and 10 msec exist within the model, pseudo sampling rates of 8, 10 msec are created in the basic 2 msec sampling rate. The execution frequency per 2 msec is counted as follows: 8 msec is executed once for every four 2 msec cycles, and 10 msec is executed once for every five. The sampling interval function specified by this frequency is executed. Attention needs to be paid to the fact that there is generally as much complex processing as functions of a lower frequency, and 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 for embedded software running in real-time, the 8 msec and 10 msec functions are in this kind of cases split into several so that all 2 msec computations are of an almost equal volume. In this way the computation volume per cycle is reduced through partitioning, and the CPU load is equally divided. For that reason the 10 msec sampling function is divided into the following 5.

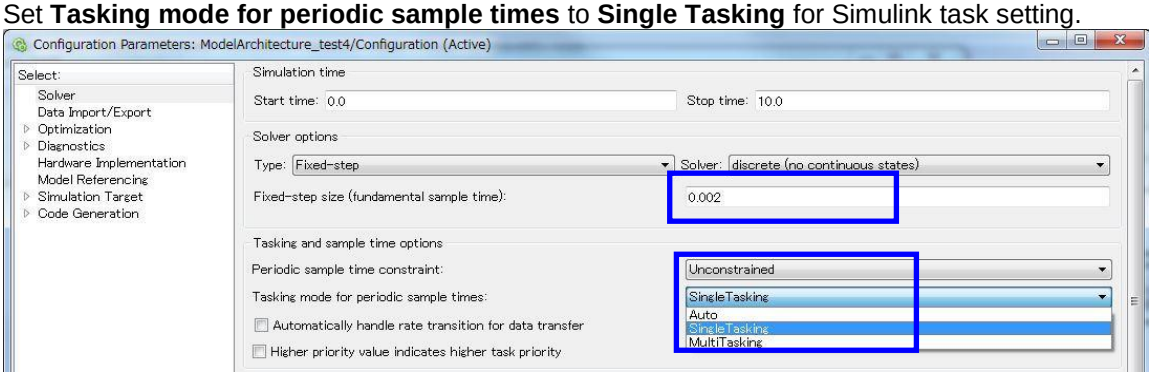
Fundamental frequency	Offset
10msec	0msec
10msec	2msec
10msec	4msec
10msec	6msec
10msec	8msec

In the same way, the 8 msec sampling function is divided into 4.
However, as equal division is not always possible, functions cannot be allocated to all cycles, but it is important to keep a uniform CPU load.

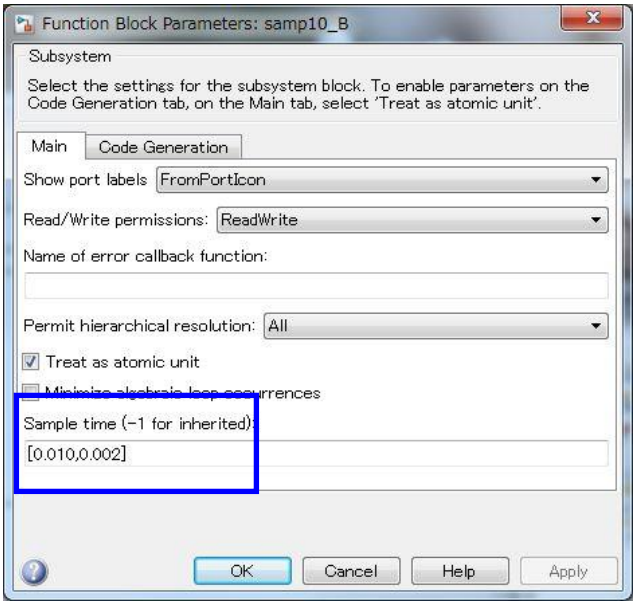


- How to set frequency-divided setting of task

Set **Tasking mode for periodic sample times** to **Single Tasking** for Simulink task setting.

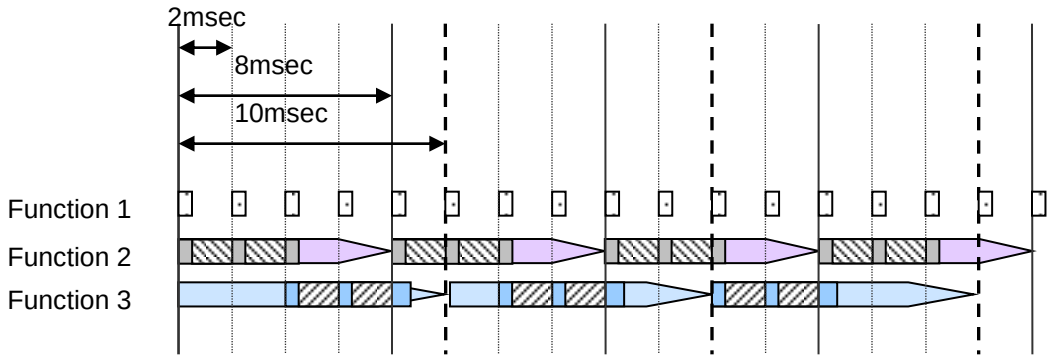


Then enter values of "*sampling period, offset*" in the subsystem's "Sample Time" setting field. A subsystem to which a sampling period can be specified is an atomic subsystem.



10.4.2. Multi-task

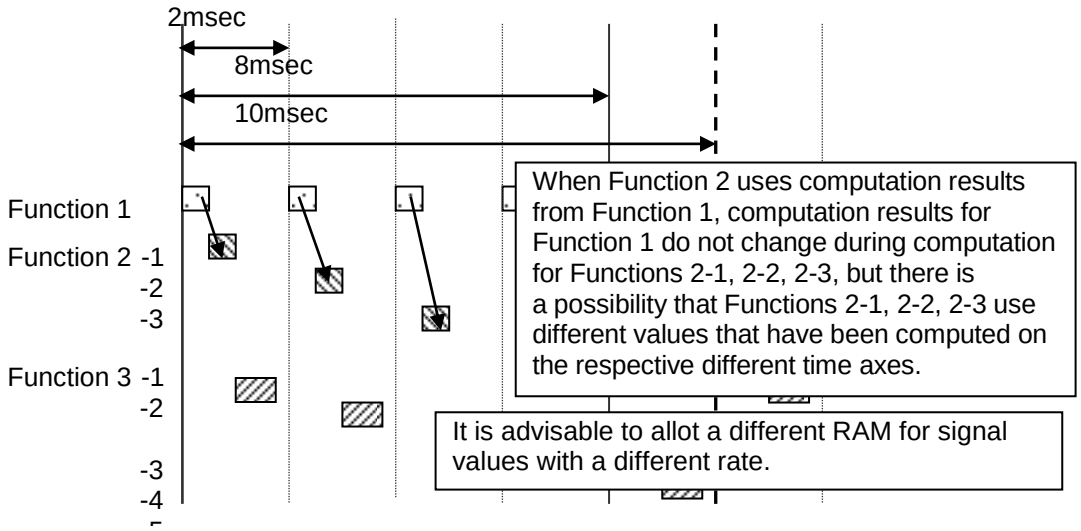
Multi-task sampling is executed using a real-time OS that supports multi-task sampling. In single-task sampling, described above, equalizing the CPU load is not done automatically, but a person divides the functions and allocates them to the appointed task. In multi-task sampling, the CPU performs the computations automatically in line with the current status, and there is no need for a person to set detailed settings. Computations are performed and results are output starting from the task with the highest priority, but task priorities are specified by a person. In most cases fast tasks are assigned highest priority.



It is considered important that computations are completed within the cycle, including slow tasks, and when a high priority computation has been processed and the CPU is freed up, the computation for the system with the next priority ranking is performed. If a high priority computation process comes in during a computation, the low priority computation is aborted and the high priority computation process is executed first.

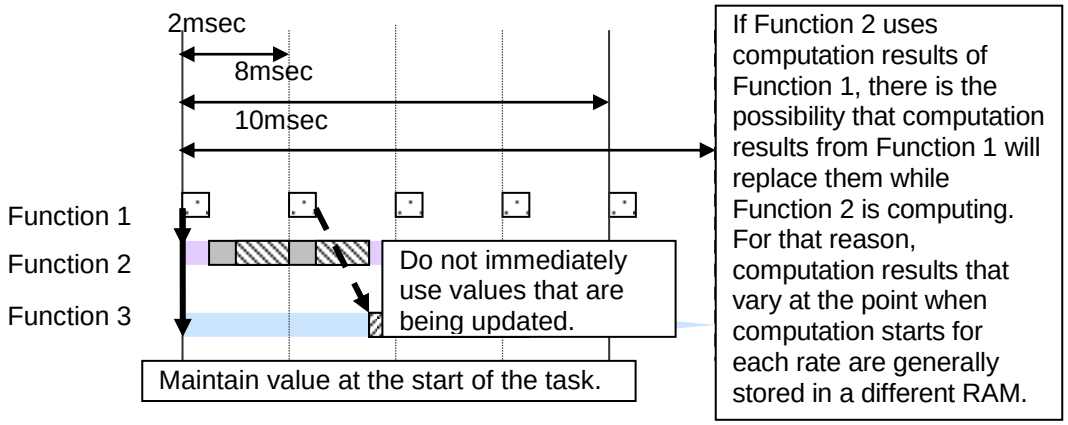
10.4.3. Effect of connecting subsystems with sampling differences

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 could change while subsystem B is computing. If values change during the process, computation results in subsystem B can result in unexpected values. For instance, a comparison is made in system B's first computation with the system A output, the result is computed with the conditional judgment based on this output, and then it is compared again at the last computation in system B. If the subsystem A output at this point is a different value, 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, if tasks generally change, output results from subsystem A are fixed immediately before they are used by subsystem B. 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. In the RTW concept using AUTOSAR, different RAMs are all defined at the receiving and exporting side.



Single-task
Signal values are the same within the same 2 msec cycle, but please note that for different 2 msec cycles the computation value is different to the preceding one. If Function 2-1 and 2-2 used signal A of Function 1, then 2-1 and 2-2 will be using results from different times, so please be aware.

Multi-task
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 a new RAM.
Before new computations are performed within the task, values are all copied in one go.



11. Simple checking sample program for guidelines

Some guideline rules allows check by setting automatic check with conditions as well as check using Model Adviser. Here show some sample programs for the method to set automatic check setting. Model-based development enables reduction of man-hour and product quality improvement using such automatic correction. It is necessary not only requiring users to keep to established rules but also improving usability by correcting bugs automatically.

11.1. Check by automatic setting

11.1.1. na_0004: Simulink model appearance settings

```
SettingItems= {...
        ...% Display option
         'ModelBrowserVisibility',       'off' 'browser display', ...
         'ScreenColor',                  'white' 'screen color',...
         'StatusBar',                    'on' 'status bar',...
         'ToolBar',                      'on',  'toolbar',...
         'ZoomFactor',                   '100' 'zoombar', ...
        ...% port display option
         'ShowPortDataTypes',            'off',   'port data types';...
         'ShowLineDimensions',           'off',   'signal dimensions';...
         'ShowStorageClass',             'off',  'storage class';...
         'ShowTestPointIcons',           'on',   'testpoint indicator';...
         'ShowSignalResolutionIcons',    'on',   'testpoint indicator';...
         'ShowViewerIcons',              'on',   'viewer indicator';....
         'WideLines',                    'on',   'display wide lines for non-scalars';...
         };
for k=1: size(SettingItems,1)
            set_param(0,SettingItems{k,1},SettingItems{k,2})
            set_param(bdroot,SettingItems{k,1},SettingItems{k,2})

end
```

set_param(0 is setting for Simulink. If it is applied, the setting above is inherited to the newly created model files. The setting is enabled only after it is rerun during Simulink restart. The setting is executed during Simulink restart by describing it to the startupsl.m file on the path.
To change the settings of existing file, use **set_param(bdroot, SettingItems{k,1},SettingItems{k,2})**.

Reference: Simulink/modeling/model configuration/block/model parameters

11.1.2. db_0043: Model font and font size

```
SettingItems= {...
         ...%% font setting
         ...% block default setting
         'DefaultBlockFontName',   'MS UI Gothic', 'default block font name'; ...
         'DefaultBlockFontSize',   12,             'default block font size', ...
         'DefaultBlockFontWeight', 'normal'          'default block font thickness'; ...
         'DefaultBlockFontAngle',  'normal',       'default block font tilt', ...
         ...% default line font settings
         'DefaultLineFontName',    'MS UI Gothic', 'default line font name' ; ...
         'DefaultLineFontSize',    12,             'default line font size'; ...
         'DefaultLineFontWeight',  'normal',       'default line font weight'; ...
         'DefaultLineFontAngle',   'normal',       'default line font tilt; ...
         ...% default annotation font settings
         'DefaultAnnotationFontName',   'MS UI Gothic',  'default annotation font name; ...
         'DefaultAnnotationFontSize',   14,              'default annotation font size'; ...
         'DefaultAnnotationFontWeight', 'normal',        'default annotation font weight'; ...
         'DefaultAnnotationFontAngle',  'normal',        'default annotation font orientation'; ...
         };

for k=1: size(SettingItems,1)
            set_param(0,SettingItems{k,1},SettingItems{k,2})
            set_param(bdroot,SettingItems{k,1},SettingItems{k,2})
end
```

Executing set_param(bdroot, SettingItems{k,1},SettingItems{k,2}) does not change entirely. To change file content entirely including content manually modified, using find_system is required to search all information within the model file to change, however, it may change the intendedly modified description. To avoid this, it is recommended to complete settings in the stage of new creation.

11.1.3. na_0001: Bitwise Stateflow operators

The following is an example of changing the settings of a Stateflow Chart contained in the existing model.

```
rt = sfroot;
modelH = get_param(bdroot, 'Handle');
  rt = rt.find('-isa', 'Simulink.BlockDiagram', '-and', 'handle', modelH);
result = rt.find('-isa', 'Stateflow.Chart');
if ~isempty(result)
  for n1=1:length(result)
   result(n1).EnableBitOps=true;
  end
end
```

12. Update history

■ Update time and date

Date	Change
02.04.2001	NAMAAB Initial document Release, Version 1.0(Eng)
xx.04-2003	JMAAB Initial document Release,Version 1.0(Jp)
04.27.2007	MAAB Version 2.0 Update release(Jp&Eng) This document is a collaboration of JMAAB and NAMAAB.
07.30.2011	Version 2.2 Update release(Eng)
08.31.2012	Version 3.0 Update release(Eng)
05.30.2013	Version 3.0 Japanese localization(Jp)
31.03.2015	Version 4.0 Update release(Jp&Eng)
19.06.2015	Version 4.01 correct (Jp&Eng)

12.1. Termination rule

12.1.1. Removed in version 2.2

JM_0013: Annotations: The rule was original written due to a printing bug in R13. The bug was fixed in R14 SP1.

12.1.2. Removed in version 3.0

No guidelines were removed in version 3.0

12.1.3. Removed in version 3.1

No guidelines were removed in version 3.1

12.1.4. Removed in version 4.0

- Removed after being integrated to another rule or altered

Integration source ID	Supporting ID
jc_0221: Sentences that can be used for the name of the signal line	jc_0222
na_0030: Sentence that can be used for a Simulink path name	
jm_0010: Names of Import block/Outport block	na_0005, jc_0082, jc_0083
jc_0081: *Icon display* of Inport block/Outport lbock	
db_0148: Transition condition pattern of the flow chart	jc_0742
db_0150: Transition condition pattern of the state	
db_0149: Condition action pattern of the flow chart	jc_0743
na_0019: Restricted Variable Names	jc_0251

- Deleted because it became unnecessary with the recent year's MATLAB version
  jc_0541: Usage of adjustable parameter at Stateflow

- Deleted because contents are not rule.
  db_0133: Usage of a flow chart pattern
  (It is covered by db_0132,jc_0770,jc_0771,db_0134,db_0159 and db_0135)

12.1.5. Moved to attachment in version 4.0

They were moved to appendix. And their IDs were deleted.

- Since they are not guideline rules but how to think, they were moved to appendix.

db_0040: Hierarchy structure of the model
jc_0301: Controller model
jc_0311: Top layer/root level
jc_0321: Trigger layer
jc_0331: Structure layer
jc_0341: Data flow layer

- Rules which explain functions of Simulink were moved to appendix.

na_0032: Use of merge blocks
jc_0021: Model diagnostic settings
jc_0351: Methods of initialization

- Since rules about development process are not treated in this guidelines, they were moved to appendix

na_0026: Consistent software environment
na_0027: Use of only standard library blocks

12.2. The flow of the style guideline revision