Model Design Guidelines

*Guidelines Team*:

Chinnakolandai, Praveenkumar – ASW BRAKES ([Praveenkumar.Chinnakolandai@wabco-auto.com](mailto:Praveenkumar.Chinnakolandai@wabco-auto.com))

Velraj, Devanand – ASW SUS ([Devanand.Velraj@wabco-auto.com](mailto:Devanand.Velraj@wabco-auto.com))

Balasubramani, Arunchandran – ASW BRAKES ([Arunchandran.Balasubramani@wabco-auto.com](mailto:Arunchandran.Balasubramani@wabco-auto.com))

Ganesan, Karthikeyan – ASW BRAKES ([Karthikeyan.Ganesan@wabco-auto.com](mailto:Karthikeyan.Ganesan@wabco-auto.com))

Chamarthi, Harshini – ASW BRAKES ([Harshini.Chamarthi@wabco-auto.com](mailto:Harshini.Chamarthi@wabco-auto.com))

*Reviewed by:*

Jakobs, Holger – Technology team ([holger.jakobs@wabco-auto.com](mailto:holger.jakobs@wabco-auto.com))

Pavel, Jelinek – Technology team ([pavel.jelinek@wabco-auto.com](mailto:pavel.jelinek@wabco-auto.com))

Table of Contents

[1. Scope 4](#_Toc484791091)

[2. Purpose 4](#_Toc484791092)

[3. Acronyms 4](#_Toc484791093)

[4. Points to note: 4](#_Toc484791094)

[5. SW detailed design specification in MKS RM 4](#_Toc484791095)

[6. Interface specification in MKS RM 4](#_Toc484791096)

[7. MKS Project structure creation 5](#_Toc484791097)

[7.1.1. Pre-requisite 5](#_Toc484791098)

[7.1.2. Guidelines 5](#_Toc484791099)

[8. Behavior modeling in Simulink/Stateflow 6](#_Toc484791100)

[8.1. Component model design 6](#_Toc484791101)

[8.1.1. Purpose 6](#_Toc484791102)

[8.1.2. Pre-requisite 6](#_Toc484791103)

[8.1.3. Guidelines for component model design 6](#_Toc484791104)

[8.1.3.1. General guidelines 6](#_Toc484791105)

[8.1.3.2. Integration of modules 7](#_Toc484791106)

[8.1.3.3. Efficient interface handling 9](#_Toc484791107)

[8.1.3.4. Specify SL data types at component library interface. 10](#_Toc484791108)

[8.1.3.5. Execution sequence of modules 11](#_Toc484791109)

[8.1.3.6. Common libraries for more than one module (within component): 11](#_Toc484791110)

[8.1.3.7. Common libraries for more than one module (across components): 11](#_Toc484791111)

[8.1.3.8. Definition of states for component level interface signals 12](#_Toc484791112)

[8.1.3.9. AUTOSAR frame model integration 12](#_Toc484791113)

[8.1.3.9.1. Naming convention for FiM and DEM ports 12](#_Toc484791114)

[8.1.3.10. Creation of test environment for component 13](#_Toc484791115)

[8.1.3.11. Use of test trigger 14](#_Toc484791116)

[8.1.3.12. Pre-Checkin of the test environment 15](#_Toc484791117)

[8.1.3.13. Variant handling in Simulink 16](#_Toc484791118)

[8.2. Module Model design 19](#_Toc484791119)

[8.2.1. Purpose 19](#_Toc484791120)

[8.2.2. Pre-requisite 19](#_Toc484791121)

[8.2.3. Guidelines for module design 19](#_Toc484791122)

[8.2.3.1. General guidelines 19](#_Toc484791123)

[8.2.3.2. Naming of module interfaces 19](#_Toc484791124)

[8.2.3.3. Generation of module frame from interface specification 20](#_Toc484791125)

[8.2.3.4. Using Simulink or Stateflow for module design 21](#_Toc484791126)

[8.2.3.5. Signal Naming 22](#_Toc484791127)

[8.2.3.6. Specifying SL data types 23](#_Toc484791128)

[8.2.3.6.1. Specifying data types at the library interfaces 23](#_Toc484791129)

[8.2.3.6.2. Specifying data types for Simulink bus port 24](#_Toc484791130)

[8.2.3.6.3. Virtual and non-virtual bus in Simulink 26](#_Toc484791131)

[8.2.3.6.4. Specifying data types for blocks 27](#_Toc484791132)

[8.2.3.7. Scalability of model for different vector/array sizes 28](#_Toc484791133)

[8.2.3.8. Design of initialization functions 30](#_Toc484791134)

[8.2.3.9. Modeling of enabled/triggered subsystem 31](#_Toc484791135)

[8.2.3.10. Virtual and non-virtual (atomic) subsystems 31](#_Toc484791136)

[8.2.3.11. Maximum depth level of subsystems 31](#_Toc484791137)

[8.2.3.12. Restrictions in for-subsystems usage 31](#_Toc484791138)

[8.2.3.13. Explicit division by zero protection 34](#_Toc484791139)

[8.2.3.14. Float Comparison 34](#_Toc484791140)

[8.2.3.15. Proper handling of counter overflows or underflows 35](#_Toc484791141)

[8.2.3.16. Creation of module test environment 35](#_Toc484791142)

[8.2.3.17. Usage of blocks 36](#_Toc484791143)

[8.2.3.18. Adherence to naming guidelines 38](#_Toc484791144)

[8.2.3.19. Stateflow design 39](#_Toc484791145)

[8.2.3.19.1. Stateflow with states 39](#_Toc484791146)

[8.2.3.19.1.1. States and transitions 39](#_Toc484791147)

[8.2.3.19.1.2. Interface signal naming 41](#_Toc484791148)

[8.2.3.19.1.3. Basic usage of states 42](#_Toc484791149)

[8.2.3.19.1.4. Default transitions 45](#_Toc484791150)

[8.2.3.19.1.5. State transition rules 45](#_Toc484791151)

[8.2.3.19.1.6. Avoid using “Entry”, “During”, “Exit” state action keywords 52](#_Toc484791152)

[8.2.3.19.1.7. Flow within states 52](#_Toc484791153)

[8.2.3.19.2. State flow without states (flow charts) 54](#_Toc484791154)

[8.2.3.19.3. Stateflow with graphical functions local to a chart 56](#_Toc484791155)

[8.2.3.19.4. Stateflow with global graphical functions 57](#_Toc484791156)

[8.2.3.19.5. Vector signal processing in Stateflow 58](#_Toc484791157)

[8.2.3.19.6. Usage of intrinsic operators 59](#_Toc484791158)

[8.2.3.19.7. Calibration parameters inside Stateflow 59](#_Toc484791159)

[8.2.3.20. Adherence to coloring guidelines 63](#_Toc484791160)

[8.2.3.21. Adherence to standard modeling guidelines 63](#_Toc484791161)

[8.3. MKS revision inside models 65](#_Toc484791162)

[8.4. Linking of detailed design specification in MKS to the model 65](#_Toc484791163)

[8.5. Guidelines for SL-TL handshake 66](#_Toc484791164)

# Scope

This document covers only guidelines related to Simulink or Stateflow model design from detailed design specification

Guidelines related to TargetLink code generation and testing are not in the scope of this document.

# Purpose

This document contains Model Based Design guidelines for modeling a component and module using Matlab, Simulink, Stateflow tools.

# Acronyms

AUTOSAR – Automotive Open System Architecture

SWC – Software Component

SL – Simulink

TL – TargetLink

mBSP - modular Braking System Platform

FACT - Functional Architecture Construction Tool

# Points to note:

Some guidelines highlighted in yellow background are still under discussion.

# SW detailed design specification in MKS RM

* As a prerequisite, a SW detailed design specification (for component & modules underneath) should be available in the MKS RM before component or module design
* The design specification is the basis for creating module test cases & module testing and hence the specifications shall be testable.
* The detailed design document shall be inspected before start of modelling.
* Grouping of design specification as per the functionality will be helpful in dividing the component into modules.

# Interface specification in MKS RM

* The component and module interfaces shall be defined in the respective detailed design specification of the component or module.
* The interface specification shall contain:
  + - * + Name of the signal (as per mBSP naming convention)
        + Description of the signal
        + Type, range and resolution of the signal

Refer the naming convention for mBSP project

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Documents/Guidelines/naming_rules/project.pj&revision=:member&selection=MBSP_NamingConventions.xls>

* The SWC interface specification will be transferred into any of the SW architecture tool FACT, Enterprise Architect etc. for further processing
* The Module level interface specification shall contain additional attributes for TL frame generation.

Refer the below link for detailed information:

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Misc/XLS2DD/project.pj&revision=:member&selection=Workflow_RM_IfSpec.docx>

# MKS Project structure creation

Once the detailed design of the component is completed then the project structure in MKS shall be created for the version management of the model, scripts, documents etc.

# Pre-requisite

1. Detailed design specification of component

# Guidelines

* All the components/modules/sub-modules shall follow the project structure guidelines specified for the project
  + It is recommended to avoid sub-modules in possible cases because it increases the complexity and makes the maintenance more difficult.
  + Usage of sub-modules will increase the path length of blocks especially state charts which causes certain errors due simulation/code generation
* The MKS project structure for component/module/sub-module shall be created automatically using a python script.

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MKS/MindMap_based_SI_Project_Creation/Manual/project.pj&revision=:member&selection=MM_MKS_ProjectCreation_UserManual.pdf>

* The depth of folder structure shall not go beyond sub-modules

*SWC*

*- Module1*

*- Module2*

*- Sub-Module1*

*- Sub-Module2*

* MKS project structure creation right is available only for the project admins. A request shall be sent to one of the project admin for creation of folder structures.
* Proper naming guidelines shall be followed in naming the component/ module/ sub-modules. Too long names shall not be used. Please refer the below link for naming abbreviations are per AUTOSAR standard.

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Documents/ModellingGuidelines/project.pj&revision=:member&selection=Autosar.NamingConventions.xls>

# Behavior modeling in Simulink/Stateflow

The functional requirements are realized by using Simulink or Stateflow. Based on the functions the SW application is divided into SW components (SWC) and further into SW modules (further into sub-module only if the module is too complex to test or maintain) as specified in the detailed design document.

SW Application

Module1

SWC\_2

Module1

SWC\_1

Module2

Sub module2

Sub module1

# Component model design

# Purpose

The main purpose of component design is to show the signal flow and interaction between the underlying modules.

# Pre-requisite

* Project structure available in MKS SI
* Detailed design specification of component
* Interface specification for the component in MKS RM

# Guidelines for component model design

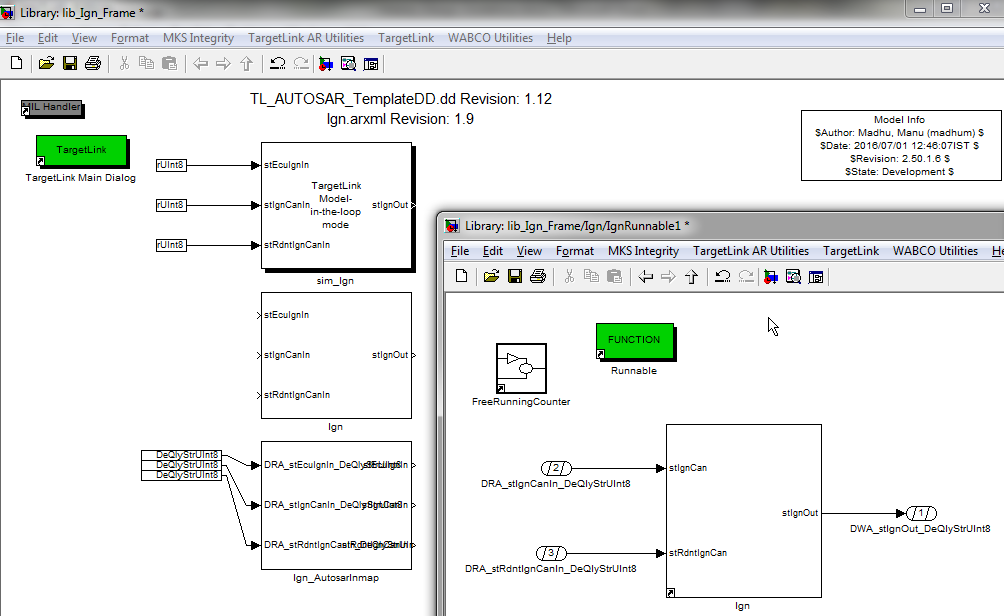
# General guidelines

* The interfaces of a component shall be available in the form of interface specification(In MKS RM)
* Each component shall be modelled as a library.
* The component library shall be independent of the inter-component data exchange layer implementation. (Inter-component data exchange layers are *Integration Layer (IL)*, *Thread Data Manager (TDM) of CESAR*, *Run-Time Environment (RTE) of AUTOSAR*).

The advantage is that the components can be used in any of the above data exchange layer concepts in future without any modification.

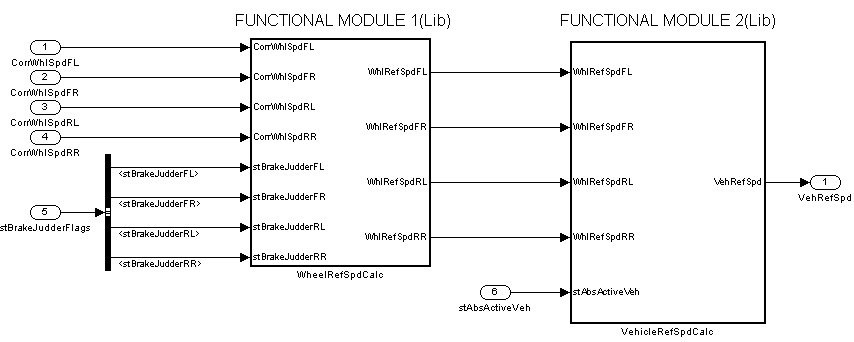
For example: in AUTOSAR based SWCs, the AUTOSAR frame containing the runnable shall be stored as *lib\_<SWC>\_frame.mdl* and the component library shall be stored as *lib\_<SWC>.mdl*

* For different data exchanges, a wrapper shall be implemented for conversion



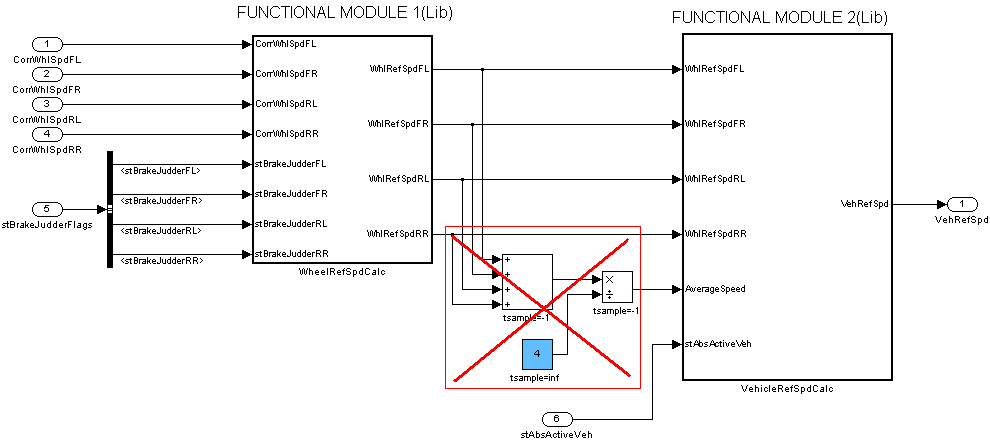
# Integration of modules

* The component shall be divided into functional modules. Grouping blocks into modules which doesn’t signify any functionality shall be avoided.



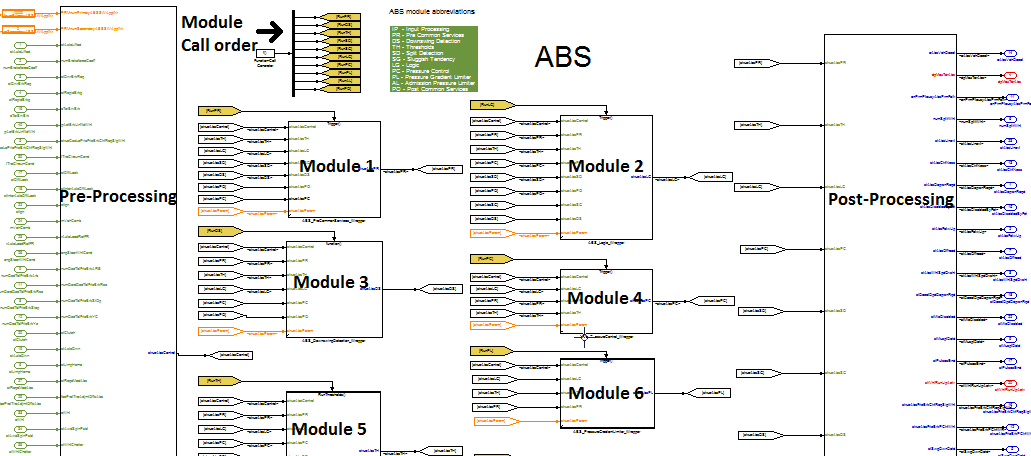
*Pros:*

* *Easy to visualize the functionalities inside a component*
* *Easy to visualize the interaction of the functional modules*
* The component shall contain only integration of the functional modules and shall not contain any free blocks in between the functional modules



*Cons:*

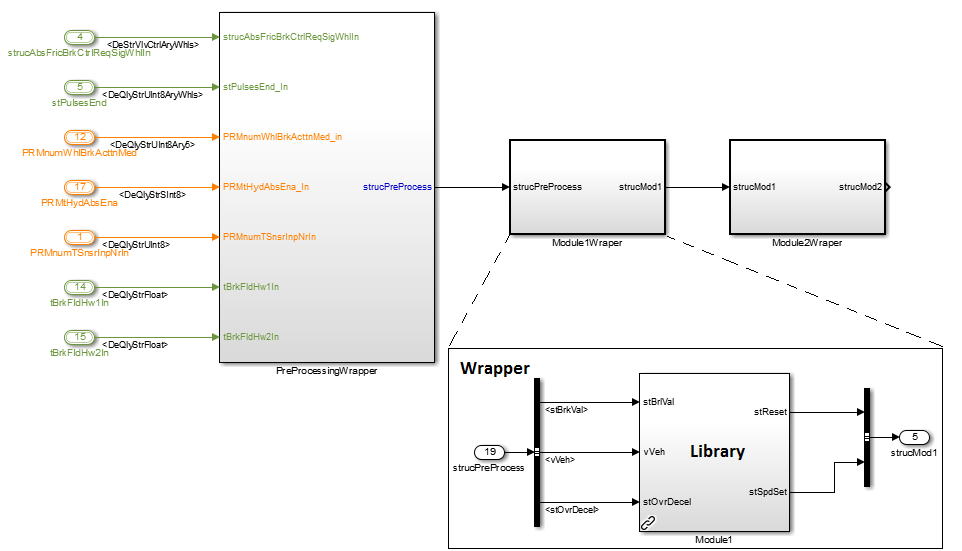
* + *In the above case FUNCTIONAL MODULE2 cannot reused as such. It requires logic for calculating “AverageSpeed” signal to be implemented where ever the module is used.*
  + *The testing of the free blocks will be not be covered by any of the above module testing. Separate testing needs to be done.*
* Other than functional module libraries, the component library shall contain modules for *pre-processing of input signals, post processing of output signal, scheduling and call order modules*(refer pic below)
* Pre-processing module shall include quality checks, parameter processing, range check, grouping of input signals, unit conversion etc.
* Post-processing module shall include output signal bus structure creation etc.



* Allowed blocks at the component level apart from module libraries are:
  + Unit-delay
  + From-Goto blocks
  + Bus selectors
  + Bus creators
  + Blocks for scheduling logic

# Efficient interface handling

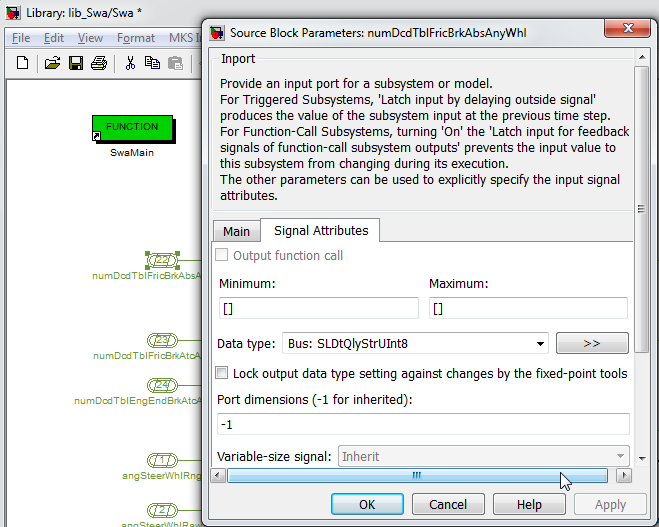
* It is recommended to have all module library interfaces as individual signals. No user-defined bus structures shall be part of module interfaces except AUTOSAR quality structures.
* User defined structures shall be used only in a wrappers which should be a part of the model integration level (Component/module).
* User defined structures shall have only Simulink bus ports with bus creators/ bus selectors.



* The above method of interface handling has the below advantages over other methods:
  + In case of user-defined TL bus structures, even if the signal is not used inside a module, changes in the structure elements requires:
    - * Rescan of bus ports
      * Update of respective test environments
      * Code generation and testing

# Specify SL data types at component library interface.

* Inports and Outports of a component library shall be specified with the required data type explicitly.
* If the component level library is integrated within an AUTOSAR frame, then the Simulink equivalent of the AUTOSAR types at the runnable level shall be specified as data type for the component level library interfaces using bus objects.



Refer section 8.2.3.7 for more details regarding Simulink bus objects

# Execution sequence of modules

* If required, explicit call order or execution sequence of the modules shall be modelled using Stateflow charts. Refer link for example:

<http://wdesxwiki/dokuwiki/doku.php?id=process:mbe:sl_bestpractises#definition_of_execution_sequence>

* If calling order is implicitly taken care by signal flow, then no explicit scheduling logic shall be required.

# Common libraries for more than one module (within component):

* If a common functionality is required for more than one module, then the functionality shall be created as a separate library.
* The common library shall be maintained as separate testable module based on the model complexity. If the common library is less complex then it shall be tested along with the module where it is referred.
* If the functionality is complex then the library shall be maintained and tested as separate module.
* Rules for creating common libraries:

1. Common libraries shall be named as lib\_<Swc>CmnFct<Functionality>.mdl
   * e.g: lib\_AbsCmnFctValidityChk.mdl
2. Common libraries shall be stored in component level “Lib” folder
3. Separate library model shall be created for individual functionality/logic
4. Two or more common functions shall not be a separate module

# Common libraries for more than one module (across components):

* It is required to have common libraries that are used across components. The reuse of libraries across components shall be achieved by using appropriate project structure in configuration management along with project settings.
* Refer below link for the list of available generic libraries.

<http://wdesxwiki/dokuwiki/doku.php?id=process:mbe:sl_bestpractises#mbsp_common_functions>

# Definition of states for component level interface signals

* Signal states of component interfaces shall be specified in the global initialization script. This is helpful in single place maintenance and ensures consistency of the states across the components for the same signal
* In mBSP project, the global constants / enumeration are maintained the below excel sheet

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Simulink_models/CommonMbsp/Common/Config/project.pj&revision=:member&selection=mbsp_global_const.xlsx>

* From the above excel sheet global ini.m, DD constants are generated using the below m-script

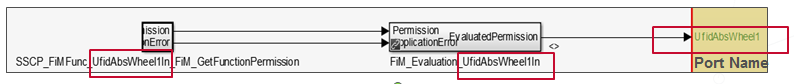
<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Include/Common/Config/project.pj&projectRevision=1.49&revision=1.2&selection=fcn_generateGlobals.m>

# AUTOSAR frame model integration

* For AUTOSAR compliant projects, if the AUTOSAR frame model is available early then component library shall be integrated before proceeding with further model development. (Refer [TargetLink guidelines](http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Misc/GuidelineDocuments/project.pj&revision=:member&selection=TargetLink_Guidelines.docx) document)

# Naming convention for FiM and DEM ports

* The FIM Port Name which is used at the component library interface should match with the name mentioned in the FIM blocks as shown above.



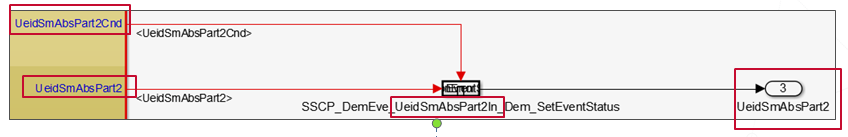
In the above screenshot:

FiM permission block name is SSCP\_FiMFunc\_UfidAbsWheel1In\_FiM\_GetFunctionPermission

FiM evaluation block name is FiM\_Evaluation\_UfidAbsWheel1In

Hence the in-port name inside the component library also should be UfidAbsWheel1.

* In the case of DEM, we have two ports. One is the actual DEM port and another is the DEM enable port. The DEM condition is connected to enable port of DEM Block.



In the above screenshot:

DEM block name is

SSCP\_DemEve\_UeidSmAbsPart2In\_Dem\_SetEventStatus

Hence the DEM port name should be UeidSmAbsPart2In

The DEM condition port name in the component library should be UeidSmAbsPart2Cnd.

* The naming convention will helps in connecting FiM & DEM ports automatically during AUTOSAR frame integration.

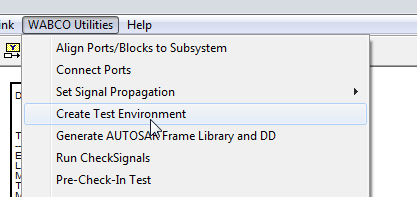
# Creation of test environment for component

* For the compilation and testing of the component/module library, a test environment is required.
* Basic test environment for a SWC is available along with the AUTOSAR frame model generation

Refer [TargetLink guidelines](http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Misc/GuidelineDocuments/project.pj&revision=:member&selection=TargetLink_Guidelines.docx) document:

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Misc/GuidelineDocuments/project.pj&revision=:member&selection=TargetLink_Guidelines.docx>

* After integrating the module level libraries into the component frame the component test environment shall be compilable with default input signal values.
* Then using below option, test environment for the selected subsystem (module libraries) shall be created automatically.



* The output of this script is a compilable test environment model along with default test vector XLS sheet.
* Before generation of the module test environment, it shall be ensured that correct SL types are specified for the Inports, Outports of the module library.

*(Future steps: Creation of module test environment from XLS2DD script)*

# Use of test trigger

* Test trigger will load the test data into Matlab workspace only during initialization. Since the test data is not stored along with the model the size of the \*.mdl file is less.
* The test data can be stored in .m .mat .xls .xlsx, .csv formats.
* The default and the most common format is \*.xlsx. The other formats enable the use data logs from vehicles for simulation. \*.m format shall be used for special functions like sine/cosine etc.

(Refer the m-template from test trigger for more info.)

Refer the below link for more details about test trigger:

[\\wdegsdata2\matlab$\Trainings\TrainingVideos\TestTrigger\](file:///\\wdegsdata2\matlab$\Trainings\TrainingVideos\TestTrigger\)

# Pre-Checkin of the test environment

Every time a model is changed there are certain checks needs to run on the model before its being checked-in in MKS. These checks shall be done automatically using the “Pre-CheckIn” button in the test environment model. The checks can be run by double-clicking the block.

Below are checks which are currently included in the “Pre-Checkin” and their purpose.

*List as on 1st Nov 2016. List is subject to update*

|  |  |  |
| --- | --- | --- |
| **Sno** | **Check for** | **Purpose** |
| 1 | Disabled library links | There should not be any disabled library link.   * *If a model contains disable links then the changes in done in the model is not consistent with the linked library which is checked in along with the model* |
| 2 | Magenta blocks checks | There should not be any magenta blocks   * *As per the coloring guideline “magenta” background colored blocks indicate that the block needs to be prepared for TargetLink code generation. Once the TL properties are assigned the blocks color needs to be changed to default white background color* |
| 3 | TL-SL mismatches | There should not be any TL-SL datatype mismatches in any of the blocks.   * *If there is mismatch between the Simulink and TargetLink datatypes then this would result in MIL-SIL mismatches in module testing* |
| 4 | Missing DD entries | All TL ports, constant blocks (with some exceptions), calibration parameters shall be linked to the DD   * *The code generation properties can be easily maintained in DD rather than in the model itself* |
| 5 | Coloring guidelines check | All model blocks shall follow the coloring guidelines specified for the project.   * *Coloring guidelines for port, constant, calibration parameters will ensure better readability of the model* |
| 6 | Constant blocks values check | The numeric constants used in the model shall not have values other that -1, 0, 0.5, 1, 2   * *Magic numbers shall not be used as constant value. Instead all constant values other than above mentioned shall be defined in the ini.m file with a meaningful name. This improves the understandability of the functionality* * *Above mentioned numeric constants shall not be used for enumerations. It is mandatory to define all constants for enumerations in the ini.m with a proper naming.* |
| 7 | Missing interface SL datatypes | All library interfaces/ports shall be assigned to proper SL datatypes.   * *Interfaces with explicit data types specified will avoid wrong datatype inheritance from other libraries. If wrong datatypes are inherited then the functionality may not behave as expected* |
| 8 | TargetLink variable width not empty if class is FCN\_IN\_SCALAR | There shall not be variables in DD with variable class “FCN\_IN\_SCALAR” and width as 1.   * *This above case will result in generated code which causes ECU trap* |

The library block for “Pre-checkin” can be found in the below link:

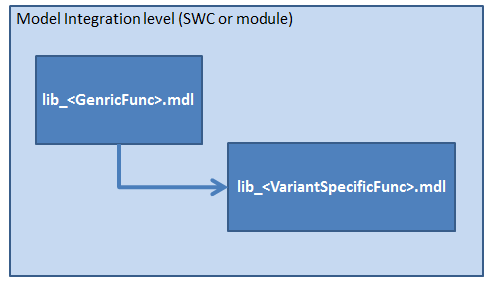
<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Misc/PreCheckInTest/project.pj&projectRevision=:member&revision=:member&selection=ButtonModelReviewScripts.mdl>

# Variant handling in Simulink

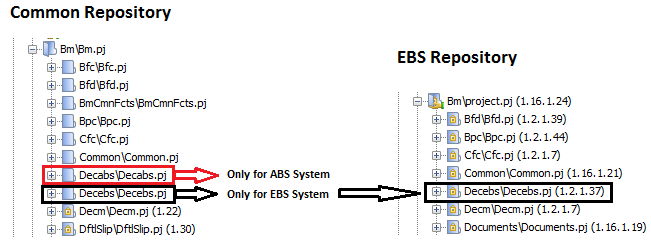
* Depending on the complexity of the logic the variant handling method shall be chosen.
  + Variant handling using library model:
    - This method shall be used for variant handling of complex logics which contains more than 10 blocks or blocks equivalent to 5 lines of executable source code.

Implementation method:

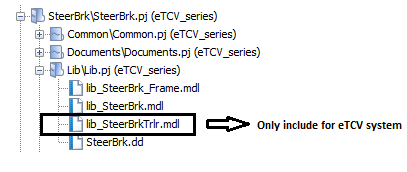
* + - All variant specific model design shall be part of a separate library (lib\_<VariantSpecificFunction>.mdl)
    - All generic model design common for all variants shall be part of a separate library (lib\_<GenericFunction>.mdl)
    - During model integration at the component or module level, the generic model library and respective variant specific model library shall be integrated.



* + - All ports of generic library which are not used by a variant specific library shall be terminated at the integration level
    - Based on the complexity, the variant library shall be treated as separate module with all quality documents or as a standalone library model tested at the higher level of integration.
    - In case of variant as separate module, the sub-project shall be available in the common repository and shared to variant specific repository (Refer below)



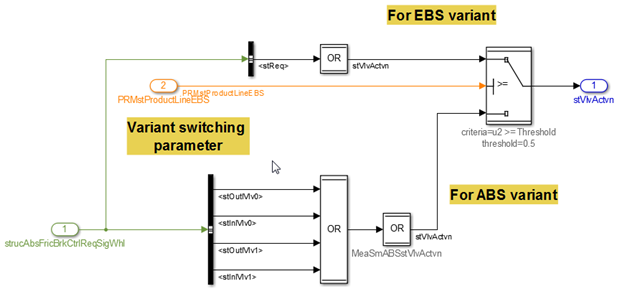
* + - In case of variant as standalone library model (Refer below)



* + - The MKS sub-projects of the variant shall contain only generic function library model and the respective variant specific library model.
  + Variant handling using run-time logic switching.
    - This method shall be used only functional implementation with less than 10 blocks or blocks equivalent to 5 lines of executable source code.

Implementation method:

* + - The variant switching shall be based on a run-time signal, EEPROM parameter or CALIBRATION parameters
    - Proper comments or highlighting shall be used to distinguish the logic used for different variants.
    - Use of subsystems is recommended to group the variant specific blocks.



# Module Model design

# Purpose

The purpose of this section is to provide how to do a module level design from module detailed design specification.

If draft model is available from concept phase, then the model shall be analyzed and restructured, if necessary into functional subsystems/sub-modules/modules.

# Pre-requisite

* Project structure available in MKS SI
* Module detailed design specification
* Module interface specification document

# Guidelines for module design

# General guidelines

* Each module shall be modelled as a library
* All the functionalities inside a module shall be self-contained. This enables the portability of the module
* Each module, and if possible each subsystem inside a module, shall represent a functional behavior.
* Every subsystem shall have a short but descriptive comment
* A module shall be divided into sub-modules based on the complexity of the functionality. For small functions, sub-modules are not required since this will increase effort for maintenance.
* All blocks shall be modelled inside a functional subsystem. There shall not be free blocks between the subsystems other than the below blocks.
  + Unit-delay
  + From-Goto blocks
  + Bus selectors
  + Bus creators
  + Selectors
  + Block for scheduling logic
* At any level of the module, either subsystem/state chart interconnection or block implementation shall be visible; a combination of blocks and subsystem shall not be used.

# Naming of module interfaces

* Module interface signals shall follow same naming convention (signal abbreviations) followed for component interfaces.
* Number of characters in a signal name / port name shall be limited to 63: due to limitations in logging in test environment
* Refer below link for more details regarding naming rules.

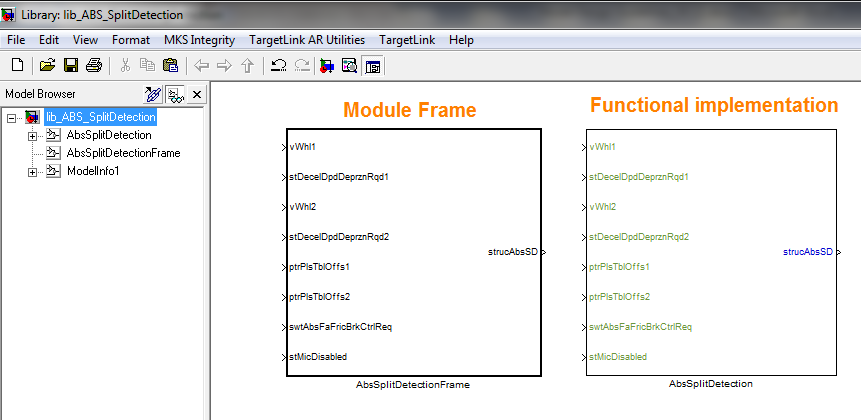
<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Documents/Guidelines/naming_rules/project.pj&revision=:member&selection=MBSP_NamingConventions.xls>

# Generation of module frame from interface specification

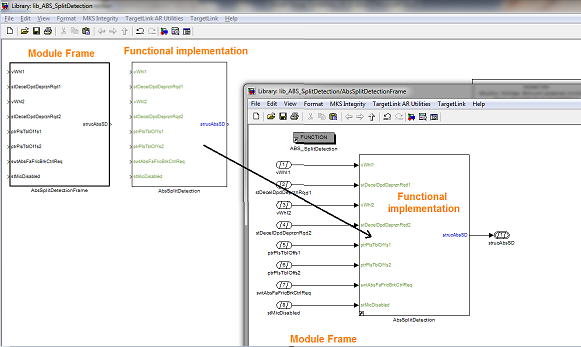
* With the module interfaces specified in the MKS RM document, module frame shall be generated automatically using scripts.
* Refer the below link for more details

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Misc/XLS2DD/project.pj&revision=:member&selection=Workflow_RM_IfSpec.docx>

* The functionality shall be implemented as a subsystem placed at the top level of the library and referred inside the module frame.



Functional implementation linked inside module frame



# Using Simulink or Stateflow for module design

* If the module functionality contains only arithmetic or logical operations, Simulink shall be used for modeling the behavior.
* If the module functionality contains different control states and their transitions, Stateflow shall be used for modeling the behavior.

Example functionality that be implemented in Stateflow:



* If the module functionality contains nested conditions and decisions, Stateflow shall be used for modeling the behavior.

# Signal Naming

Naming of the signals depends on the type of signal.

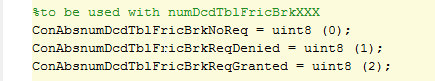
* Parameters
  + - The parameters shall be categorized into normal, derived, learning parameter.

For naming convention for different parameters refer:

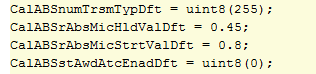
<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Documents/Guidelines/naming_rules/project.pj&revision=:member&selection=MBSP_ConstantsParametersVariables_Summary.ppt>

* Global, Component/Module specific constants and Calibration parameters
  + - All constants and calibration parameters shall be initialized only in the respective ini-script of the module, *<Module>/Common/Config/ini.m*

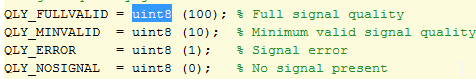
Constants:



Calibration parameters:



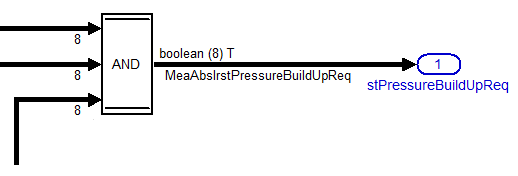
* + - Constants\Calibration parameters that are common to more than 1 module shall be initialized in the ini-script of the component, *<SWC>/Common/Config/ini.m*
    - Constants that are common to more than one components shall be defined as global constant and shall be initialized in the global ini-script, *Simulink\_models/Common<Project>/Config/ini.m*



For naming convention for different constants refer:

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Documents/Guidelines/naming_rules/project.pj&revision=:member&selection=MBSP_ConstantsParametersVariables_Summary.ppt>

* Measurement signals
  + - Measurement points are needed to debug the functionality in case of unexpected behavior / or to observe the internal behavior of a component.
    - Measurement signal lines in a model shall be added with a signal name with a prefix “Mea”



Refer the below link for the naming convention for different types of signals

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Documents/Guidelines/naming_rules/project.pj&revision=:member&selection=MBSP_ConstantsParametersVariables_Summary.ppt>

# Specifying SL data types

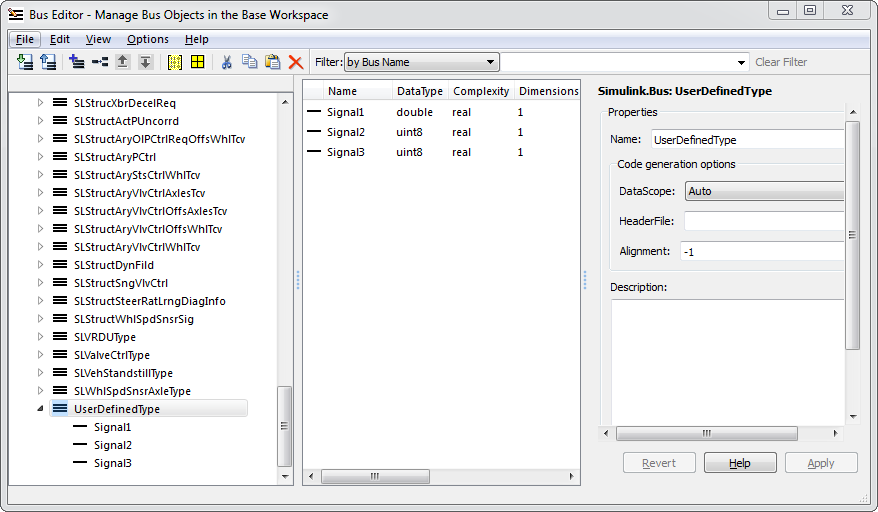
# Specifying data types at the library interfaces

* SL types should be specified at the top level interface of any library.
* In all cases appropriate data types shall be used for the ports. Refer the below document regarding how to specify the data types for different signals.

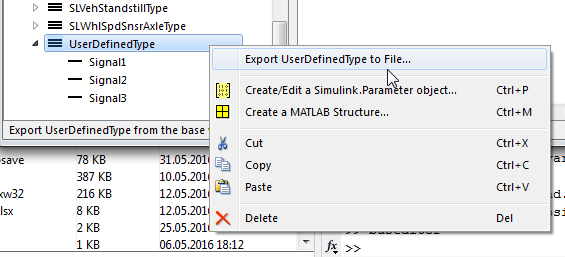
<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D12_Tooling/Documents/SL_TL_handshake/project.pj&revision=:member&selection=SimulinkModelInterface_modellingGuidelines.ppt>

# Specifying data types for Simulink bus port

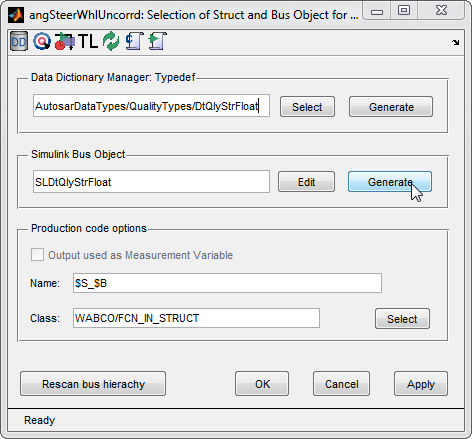
* Similar to scalar signals, data types shall be specified for Simulink bus ports as bus objects.
* User defined Simulink Bus Objects:
  + Simulink bus objects can be created for user defined types using the bus object editor of MATLAB. (use “*buseditor”* command in MATLAB to open the GUI)



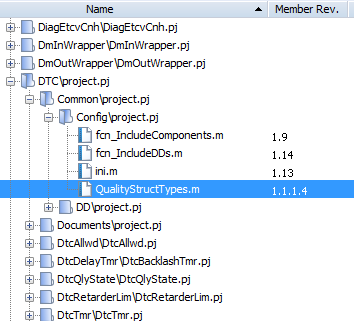
* + After creating the bus objects in the bus editor it can be exported to an m-file



* Simulink bus objects for AUTOSAR types:
  + If AUTOSAR types are available via DD then corresponding Simulink bus objects can be generated using the TL bus port dialogue



* + After creating the bus objects it can be exported to an m-file using the bus object editor
* All bus objects related to a SWC shall be exported as single m-file and placed in the /Common/Config folder of the respective component. The exported m-file shall include the function call *m\_add2MKSList(mfilename('fullpath'))* so that there is traceability to the generated code.



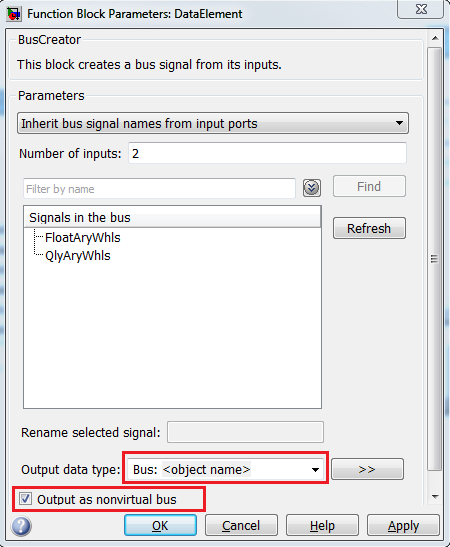
* The function call to load the bus objects into the workspace during model start-up shall be implemented in the /Common/Config/ini.m file of the respective component

# Virtual and non-virtual bus in Simulink

* In Simulink there are two types of buses available, virtual & non-virtual buses.

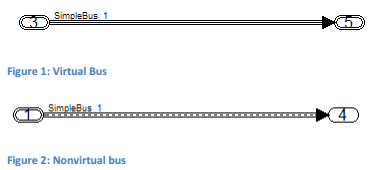
(Example needs to be added stating when virtual/non-virtual buses needs to be used)

* By default the Simulink buses are Virtual.
* A virtual bus can be transformed into an non-virtual bus by providing the bus-object and select the below option.



* It’s advisory to specify a bus object at the origin of a bus, i.e. on a bus creator. If you do so, the bus consistency is checked where the bus is created, not somewhere downstream on some function interface
* Non-virtual bus can be used to initialize bus or synchronize data update on a bus, if SL complains due to various SL execution strategies. See Simulink help for the difference.

* After model update the buses are differentiated as below.



* For TargetLink code generation there is no differentiation between the bus types.
* All TargetLink bus ports will be generated as structure in the code.
* For known problems, Lessons learnt issues please refer the below link

[http://wabco-pe/Method\_Tools/targetlink/MbSWDDocuments/Forms/AllItems.aspx?RootFolder=%2FMethod%5FTools%2Ftargetlink%2FMbSWDDocuments%2FLessonsLearned&FolderCTID=0x012000CF81460658EA67419D9AAE7F7CE94CA6&View={A5B73FFF-ED30-4B49-9723-4A833F4BF6C7}&InitialTabId=Ribbon%2EDocument&VisibilityContext=WSSTabPersistence](http://wabco-pe/Method_Tools/targetlink/MbSWDDocuments/Forms/AllItems.aspx?RootFolder=%2FMethod%5FTools%2Ftargetlink%2FMbSWDDocuments%2FLessonsLearned&FolderCTID=0x012000CF81460658EA67419D9AAE7F7CE94CA6&View=%7bA5B73FFF-ED30-4B49-9723-4A833F4BF6C7%7d&InitialTabId=Ribbon%2EDocument&VisibilityContext=WSSTabPersistence)

* Non-virtual bus can be used to fix the below errors during code generation (TL 3.4)
  + [TL34 Error : The output data type of the TargetLink subsystem’s Outport block ... is equal to 'auto'](http://wabco-pe/Method_Tools/targetlink/Lists/Posts/Post.aspx?ID=3)

<http://wabco-pe/Method_Tools/targetlink/Lists/Posts/Post.aspx?ID=3>

* + [TL34 - Error #01225: An explicit copy of the bus signal driving the Outport block](http://wabco-pe/Method_Tools/targetlink/Lists/Posts/Post.aspx?ID=2)..

<http://wabco-pe/Method_Tools/targetlink/Lists/Posts/Post.aspx?ID=2>

# Specifying data types for blocks

* It is mandatory to specify the appropriate SL types for source block such as constant blocks per the function or logic.
* Rest of all blocks shall inherit the type from the previous block

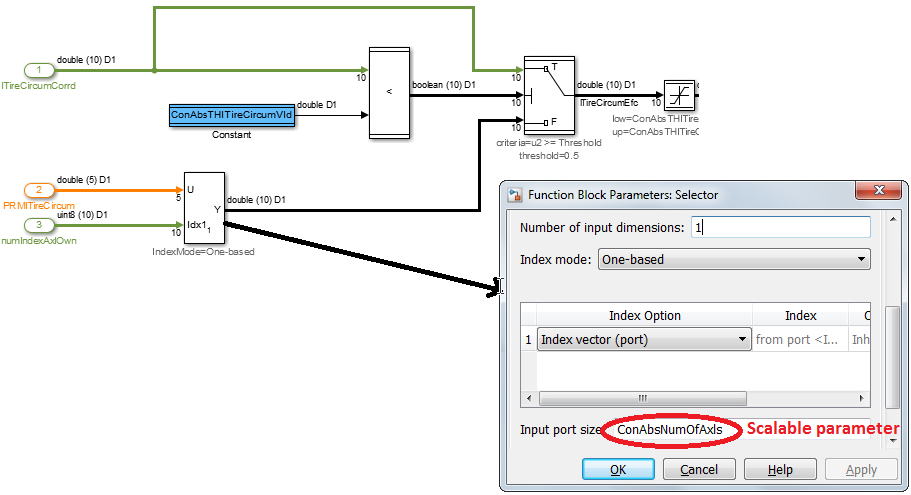
# Scalability of model for different vector/array sizes

* Models processing vectors signals shall be implemented in such a way it is scalable to any vector/array size with minimal implementation effort.

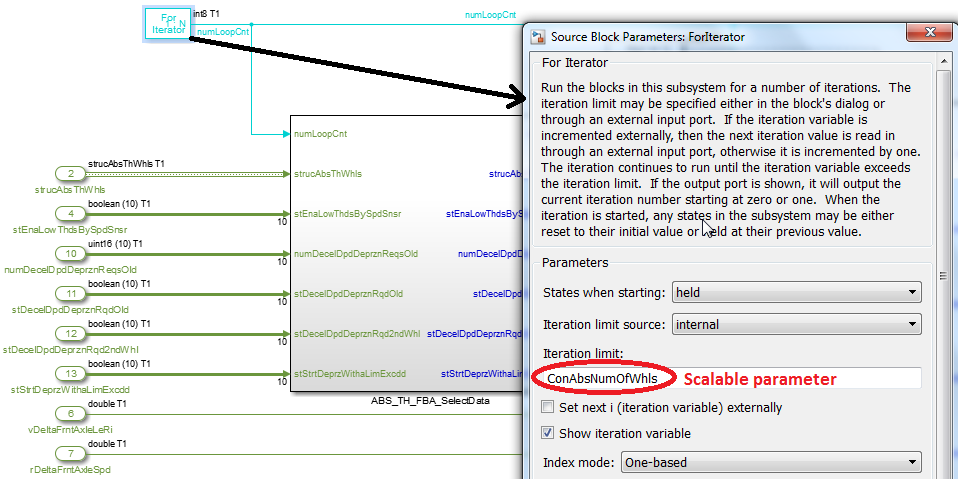
For example, models processing wheel/ axle/ valve specific signals shall be implemented as scalable models

* Scalable Simulink model can be designed using the Inports, Selector blocks, For-Subsystems etc.
* Parameter used for scalability should be declared in the ini.m

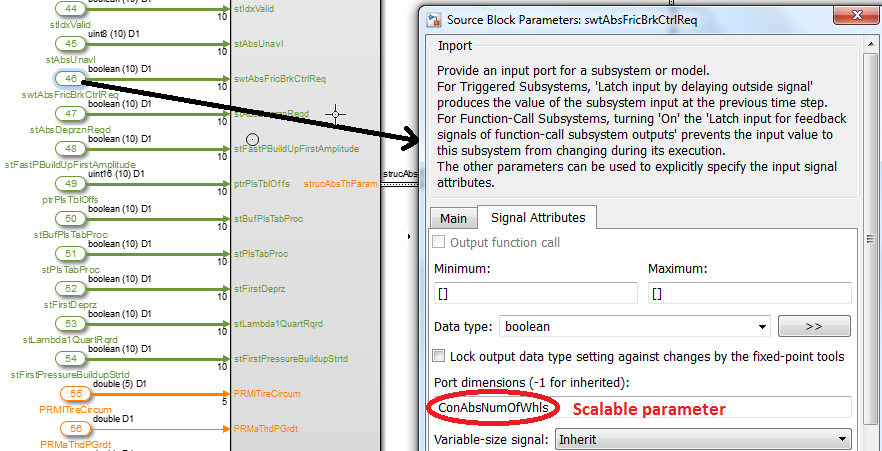
Using Selector Block:



Using For-Subsystem:

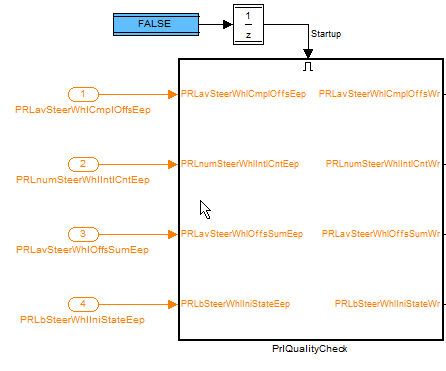


Using In-port:



# Design of initialization functions

* Initialization functions are called only during the start-up only for one time
* Initialization logics shall be implemented inside a subsystem so that in case of a separate Runnable created for initialization function this subsystem can be ported.
* In case if there is no separate Runnable for the initialization function then the subsystem shall be triggered using a unit-delay block with initial value 1.



The above method has the following pros and cons;

This approach has a drawback on RAM consumption, whereas it saves execution time. Depending on the resource situation one or the other implementation may be needed.

This implementation has another drawback: if the execution of the functionality is not disabled at the same time, the execution time is t\_init + t\_function which might lead to a task overload. So even if the execution time after initialization is less, startup itself might be critical.

CONCEPT UNDER DISCUSSION: In case the initialization functions shall be part of SW architecture like mode management in dSPACE SystemDesk, then issue related to run time avoided but there will be RAM consumption.

# Modeling of enabled/triggered subsystem

* Enabled or triggered subsystem are needed to execute some subsystems conditionally
* Refer the below document regarding the usage of enabled/triggered subsystems

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D12_Tooling/Documents/project.pj&revision=:member&selection=EnabledTriggeredFunctCallSubsystems_HowTo.ppt>

# Virtual and non-virtual (atomic) subsystems

* Subsystems can be virtual or atomic. Simulink ignores virtual subsystem boundaries when determining block update order. By contrast, Simulink executes all blocks within an atomic subsystem before moving on to the next block.
* Conditionally executed subsystems are atomic. Unconditionally executed subsystems are virtual by default. But an unconditionally executed subsystem can be configured as atomic in the subsystem parameters.
* Delay blocks used to feedback signals between subsystems shall not be placed inside a non-virtual or atomic subsystems, because while compiling the model the delay block is not visible (since it is enclosed within an atomic subsystem) and so “algebraic loop” error will be reported.

# Maximum depth level of subsystems

* The maximum level of subsystems shall be limited for better readability.
* Subsystems depth up to 9 is acceptable. (limit as per MXRAY)

# Restrictions in for-subsystems usage

* In case of vector implementation, for-subsystems shall be used to optimize the generate code size (ROM). On the other hand usage of for-subsystems may increase the execution time.

Limitations due to code generation issue

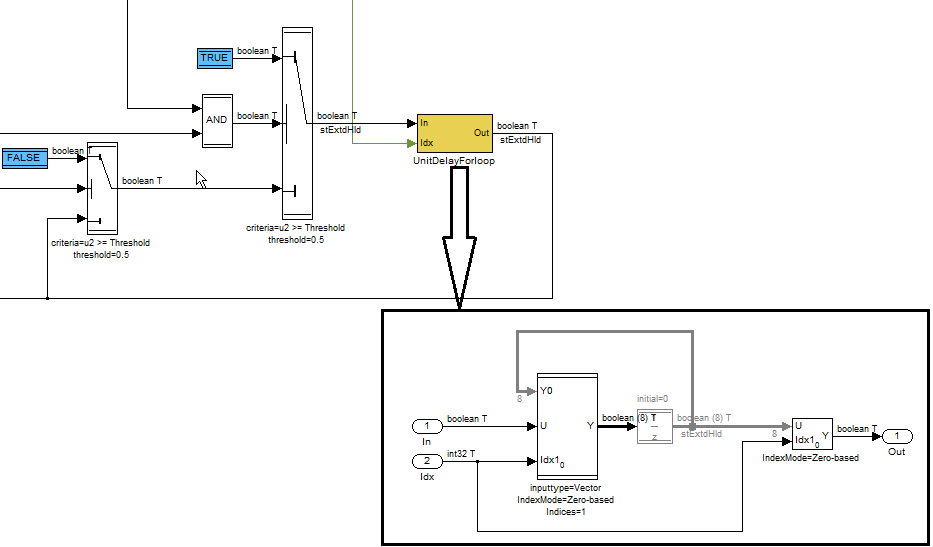
* In case of for-subsystems, Unit Delay blocks shall not be used. Usage of unit delay blocks inside for- subsystems will result in erroneous code generation

(Instead of holding the previous value of the 1st array element for next iteration the code generated such that last array element processed in the last iteration will be taken as the previous value of the 1st array element)

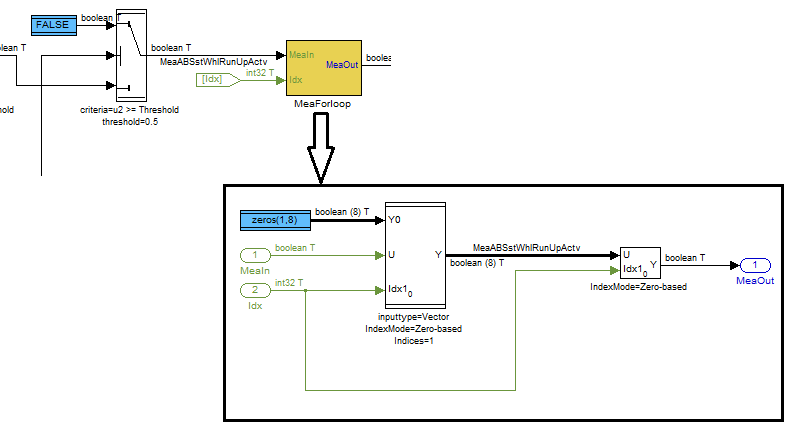
* Measurement variables which are not output of the for-subsystems shall be handled with caution because overwriting of elements of happens similar to unit delay blocks.
* Enable/Triggered subsystems are not allowed inside for- subsystems since the outputs are created as static variables, there is possibility of overwriting values in the next iteration similar to unit delay blocks.
* During code generation TL will generate static variables for output assignment blocks in case the for-subsystem is residing inside a ICG (Incremental code generation) enabled module/sub-module.

Workaround for Unit delays, enabled subsystems & measurement variable inside for-subsystem

* A possible workaround for using Unit delay blocks is to move the delay blocks outside the for-subsystems.
* Enabled subsystems with outports property ‘Output when disabled’ is set to “reset” shall be changed in to IF-ELSE Action subsystem so that the output variable is reset explicitly if the conditions are not true. Only IF-Action subsystem shall not be used because this will create static variables similar to Enable subsystems
* For Unit delay blocks inside a for-subsystem the below implementation shall be used.

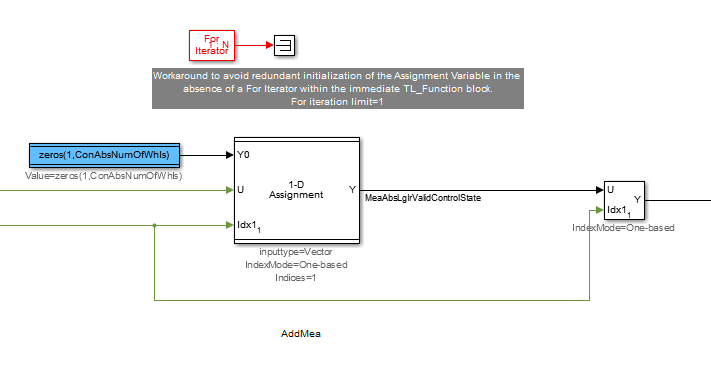


* For measurement variables inside for-subsystems the following implementation shall be used.



If the above workaround is used in sub-module/modules which doesn’t have any for-iterator block in the higher levels of the subsystem (within a TL function subsystem), then the below additional workaround needs to be done.

This is to avoid re-initialization of the measurement variables during every iteration of the for-loop at the higher level.

The For-Iterator should have an iteration limit set to 1 

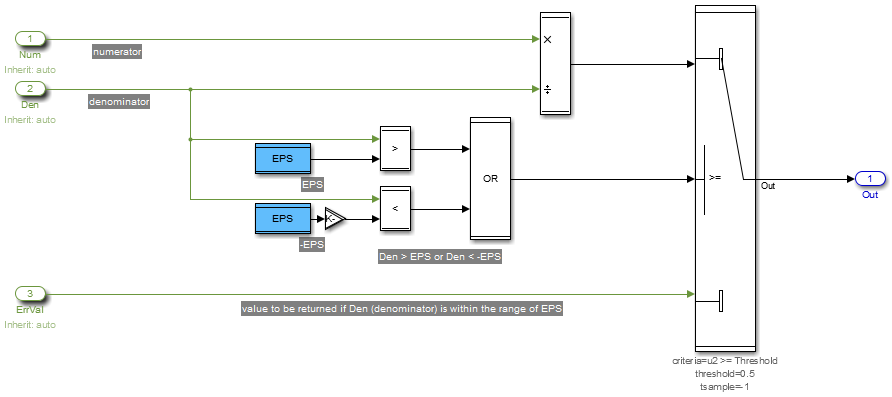
* In case the for-iteration is performed only for selected indices, to avoid static variable generation in the code for output assignment blocks refer the workaround mentioned in the below link.

<http://wabco-pe/Method_Tools/targetlink/MbSWDDocuments/Guidelines/ForLoopsAndAssignments.docx>

# Explicit division by zero protection

* Whenever a division block is used the divisor should always be non-zero
* Separate implementation to achieve non-zero divisor needs to be done.

An example implementation for safe division. This is available as a common library in mBSP project (http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS\_proj/MBSP\_SW\_Components/D07\_Design/Simulink\_models/CmnFcts/SafeDivision/Lib/Lib.pj&projectRevision=1.13&revision=1.7&selection=lib\_SafeDivision.mdl)



* Make sure the “Protection against division-by-zero” is unchecked in the TL divide block properties if it is handled explicitly. This is avoid generation additional code by TargetLink.

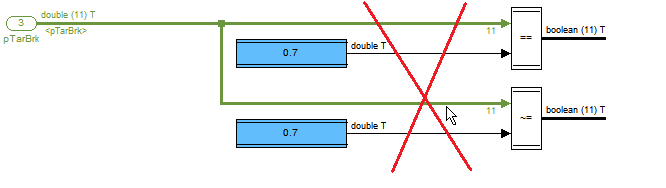
Refer the below link for more explanation:

<http://wdesxwiki/dokuwiki/doku.php?id=process:mbe:sl_bestpractises#division_by_zero>

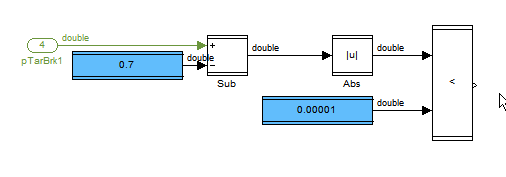
# Float Comparison

* Equality and inequality comparison operators shall not be used for float signals. Since IEEE 754 32 bit floating point algorithm is used for storing the floating point numbers in software, some of the numbers(fractions, large whole numbers etc.) are not exactly representable, hence for floating point values the exact equality/inequality cannot be checked. It will result in unpredictable comparison results.
* Static code analysis tool like QAC, Polyspace will report warnings for the generated code for float equality/inequality comparisons.

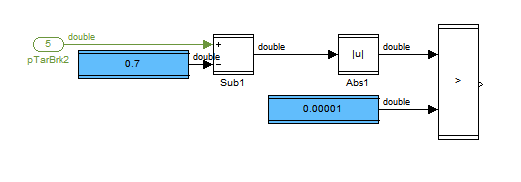
For example: Value 0.7 is not representable in 32-bit IEEE 754 floating point format. It will be stored in software as 0.699999988079071.Same is the case for large whole numbers i.e. the value 129999997 will be stored as 130000000.



Floating point equality can be checked as:



Floating point inequality can be checked as:



For equality comparisons, choosing the constant value 0.00001 in the above model shall be based on the precision required for the comparison.

Some of the fractions can be exactly representable in float64 (SL type: double) but not in float32. Since in Simulink floating point numbers are represented using float64 (double) but in TargerLink floating point numbers are generally represented using float32, this may result in MIL/SIL mismatch. To avoid failures in back-to-back tests always choose a fraction value that can be representable in both format, by adding a little offset (like 0.0000001) to the fractions.

# Proper handling of counter overflows or underflows

* Counters implemented in Simulink or Stateflow shall be implemented with an explicit logic to take care of overflows or underflows

# Creation of module test environment

* A test environment shall be created for the module level library for compilation, testing of the module. The same test environment will be used until MIL-SIL testing.
* Scripts are available to generate test environment model from Library models. Refer the below link for the scripts.

d:/MKS\_proj/MBSP\_SW\_Components/D07\_Design/Simulink\_models/T\_Tools/Utility/TestEnvironment/project.pj

*Note: It’s a good practice to compile the model frequently during implementation. This will save time in removing the compilation errors at the end of the model implementation*.

# Usage of blocks

* Custom made libraries are available for basic functions like filters etc. These libraries shall be used if applicable. <http://wdesxwiki/dokuwiki/doku.php?id=process:mbe:sl_bestpractises#mbsp_common_functions>
* For implementation of simple IF-ELSE constructs with only one control input like:

If (a>0) {b=1;} Else {b=0 ;}, Switch blocks in Simulink shall be used.

* For implementation of complex IF-ELSE construct, State flow charts shall be used (Refer <http://wdesxwiki/dokuwiki/doku.php?id=process:mbe:sl_bestpractises#replacements_for_simulink_if-else_block_construct>)
* Approximate math functions shall be used instead of actual functions. This will reduce the CPU execution time

For example: Trigonometric function shall be approximate as *tan(x) = x + (1/3)x3 + (2/15)x5 + (17/315) x7*

* For integers, use of Square root block shall be avoided because the execution time of the generated code will consume more CPU time. This may be implemented using simple blocks.

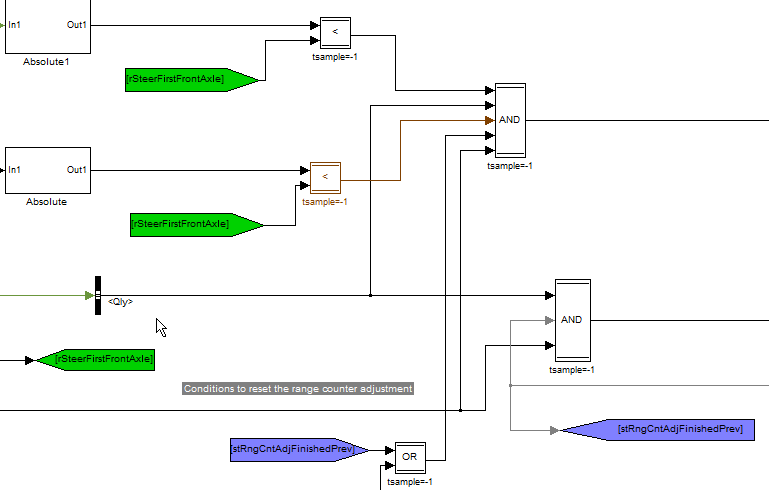
For math functions it is recommended to run PIL testing and find the change in execution time

* Crossing of signal lines can be avoided using GOTO and FROM blocks
* However, unnecessary usage of GOTO and FROM blocks shall be avoided since:

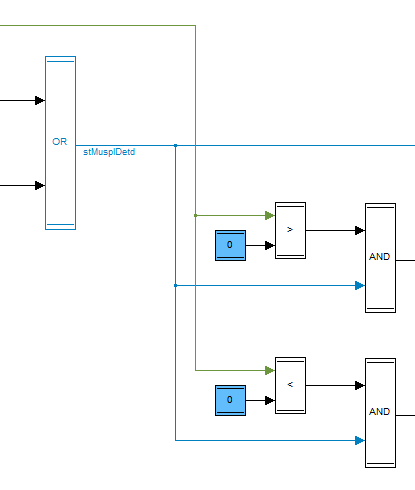
- It will affect the readability of the model signal flow

- The complexity of the model will be more

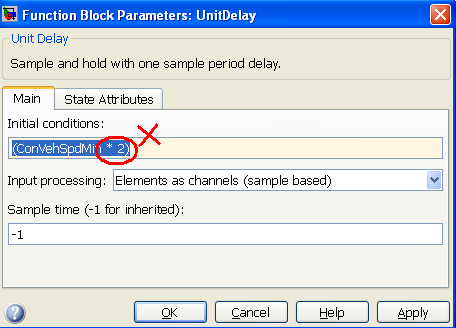
* For GOTO and FROM blocks with long names the block size should be such that the signal name is visible.
* It is advisable to use different background colors for multiple GOTO and FROM pairs. The background color of GOTO and FROM blocks shall not be same as the color of other standard blocks like Constants, Calibration constants etc.

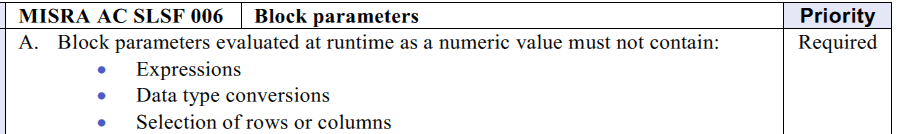


* For better readability in case of line crossings the foreground color of the source block shall be changed as below.

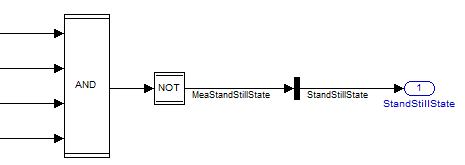


* No arithmetic calculation or functionality should be hidden in any of the block dialogues as per MISRA rules





* Use MUX blocks to rename signals as this will not increase the complexity and also has less risk of providing wrong TargetLink data types.
  + Use of gain/datatype conversion blocks will increase the complexity
  + Signal name propagation will not happen if a MUX block is used



* Care shall be taken when TargetLink blocks from a library are copied to another library. This operation will create a link of the copied blocks to the source library which has to be broken.

# Adherence to naming guidelines

* It’s strictly advisable to follow the naming conventions described for the project.
* The naming of the ports shall be consistent across subsystem. Run *CheckSignals* script to check the naming consistency and correct the failed checks.

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Simulink_models/Common/Config/project.pj&revision=member&selection=CheckSignals.p>

The report of *CheckSignals* shall be checked-in into “*Documents/ModelReview”* folder of the corresponding module or component.

* Signal propagation on the signal lines helps in better understanding of signal flows.

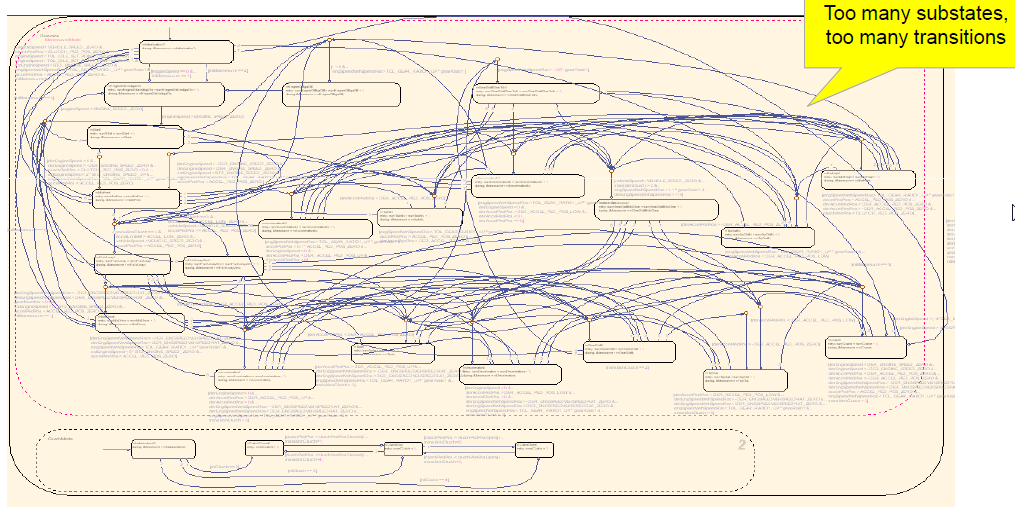
<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Documents/Guidelines/modelling_rules/project.pj&revision=:member&selection=MBSP_SignalNameAndPropagationGuide.ppt>

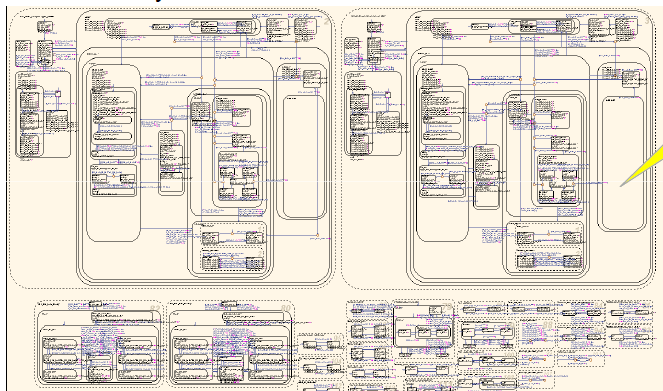
# Stateflow design

# Stateflow with states

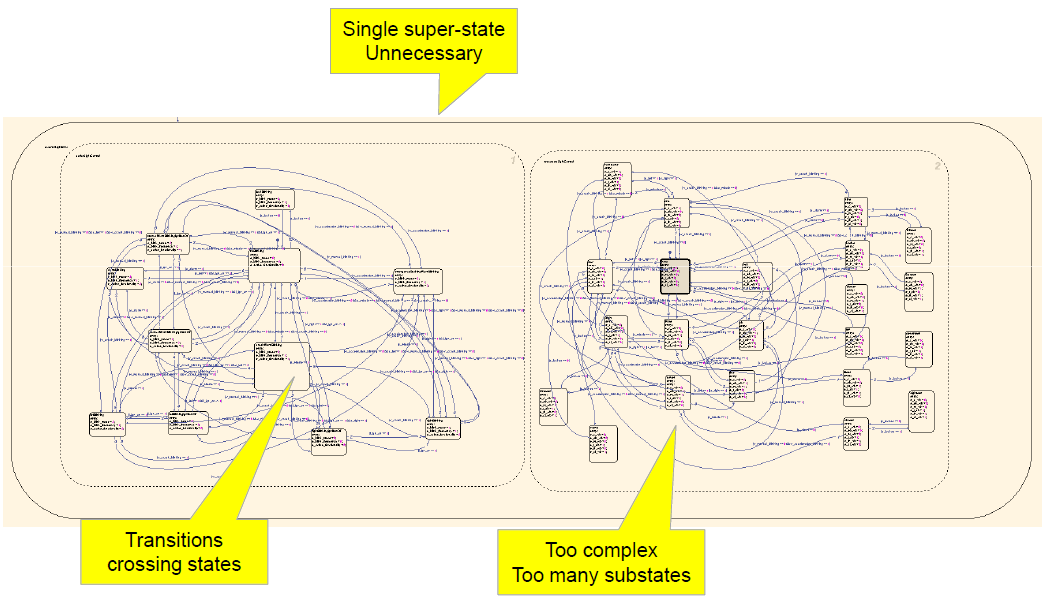
# States and transitions

* Too many states or transition shall be avoided. If there are too many states or transitions, then the State chart becomes very hard to read and hence very hard to maintain





* Crossing of transitions shall be avoided. This affects readability of the state chart

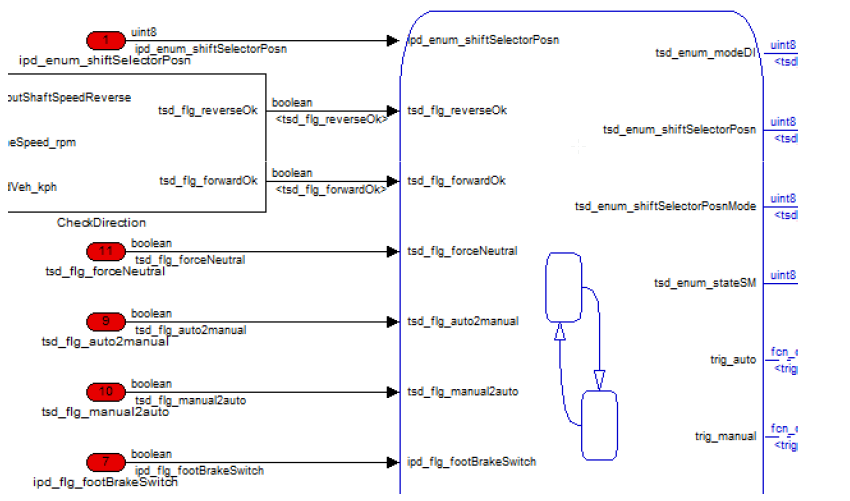


* A state chart with a single state is avoidable.

# Interface signal naming

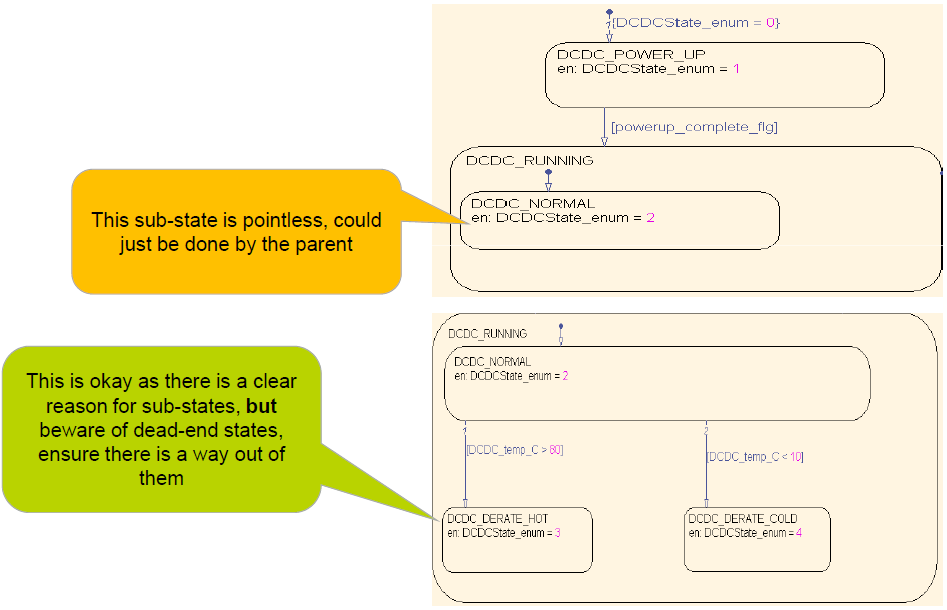
* Signals at Simulink/State flow interface shall not be renamed. i.e. refer to the same signal names in Stateflow as you would in Simulink.

*Note: Renaming the signals at the Stateflow chart interface will create additional signals and assignment operations in the generated code.*

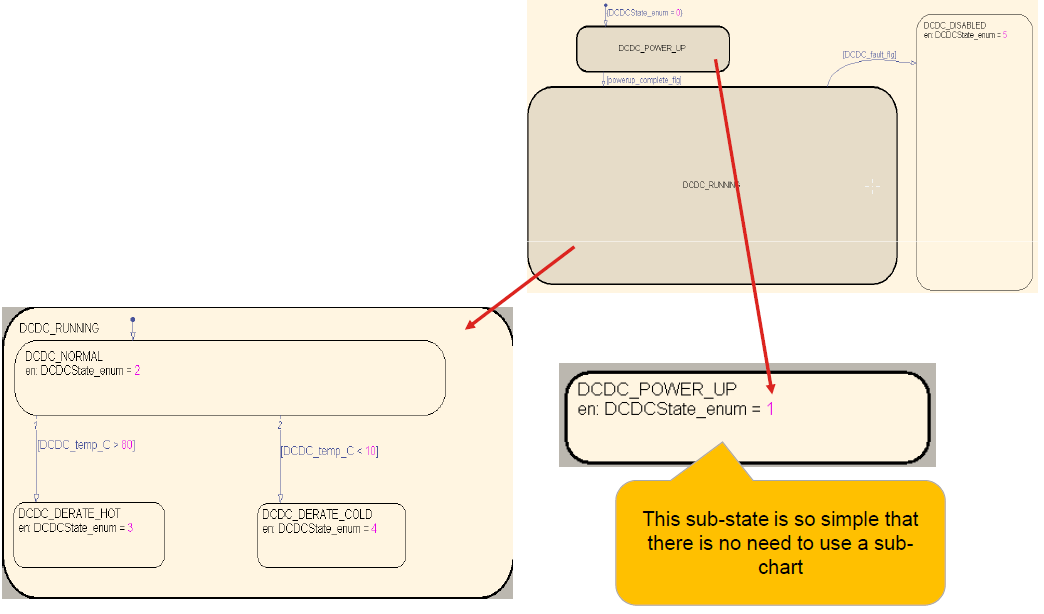


# Basic usage of states

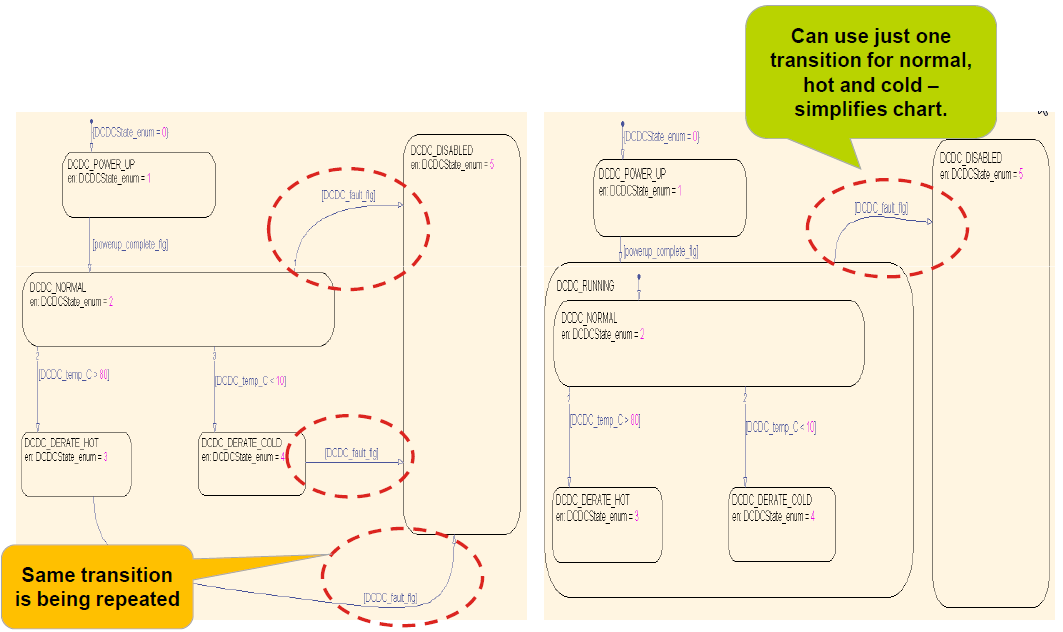
* States shall have either no sub-states or more than one sub-state



* Sub-charting (hiding contents of a chart) shall be used only when it is needed to make top-level chart more readable. In the below example a chart with sub-states has been sub-charted.



* More than one level of sub-charting shall be avoided. Otherwise charts become too complicated
* Super-states shall be used to simplify charts wherever appropriate



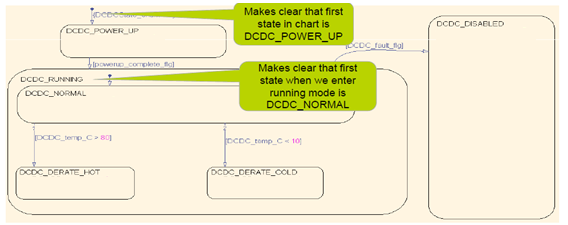
# Default transitions

* Every chart shall have a default transition
* A parent state containing exclusive sub-states shall have exactly one default transition

*Justification:*

*– Makes it clear for any chart and sub-chart what is the first state to be entered*

*– Consistent layout between engineers*



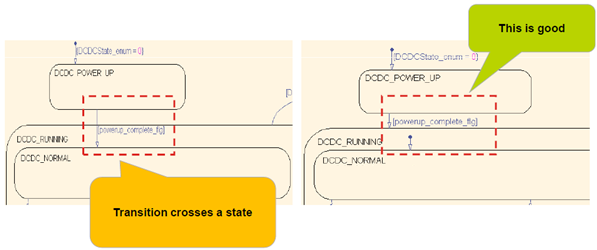
# State transition rules

* Transitions shall not cross state boundaries from a higher level parent state to a lower level child state; Default transitions shall not be avoided for state with sub-charts.

*Justification:*

*– Generally keeps charts visually simpler*

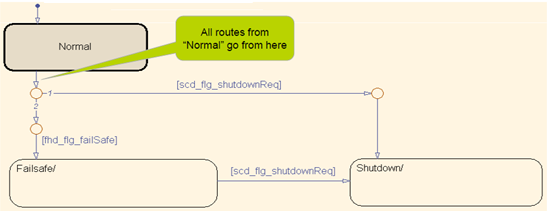
*– Overrides any default transitions (visual equivalent of a ‘C’ goto)*



* State transition condition check shall be always outside the states
* One exit transition from a state can also be used.

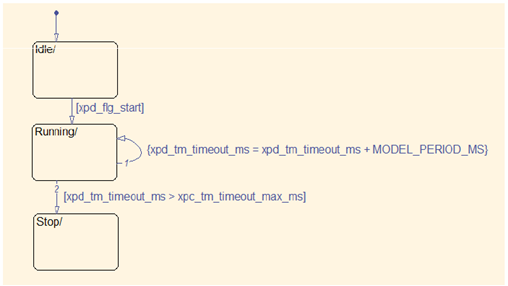
– Easier to read if all routes out of state are in one place

– The evaluation order of the transitions can be defined explicitly by the flow diagram

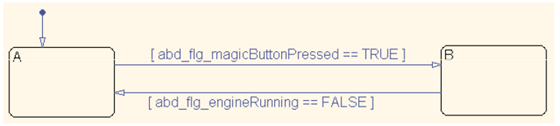


* Loop-back external transitions shall be avoided. If the transition action is true then they will exit and re-enter the state. So “exit” actions and “entry” actions will be run, “during” actions

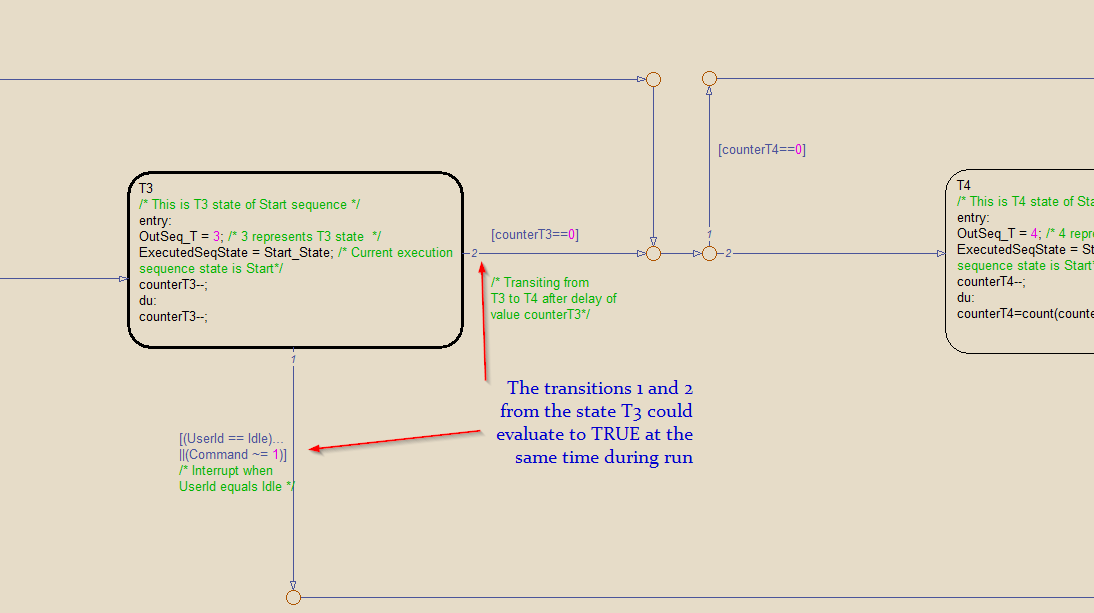
Example: This loopback transition should have been on internal flow diagram



* User definition of transition execution order from junctions and states must not be enabled as per MISRA AC SLSF 034. Condition priority needs to be explicitly modeled.
* Transition conditions between states shall be mutually exclusive. In case of chart that contains transitions between two states with transition conditions that are not mutually exclusive may end up ‘bouncing’ between states on successive controller ticks

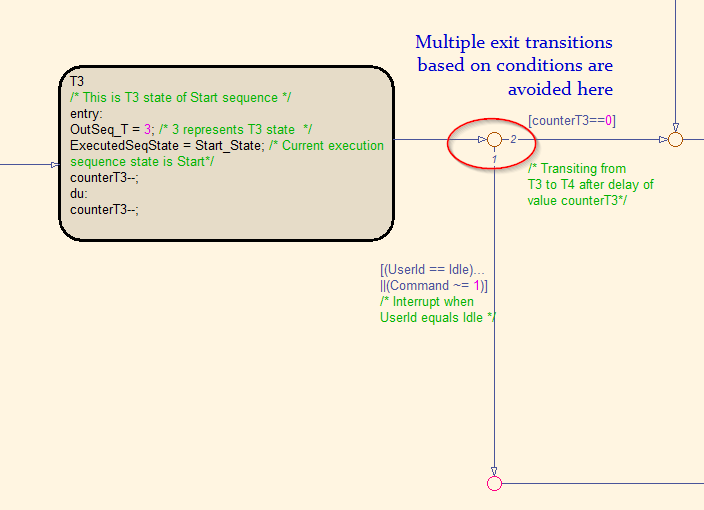


* Stateflow normally evaluates for a state all its exit transitions. So if during run time that it may be true that more than one exit conditions may evaluate to true introducing a conflict during simulation in MIL mode.



Thus it is upon the developer to ensure that the state transitions are modelled in such a way to avoid these conflicts. Following is one such example to avoid these conflicts.

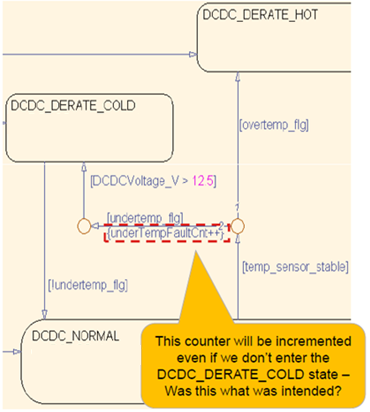
Here the state is having only one exit transition that leads to a common node from where the transition conditions are evaluated.



* Avoid actions on transitions between states, including Transition actions and Condition actions because :
  + Difference between transition and condition actions often poorly understood
  + Avoids unexpected behavior due to backtracking
  + if really necessary to use actions on transitions between states then:

– Only use condition actions

– Ensure backtracking is considered



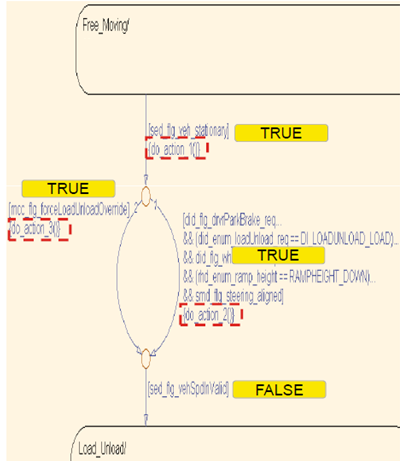
In the example here the following actions will be executed:

– do\_action1()

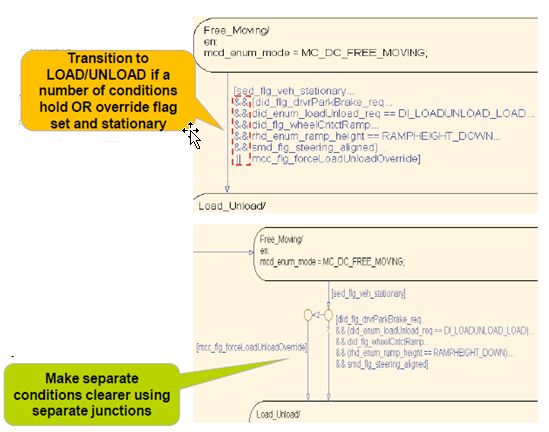
– do\_action2()

– do\_action3()

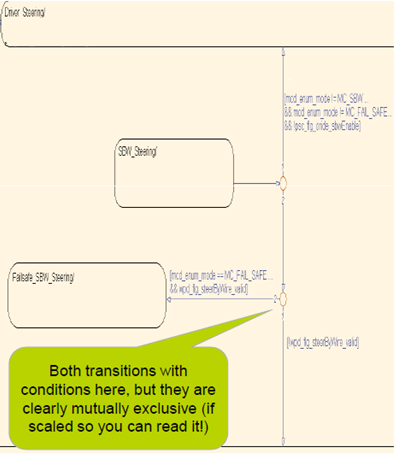
This is due to backtracking. It is obvious that both do\_action2() AND do\_action3() could both execute in same step.



* The conditions on transitions shall be kept simple. Consider calculating complicated transition conditions in Simulink and input the result to the chart.
* Mix of operators && and || shall be avoided. OR transitions shall be made more readable by indicating as separate transitions



* At each junction where a decision on transition route is required, ideally, there should be one conditional transition and one unconditional transition
* Junction decisions shall be chained together where greater than two transition routes are required. This makes junctions simple, readable and priorities of transitions are made clear



* The history junction shall not be used.

*Note: Its behavior is not intuitive (overrides default transitions) and also not easy to see in a large sub-chart*

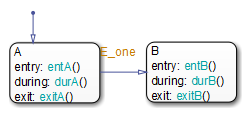
# Avoid using “Entry”, “During”, “Exit” state action keywords

* Usage of “Entry” & “During” keyword shall be avoided. The background behavior of these keywords is always hidden.

*For example “Entry” is executed only once when the state becomes active; “During” is executed only after the state becomes active.*

* “Exit” actions shall be avoided

– Consider if the action you think of making on exit from a state shall actually be taken as an entry action of the destination state(s)



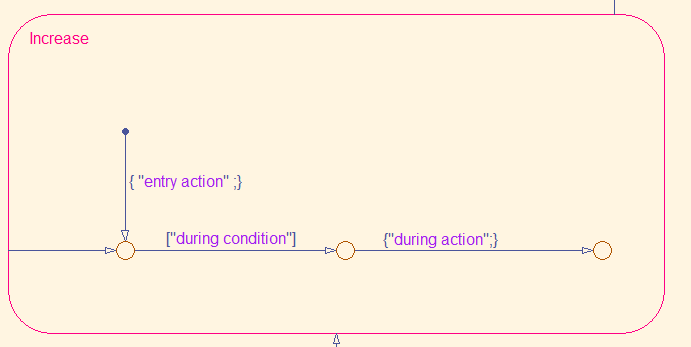
# Flow within states

* Only entry and during actions are shall be modeled inside a state.
* Flow diagrams within a state shall be used to perform actions:

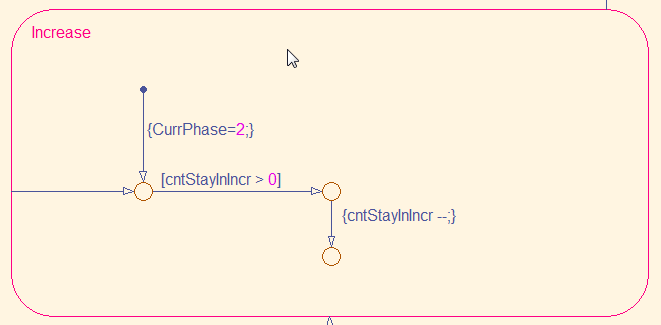
– On entry to states

– During a state

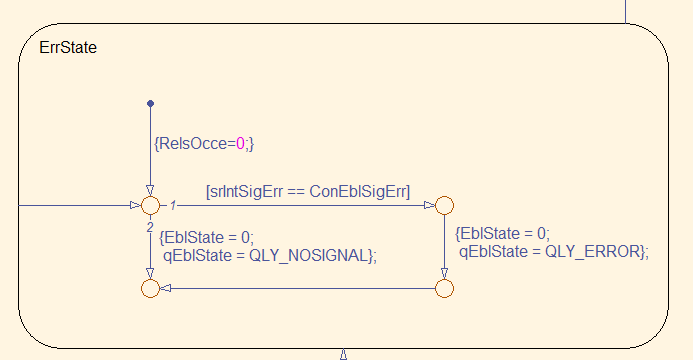
* Entry action shall be indicated in a vertical way
* During action shall be placed from the left-hand-side of the state



Example1:

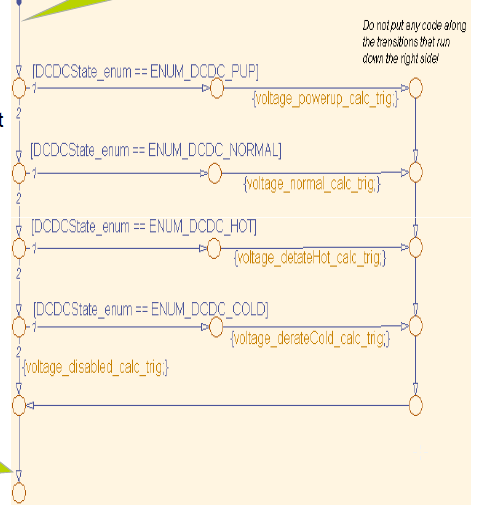


Example 2:



# State flow without states (flow charts)

* Flow-charts shall have single entry and exit point.
* Flow shall be indicated left-to-right, top-to-bottom.
* Straight lines (horizontal and vertical) shall be used mostly except for cases that demands otherwise (e.g. for loop)



* Stateflow provides a number of built in patterns for common constructs. Can be helpful as a quick starting point for your flow diagrams. Using the same patterns across the teams means engineers can immediately understand the purpose of the chart just by the “shape”.
* If…Else constructs:

For if..Else construct probably better to model in Simulink with “switch block”.

However, Stateflow construct only executes one path or the other

– Switch block may execute both paths

– Useful to know if e.g. trying to protect against division by zero

* If… else if... else constructs:

Use Stateflow for complex structures, because it is clearer than Simulink. If there are more than 3 else if conditions use Stateflow.

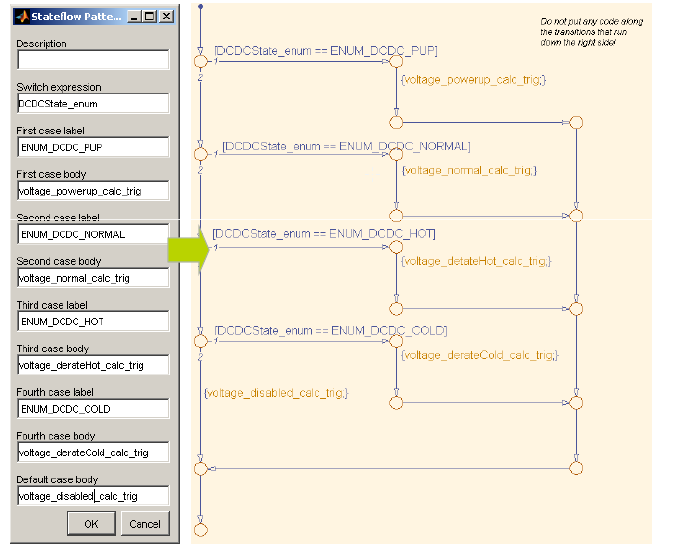
* Switch case construct:

Equivalent to “switch case” in ‘C’ language. Built-in Stateflow pattern shown below.

– Conditions (each “case”) on horizontal

– Actions on vertical

– Includes a default case



* For loop :

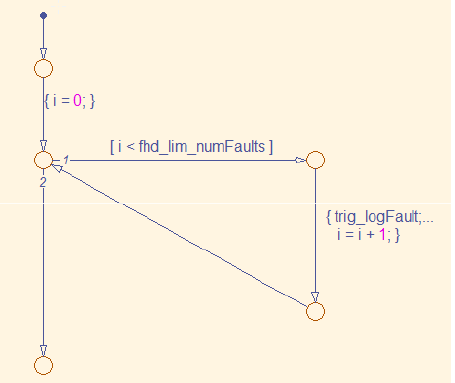
Equivalent to “for” loop in C. The “for” block in Simulink is not that intuitive to use

*for(i = 0; I < fhd\_lim\_numFaults; i++)*

*{*

*trig\_logFault();*

*}*



# Stateflow with graphical functions local to a chart

* When some logic is reused more than once inside a Stateflow chart, then a function shall be created inside the chart. This function shall be called in all other places where this logic is required.

The procedure to create a function in a Stateflow chart is as below:

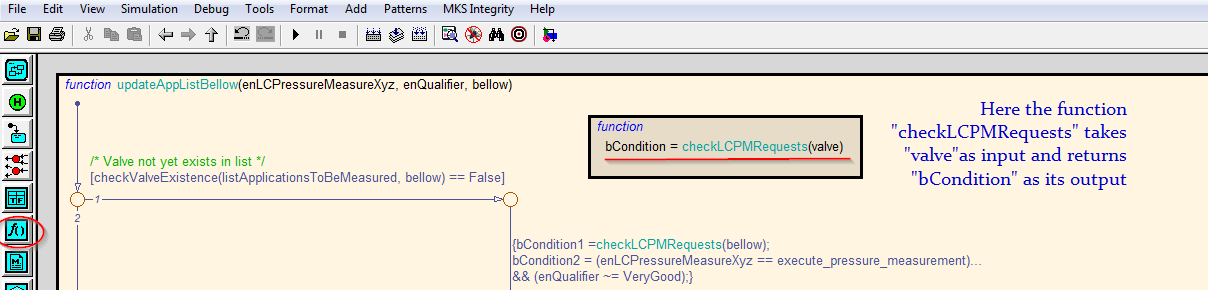
* Drag and drop the highlighted function block from the Stateflow block palette located in the left of the chart.
* Name the function.
* Right click inside the function and select “*MakeContents🡪Subcharted*”.
* Add the input and output to the function in the model explorer.
* Now add the required logic inside the Stateflow chart.
* The function shall have a meaningful name in camel case without any additional space which represents the functionality.

* It shall explicitly specify what the input and output arguments are.

*(During TL code generation this will avoid generation of static variables)*

* If a constant value is shared between more than one function in the Stateflow then this constant shall be defined in the top level of the Stateflow chart.
* Inside the sub-chart there shall be adequate comments to understand what is the actual functionality performed by it.
* Function reusability needs to be checked

An example

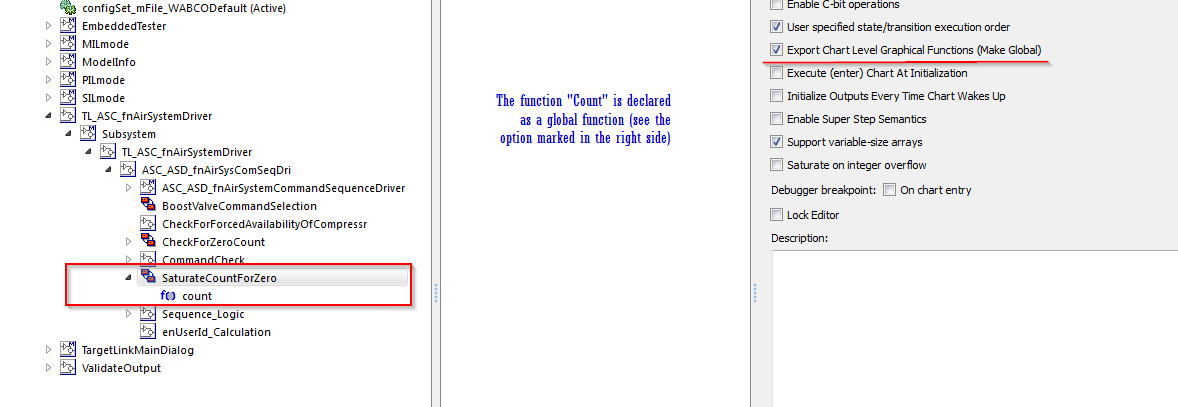


# Stateflow with global graphical functions

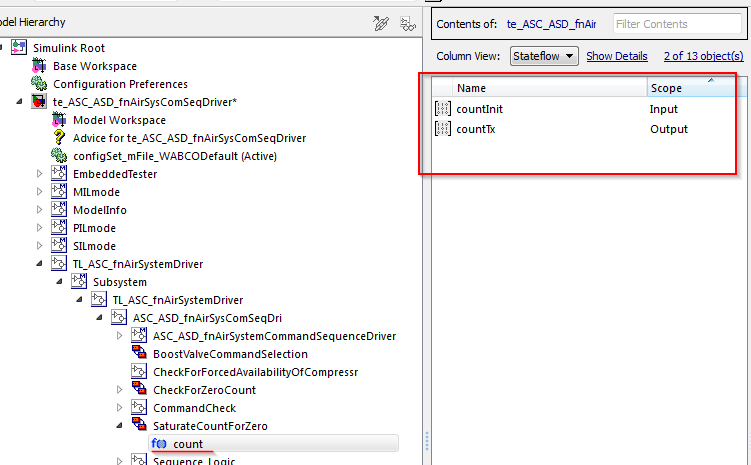
* When some logic is reused in more than one state flow chart then a common graphical function shall be defined and this function shall be called in all other Stateflow charts wherever needed.

Below is an example of how to create such reusable graphical functions:

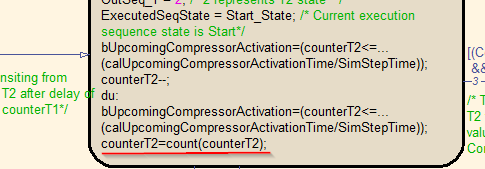
* In this example the function “Count()” is used in more than one Stateflow chart.
* So a separate Stateflow chart with the name “*SaturateCountForZero*” has to be created without any inputs and outputs as below.
* In the model explorer, click the function Count and enable the setting “*ExportChartLevelGraphicalFunctions*(Make Global)” as below:



* This Stateflow chart shall present in a level above or equal to the charts which uses the function.
* The arguments of the function shall be created as inputs and outputs of the function as below:



* Now this function “count” shall be directly called in other Stateflow charts as in the below example:

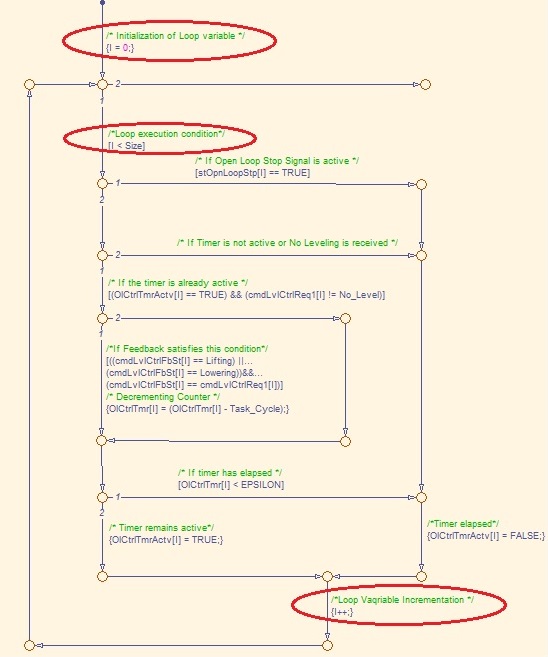


* Function reusability needs to be checked

# Vector signal processing in Stateflow

* In Stateflow chart (for Matlab R2012a and below), a ‘For Loop’ is required to iterate through each element of the array.

*This is because, until MATLAB R2012a only C language style is supported. So to work on vector inputs, using 'For loop' is the only way of doing the operation.*



* The width of the vector input used in ‘For loop’ condition shall be given as a workspace variable.

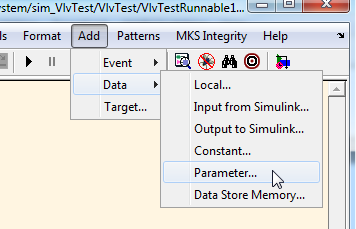
# Usage of intrinsic operators

* Usage of any intrinsic operators like “*hasChangedFrom*”, “*hasChangedTo*”, etc. in Stateflow shall be avoided.

Some operators may cause MIL/SIL mismatches

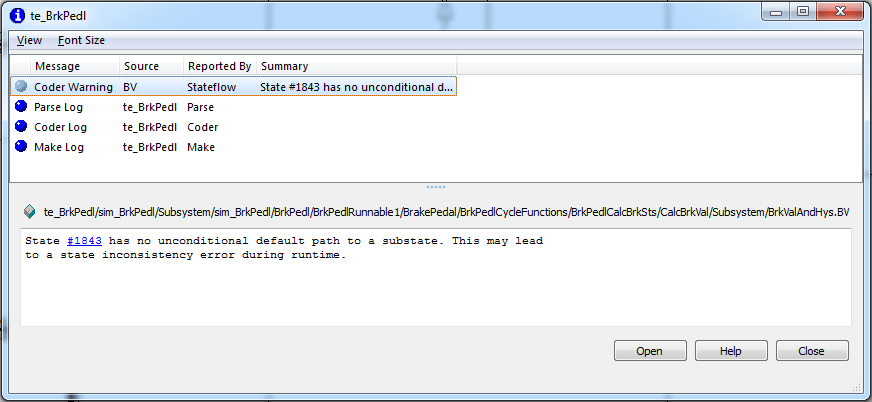
# Calibration parameters inside Stateflow

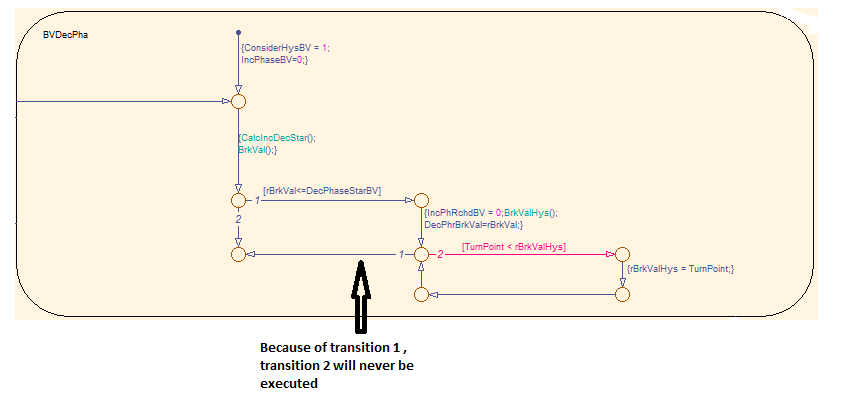
* Calibration parameters for usage inside Stateflow shall be defined as “Parameters”



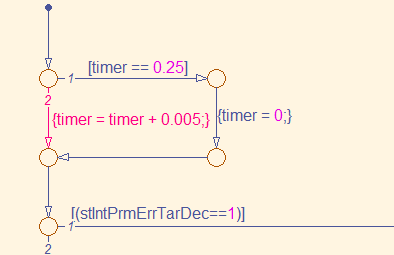
* Simulink functions inside State flow chart shall be avoided, since it is not supported by TargetLink

* It is advisable to have unconditional default path to avoid “State Inconsistency” warning





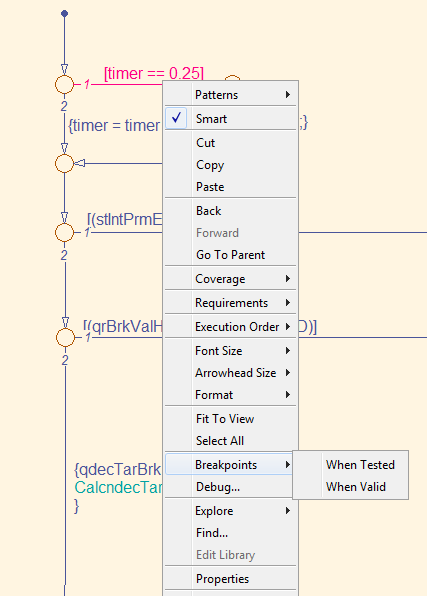
* Since Stateflow is event based, time based breakpoint is not possible with Stateflow, hence timer implementation with some threshold check shall be used to generate event based breakpoint.



The above example is to stop the simulation at 0.25s. The breakpoint can be set for the threshold check [timer == 0.25] in two ways 1.When Tested 2.When Valid

When Tested - It will break the simulation every time the condition [timer == 0.25] is checked.

When Valid - It will break the simulation only when the condition [timer == 0.25] results TRUE.



# Adherence to coloring guidelines

* Block color rules help in making the models uniform in terms of purpose of the block and improves understandability
* Certain colors are reserved for certain block types.
* For color guidelines refer the below link:

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D07_Design/Documents/Guidelines/modelling_rules/project.pj&revision=:member&selection=ColorCode_MBSP.xls>

# Adherence to standard modeling guidelines

MXRAY:

* While dividing a component into modules or modules into sub-modules the complexity of the module/sub-module shall to be taken into account.
* The future development of the model shall also be considered during the partition. If the module is going to be expanded in future then it shall be divided into sub-modules beforehand.
* Avoid module design with model complexity in the yellow threshold, because future development may push the complexity over to red threshold
* Run MXRAY to check the complexity of the model. If the module is too complex then complexity shall be reduced.
  + Incase local complexity of subsystem is more than the acceptable range then group the blocks into the subsystems. The blocks shall be grouped functionally.
  + Random grouping of blocks to subsystem to reduce complexity shall be avoided. This is a compromise in readability and maintainability of models.
  + Use of libraries for repeated subsystems will reduce the complexity. They are only counted once when calculating total complexity in every subsystem that contains them.

Find below the tables with the blocks and their standard weights which are used for complexity calculation.

(M:\MXray\MXRay\_1\_4\MXRAY\_User\_Guide\_v1\_4.pdf)

The report shall be stored in “*Documents/MXRAY”* folder of the respective module or component with the name “*te\_<ShortName>\_MXRAY.html”.*

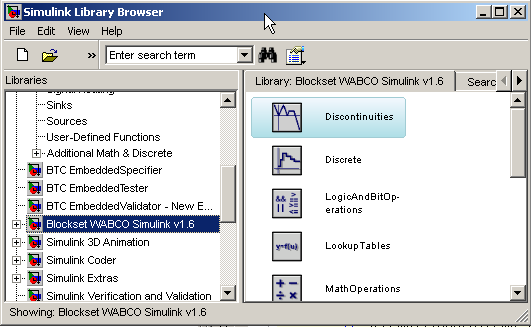
*‘ShortName’ refers to component or module short names*

For more info refer: (for mBSP)

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D13_Tests/project.pj&revision=1.1&selection=TStrat_MBSP_SW_Components.docx>

MXAM:

* Always use the Simulink blocks from WABCO block set. Blocks under this group are pre-configured to qualify MXAM checks.



* If “*Blockset WABCO Simulink v1.6*” is not visible in the simulink browser then refer the below document (from page 26-31) for steps to configure it.

[http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS\_proj/T3\_public/Tools/MbE/Matlab\_Simulink/Documents/Installation/project.pj&revision=:member&selection=Installation MatlabR2011b\_and\_others.pdf](http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Documents/Installation/project.pj&revision=:member&selection=Installation%20MatlabR2011b_and_others.pdf)

* Make sure your MXAM configuration done properly as suggested by project configuration team:

<http://wdesxwiki/dokuwiki/doku.php?id=process:mbe:mxam_bestpractices>

* Run MXAM checks often when there is considerable change(addition of more blocks) in the model
* Ignore list shall be used to ignore MXAM checks for blocks which are outside the code generation unit.

The HTML overview report shall be stored in “*Documents/MXAM”* folder of the respective module or component with the name “*te\_<ShortName>\_MXAM.html”.*

*‘ShortName’ refers to component or module short names*

The ignore list shall be stored in “*Documents/MXAM”* folder of the respective module or component with the name “*te\_<ShortName>.igl”.*

*‘ShortName’ refers to component or module short names*

For more info refer: (for mBSP)

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/MBSP_SW_Components/D13_Tests/project.pj&revision=1.1&selection=TStrat_MBSP_SW_Components.docx>

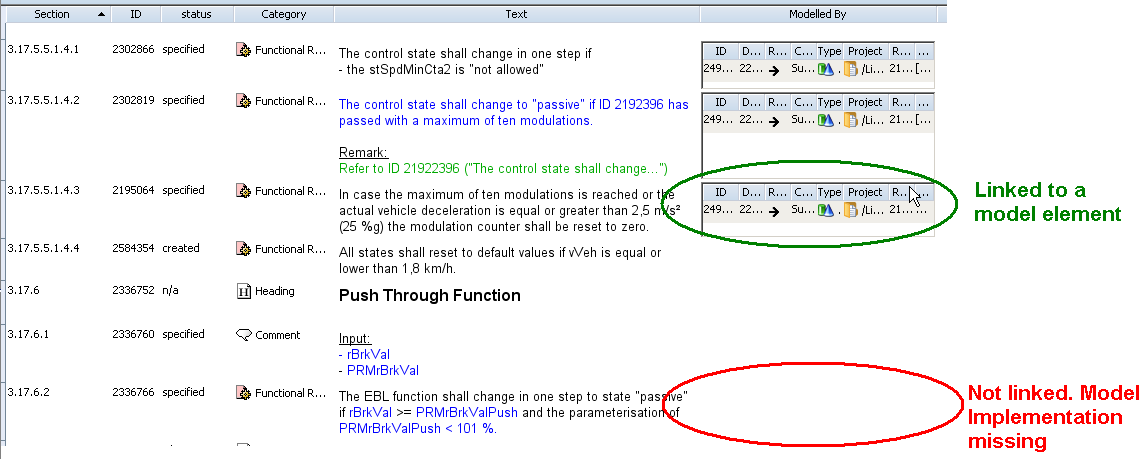
# MKS revision inside models

* All libraries and test environment model shall include the “Model Info” block at the top-level
* The “Model Info” block is used to trace the revision of library used for code generation
* Refer the below link regarding how to use and configure the “Model Info” block

<http://wdesxwiki/dokuwiki/doku.php?id=process:mbe:sl_bestpractises#mks_version_control_of_simulink_and_targetlink_models>

# Linking of detailed design specification in MKS to the model

* Linking detailed design specification to the model is more important since it improves the traceability of design specifications which are not implemented in the model
* The design specification which does not have an implementation shall be easily captured using MKS RM



* Linking of specification shall be done on subsystem level.
* For Stateflow charts the linking of specification shall be done inside the chart – for states and transition etc.
* Refer the below link for installation of MKS integration for linking of specification to model

[\\sclsfs2\mxam$\MKSIntegration\MKSIntegration.ppt](file:///\\SCLSFS2\SW_Centre$\mxam$\MKSIntegration\MKSIntegration.ppt)

* For more information regarding the MKS integrations, issues and solution, refer the below document

<http://170.205.207.224:7001/si/viewrevision?projectName=d:/MKS_proj/T3_public/Tools/MbE/Matlab_Simulink/Misc/MathworksIntegration/project.pj&revision=:member&selection=MKS_Mathworks_Integration.docx>

Figure 7.5c: Linking between model and test cases 2

# Guidelines for SL-TL handshake

* In case if the function and SW developments are handled in different teams\groups, then Simulink changes made by the function developer shall be highlighted, so the software developer could identify the changes for TL implementation.
* Simulink changes shall be highlighted by setting the block “Background” color to “Magenta”

